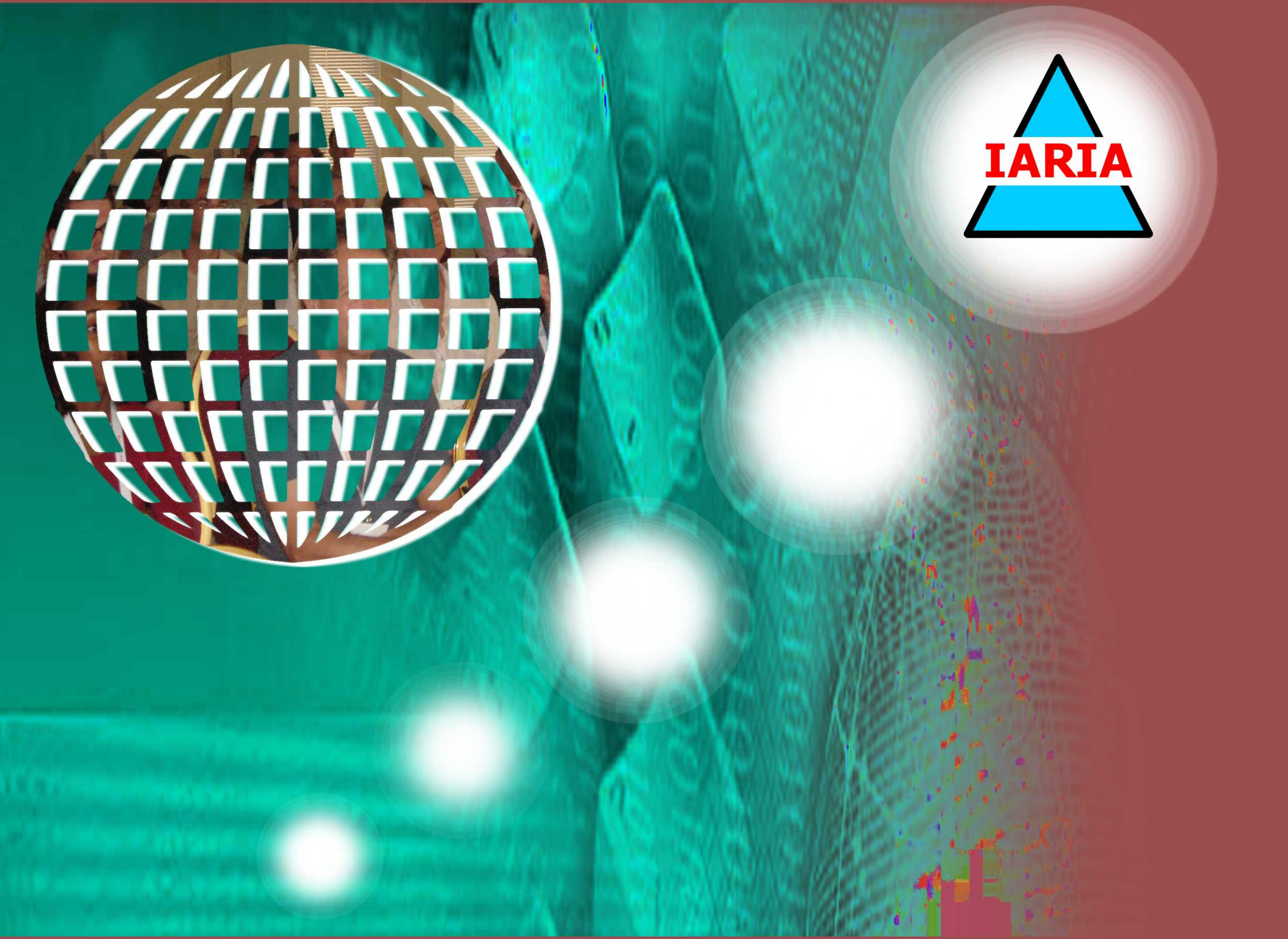
International Journal on Advances in Software



2025 vol. 18 nr. 1&2

The *International Journal on Advances in Software* is published by IARIA.
ISSN: 1942-2628
journals site: http://www.iariajournals.org
contact: petre@iaria.org

International Journal on Advances in Software
Volume 18, Number 1 & 2, 2025

International Journal on Advances in Software
Volume 18, Numbers 1 & 2, 2025

CONTENTS

A Lightweight Web Component Toolbox for Database-Driven Web Applications

Andreas Schmidt*‡ and Tobias Münch†§
* University of Applied Sciences
Karlsruhe, Germany
Email: andreas.schmidt@h-ka.de
‡ Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: andreas.schmidt@kit.edu
† Münch Ges. für IT Solutions mbH, Germany
Email: to.muench@muench-its.de
§ Chemnitz University of Technology, Chemnitz, Germany

***Abstract*—Creating, content editing and interacting with relational databases in web applications has traditionally required developer knowledge of languages such as JavaScript or PHP. We present a lightweight, framework-independent toolbox for database-driven web applications. It provides intuitive database visualization, querying, and editing directly in a web browser—requiring only HTML knowledge. Unlike traditional frameworks like Angular or React, our web components simplify database interaction using a thin REST-based access layer. This work extends prior research by detailing the architecture, integration challenges, and considerations of extensibility.**

Keywords*-*Web Component; Relational-Database; Interface; Prototyping

I. Introduction

This paper is an extended version of a conference paper [1], published in 2024 at the Sixteenth International Conference on Advances in Databases, Knowledge, and Data (DBKDA-2024) conference in Athens/Greece. In this extended paper, we examine the components we developed and describe their interactions in greater detail. We have also added an example scenario showing the components' use in a simple real-world scenario. In a detailed discussion, we also address current work in the area of security and authentication, as well as the expansion of the functionality of our components. We have also continued to work on linking the components to each other and to the other elements within the website. This means that the parameters of the components can now be read from the values of other components or HTML elements and monitored for changes. We have implemented a similar mechanism for the implementation of parameterized SQL statements.

With digitalization, there is increasing demand for user-friendly database interaction tools that support collaboration [2]. Traditionally, developers use frameworks such as Angular or React to create dynamic, database-driven web interfaces [3]. However, they also introduce noteworthy complexity, requiring developers to learn and maintain codebases, often leading to vendor lock-in [4], [5]. Non-developers can not cope with this complexity. The need for a lightweight and straightforward approach outranks the benefits of full-scale frameworks for scientific applications and internal business tools. For example, Microsoft Access has enabled user-driven database-driven development with minimal programming effort [6]. However, its desktop-based nature limits its accessibility and collaboration capabilities in modern web environments [6]. Additionally, in many companies, there is a shortage of suitable IT personnel and limited financial resources for IT services [7], [8]. The shortage of software engineers in particular poses a major challenge for companies [8], as they are needed to create and maintain large, complex software solutions. Reacting quickly to changing markets and optimizing processes are important criteria for remaining competitive. One way of overcoming these challenges is to expand the circle of potential developers. This is the path taken by *no-code* or *low-code* applications. One problem with this approach, however, is that the approaches are often proprietary and you get into a vendor lock-in, and the approaches are often difficult to maintain in a broader environment [9].

One way to get around this vendor lock-in is to use standards. In the last years, web components have emerged as an alternative for building reusable, framework-independent UI elements [10]. The World Wide Web Consortium (W3C) defines these components across web browsers [11]. These components enable developers to create custom HTML elements encapsulating functionality, style, and behaviour [10], [11]. Thus, they can be modular and easily integrated [11]. Compared to the development of database applications based on traditional programming languages, the development or integration of database applications using HTML code is less complex and therefore allows even business experts with basic HTML knowledge to implement or adapt an application according to their requirements and can therefore significantly reduce the workload of a company's IT-department.

As a consequence, we introduce a set of loosely coupled web components designed for database interaction functionality to web applications. Our approach is similar to adaptive Linked Data-driven Web components [12]. These components provide the following database operations:

- Table display and navigation (browsing relational database content)
- Dataset editing (modifying individual records)
- Query execution (dynamically running predefined SQL statements)
- Selection filtering (interactive searching and data retrieval)

These components communicate with databases via a REST-based access layer. Unlike traditional frameworks, this approach provides a lightweight, low-code alternative that enables non-experts to build database-driven web applications.

Unlike other low-code applications, which are often self-contained applications, we rely entirely on the W3C composite standard *Web Components* [13] and thus enable our components not only to create new applications, but also to be easily integrated into existing applications.

We structured the paper as follows: First, in Section II, we give an overview over related work. After that, we provide an overview of the composite standard *Web Components* in Section III. Then, we outline the system architecture in Section IV. Then, Section V presents the implemented components. The coupling of components with each other and with the other elements of the website is described in Chapter VI. Section VII demonstrates a real-world example application, while Section VIII discusses key challenges and limitations. Finally, Section IX concludes with future directions and planned enhancements.

II. Related Work

A. Database Access Through the Web

The classic approach to bringing database content to the web is via server-side programming such as PHP, Python or node.js (server-side JavaScript). To avoid having to constantly reinvent the wheel of web-based programming, frameworks based on these languages such as django (Python), symfony (PHP) or express (JavaScript) have emerged, most of which implement some form of the Model View Controller (MVC) paradigm and use an object relational mapping framework to access the database. In the field of database administration, phpMyAdmin [14] and derivatives such as pgMyAdmin are popular tools. Our approach differs fundamentally from this classic approach. Neither is the rendering done on the server side, nor is an imperative programming model used to create the content.

B. Java Applets

Java applets [15] took a different approach. These were typically small programs written in Java that were loaded from the server and ran in the user's browser in a protected environment (sandbox). A sophisticated access API for relational databases was also available through JDBC [16]. The integration of the applets into the web pages is done with the element `embed` or `object` and has a number of similarities with the web components approach we use. Both approaches run on the client within the browser and there are a number of predefined methods that must be implemented to integrate the application into the page. The integration is declarative in both cases and parameters can be passed to the program. Both approaches also allow the programmatic access to the DOM tree of the embedding website. From the mid-2010s, however, support for the applets was gradually discontinued by the browser manufacturers.

C. Declarative Web

At the ACM Web Conference 2023, Steven Pemberton delivered a presentation titled "The One Hundred Year Web," highlighting the escalating complexity and consequent formidable challenges in implementation [17]. Pemberton critiques the departure from the declarative path of the web with HTML5, referring to it as a "Cowpath" that is incongruent with the original principles [17]. He expresses hope that the damage incurred thus far can be rectified through the collaborative efforts of the web community, aiming to ensure the long-term backward compatibility of the web [17]. In our work, we want to follow exactly this approach and show that the otherwise imperative integration of database content can also be done declaratively.

Michael Hanus has introduced a concept that combines declarative programming with the use of a CGI program, which generates a web application based on the database [18]. Therefore, Hanus proposed an interface that integrates both functional and logical aspects derived from his Curry approach [18]. He posits that this conceptual framework is transferrable to a client-side context through the generation of JavaScript [18]. The primary distinction to our work lies in the rendering location of the HTML document. In the presented method, the HTML document is rendered on the server-side and transmitted to the client. On the other hand, in the client-side approach, the data is loaded from the server and rendered directly on the client-side.

D. Low Code Development

Low-code development platforms have the potential to significantly change the tasks of software engineering and help developers in companies create applications themselves without having to delve too deeply into coding, which significantly expands the pool of potential developers. Typically, low-code development platforms (LCDPs) provide a visual interface for application development that is realized through model-driven design and declarative programming [19], [20]. In addition to platform-specific platforms, there are also approaches that generate web applications as the target platform, such as the low-code platform Xelence from Sagitec Software [21].

The low-code initiative has given rise to a number of application developers, including Caspio [22], Budibase [23], webflow [24], and Bubble.io [25], which enable web-based applications to be developed in the form of single-page applications with little or no programming effort. However, these are standalone systems that are difficult or impossible to integrate into existing web applications. In contrast to these approaches, our low-code web components offer this functionality with ease of development, interoperability, extensibility, and maintainability.

III. Web Components

Web Components are custom, encapsulated, and reusable elements designed for integration into web pages or applications. They are processed and executed within contemporary web browsers. As of today, these APIs are integral components of the Web Hypertext Application Technology Working Group (WHATWG) standard [26].

A *custom element* is a JavaScript class which can define custom HTML elements [26]. This element has to be inherited from the `HTMLElement` class [26].

The `HTMLElement` serves as the foundational class for every element present on a web page [26]. While it is possible to derive from a specific class like `HTMLParagraphElement`, such an approach is not fully supported by all browsers.

A Custom Element follows a specific lifecycle when invoked. Initially, the `constructor` is executed, establishing the initial configurations. Upon inclusion in the DOM, the `connectedCallback` method is triggered. Subsequently, the component is prepared and capable of both receiving and emitting events. If a property is changed, the `attributeChangedCallback` method is called, but only if the attribute is defined in the static property `observedAttributes`. Finally, when the element is removed from the DOM, the `disconnectedCallback` method is employed to internally reset the component [26].

Custom elements are registered by calling the method `define(tagName,class)` of class `customElements`. The method expects the tag name as the first parameter, so that it can be used in the markup, e.g. with `db-table`. The name of the implemented class is specified as the second parameter [26].

IV. Architecture of the Database web components

While the web components run in the browser, they have to communicate with a database server. The developed web components don't communicate directly with the database, but through a thin Representational State Transfer (REST)-based access layer (see Fig. 1, middle). This service maps a logical database identifier to a specific database on the server side. We use the PHP-CRUD-API library [27] by Maurits van der Schee as the core for this purpose and have extended it with additional functionality. PHP-CRUD-API provides a REST-based CRUD interface for accessing relational databases, i.e., records can be created, read, updated, and deleted. We implemented the necessary extension modules ourselves, such as access to the database metadata and a module for executing SQL statements, and integrated them into the PHP-CRUD-API as *customControllers* [27]. In order to be able to handle multiple databases per endpoint, we have also implemented the wrapper module `rdbms.php`, which provides additional meta information about the databases available at the endpoint and initializes the PHP-CRUD-API module with the specific, selected database.

Specifically, the service provides the following functionality:

Database scheduler:
: The REST API can manage multiple databases running on any computer. For this purpose, the new entry point *rdbms.php* has been implemented, which takes over the management of the databases. This also includes providing meta information about which databases are available via this endpoint. Once a specific database has been specified, the *rdbms.php* module forwards the request to the PHP-CRUD-API module for processing.

Access Control:
: In the configuration for accessing the databases, it can be specified which tables are accessible. In addition to this coarse-grained access control, the PHP-CRUD-API library offers further mechanisms such as authentication via API key, JWT token, or username/password. Currently, we are implementing authentication based on Keycloak [28] via JWT token forwarding.

CRUD-Functionality:
: This functionality is completely coverd by the PHP-CRUD-API library and includes the (C)reation, (R)eading, (U)pdating, and (D)eletion of datasets in the database.

Metadata Module:
: For the purpose of constructing forms for creating and modifying data records, as well as for resolving foreign key relationships, we require information about the structure of the tables and their constraints. We have implemented this functionality as a custom controller of the PHP-CRUD-API module. This has the advantage that the authentication mechanisms used by PHP-CRUD-API can also be used for this module.

SQL Module:
: The SQL module was also implemented as a custom controller of the PHP-CRUD-API library for the same reasons as the metadata module. It allows the execution of parameterized SQL select statements.

V. Components

As part of our research work, we have developed a series of web components for the declarative integration of database functionality into HTML pages. Concrete, the following components were realized:

db-connection: This component establishes the connection between the components and the RESTful backend service. It acts as an intermediary between the other components and the RESTful service. Additionally, it is also responsible for authentication based on Keycloak via JWT token forwarding.

db-table: This component represents a database table. The functionality ranges from the simple display of data records to a wide variety of interaction options like sorting, further filtering an so on.

db-row: Component for representing a single data set (row in a table). The functionality of this component ranges from simple, non interactive visualization of the data set to the creation and editing of data sets using predefined or freely definable forms, and the use as a controller in a model-view-controller (MVC) szenario.

db-field: This component represents a single attribute of a data set. `db-field` components are used in conjunction

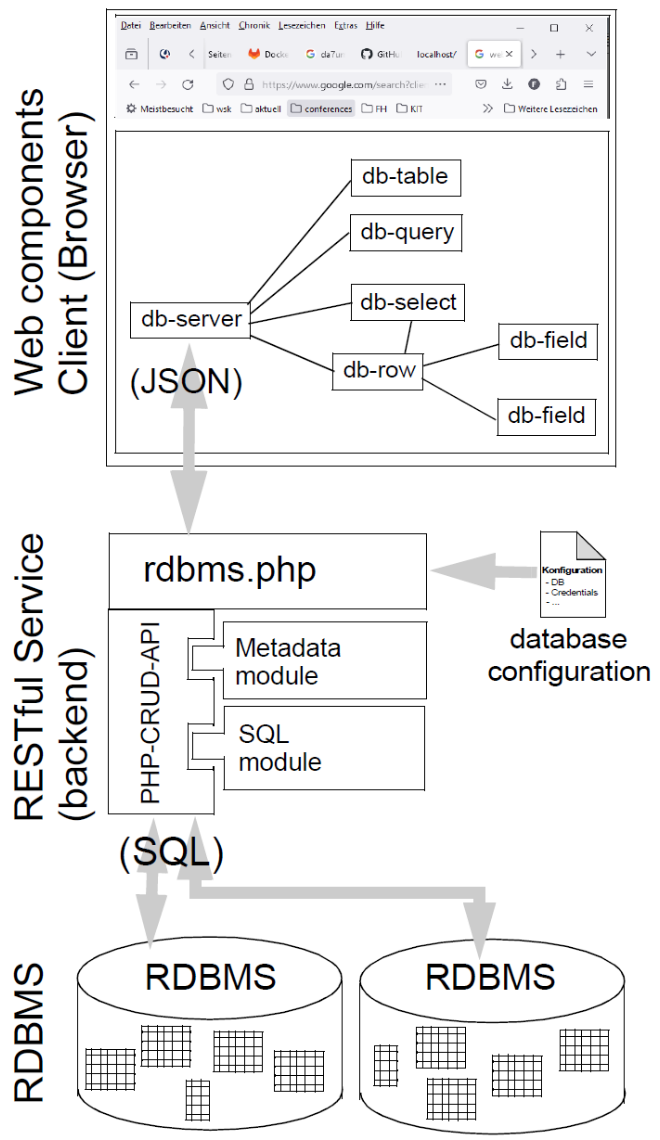


Figure 1. Architecture of our database web components.

with the `db-row` component when arbitrary layouts are to be realized for the visualization of a data set.

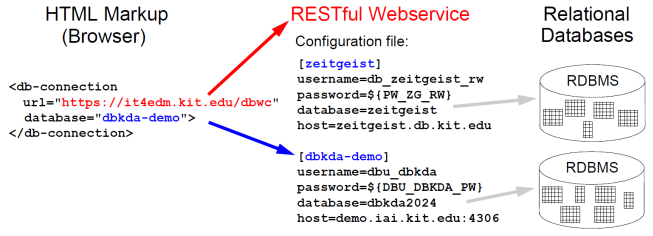
db-select: Analogous to the HTML select element, in which a value can be selected from a list of predefined values. In the case of the `db-select` component, the displayed values come from the results of an SQL query or the selection of certain columns from a table.

db-query: Component that displays the results of an arbitrary SQL query. It is also possible to browse through the results and resort them.

In the following we will present these components in detail. The data in the example screenshots shown comes from the *Mondial* database [29]. The PHP-CRUD-API library we use does not support composite keys for write operations. For this reason, we modified the *Mondial* database schema and added artificial keys and corresponding foreign keys.

A. Connection-component

The connection-component is a non-visible component in a page. It is responsible for the mapping to a concrete database on server-side. The left side of Fig. 2 gives an example, how the web component is integrated inside a HTML page. The `db-connection` component communicates with a RESTful service, which is specified by the parameter `url`. The further parameter `database` specifies a logical database name, which is mapped on server side (Fig. 2, middle) to a specific database (right side of Fig. 2). Note that the database can run on any computer and not necessarily on the computer with the REST-API endpoint. Table I shows all possible attributes of the component.



```
<db-connection
  url="https://it4edm.kit.edu/dbwc"
  database="dbkda-demo">
</db-connection>
```

```
[zeitgeist]
username=db_zeitgeist_rw
password=${PW_ZG_RW}
database=zeitgeist
host=zeitgeist.db.kit.edu

[dbkda-demo]
username=dbu_dbkda
password=${DBU_DBKDA_PW}
database=dbkda2024
host=demo.iai.kit.edu:4306
```

Figure 2. Database mapping (from [30]).

TABLE I
`db-connection` ATTRIBUTES

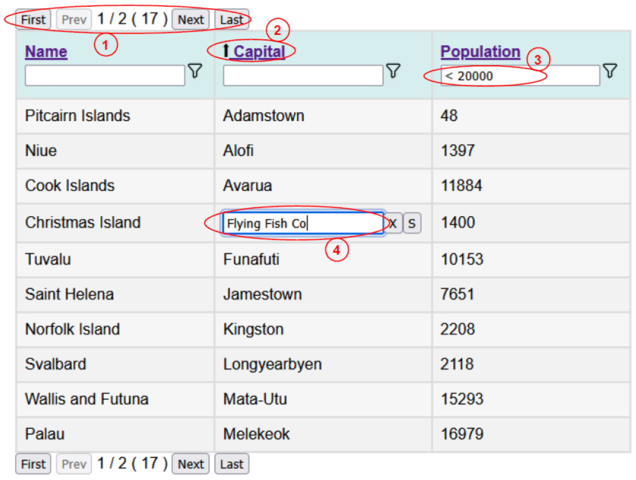
Attribute	Description	Mandatory
database	Logical database name. This name is mapped to a concrete database on the backend side.	yes
url	URL of the REST-API endpoint Default: URL, from where the web-components are loaded	no

B. Table-component

The `db-table` component is responsible for displaying the content of a database table or a part of it. Fig. 3 shows in the upper part the definition of a `db-table` component. The table to be shown is “country”, of which the three columns `Name`, `Capital`, and `Population` are to be displayed (attribute `attribute-list`). The `actions` attribute specifies the possible interaction options. In this specific case, page-by-page scrolling (`paging`) is enabled with a page size of 10 datasets (parameter `pagesize`), the datasets can be sorted in ascending and descending order according to the column values (`sort`), and additional filtering can be carried out at column level (`filter`). Inline-edit is also activated (`inline-edit`).

In the lower part of Fig. 3 the visual appearance of the component inside the browser, according to the previously described specification is shown. Paging (1), sorting (2), filtering (3) and inline-edit (4) as specified in the markup are activated. The full list of possible attributes of the component are shown in Table II. The `refresh-rate` attribute, for example, is responsible for ensuring that the current database content is always displayed by accessing the database table again every n seconds and rereading and displaying the actual values. In addition to the parameters from Table II, the appearance of the component can also be adapted to your own requirements using cascading stylesheets (CSS).

```
<db-table table="country"
          actions="paging,sort,filter,inline"
          pagesize="10"
          attribute-list="Name,Capital,Population">
</db-table>
```



Name	Capital	Population
Pitcairn Islands	Adamstown	48
Niue	Alofi	1397
Cook Islands	Avarua	11884
Christmas Island	Flying Fish Co	1400
Tuvalu	Funafuti	10153
Saint Helena	Jamestown	7651
Norfolk Island	Kingston	2208
Svalbard	Longyearbyen	2118
Wallis and Futuna	Mata-Utu	15293
Palau	Melekeok	16979

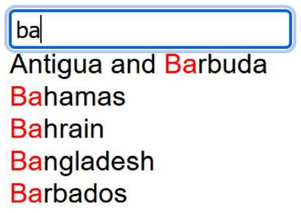
Figure 3. Specification and visual appearance of the web component `db-table`, showing different interaction elements like page-wise scrolling (1), sorting (2), filtering (3), and inline-editing (4).

TABLE II
`db-table` ATTRIBUTES

Attribute	Description	Mandatory
table	Name of the table	yes
filter	Mandatory filter condition, that all datasets must fulfill	no
pagesize	Maximum number of datasets on a page	no
page	Page to display	no
order	Sort order (column name)	no
direction	Sort direction (`asc`, `desc`)	no
connection	Id of a `db-connection` web-component. If the attribute is not set, the default server component is chosen	no
attribute-list	Comma separated list of attributes to display (default: all)	no
actions	List of possible values: sort, filter, paging, inline, edit, delete	no
refresh-rate	Time in seconds after which the table data is reloaded from the database	no

C. Selection-component

The `db-select` web component presents a selection box, from which values can be selected and searched via a prefix or infix search. The values are specified by an SQL-select statement. The SQL-statement can either be specified directly by the `sql`-attribute, or it is specified using the attributes `value`, `text`, `table` and (optional) `filter`. On the left side of Fig. 4, the visual appearance with prefix-search is shown, while on the right hand side, the markup, defining the web-component on the left, is shown. The attribute `value` represents the table-column, which values are passed to the HTML-form on submit, while the values of the column, specified by the attribute `text` are displayed by the element and are used for the prefix/infix-search. The attribute `name` specifies the name of the `db-select` element, and thus the name of the attribute under which the selected value is transferred to the form. Table III provides an overview of all possible attributes and their meaning.



```
<db-select
    table="country"
    value="Code"
    text="Name"
    name="Country_Fk">
</db-select>
```

Figure 4. web component `db-select`

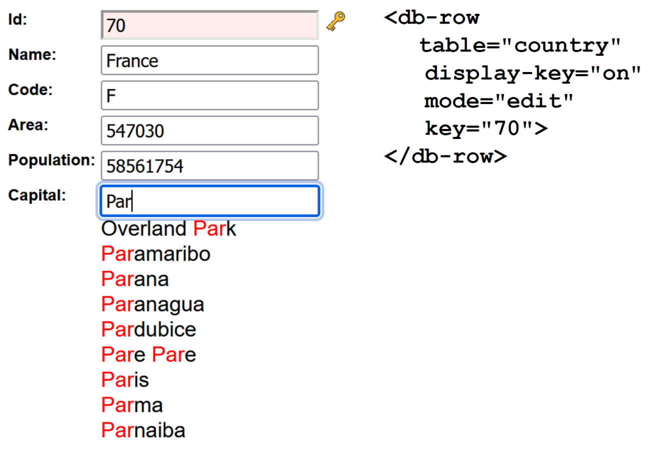
TABLE III
`db-select` ATTRIBUTES

Attribute	Description	Mandatory
connection-id	Id of a db-connection web-component	no
prefix-search	The term entered in the search field is considered as a prefix (Default: true)	no
name	Name of the element. The value of the selected entry (attribute `value` in Variant I, first column of SQL-statement in Variant II) is passed on to the form under this name.	no
Variant I		
table	Name of the table	yes
value	Name of the attribute used for the value-attribute of the option element (returned by the form).	yes
text	Name of the attribute used for the text-node of the option element (value used for search).	yes
filter	Mandatory condition the datasets must fulfill.	no
Variant II		
sql	SQL-Statement with one or two columns. The statement can have one or two columns in the select-clause. If only one column is given, the value of this column is used for the value and the text.	yes

D. Row-component

The component represents a single dataset. It allows you to create a new record or edit an existing one. The component provides its own HTML form for this purpose. To do this, the component uses the functionality provided by the metadata module to determine the structure and constraints of the table. This concerns information about the names and data types of the fields, `not null` constraint and information about primary and foreign key relationships. With the help of this information, foreign key relationships, for example, are resolved by displaying not the foreign key value but the referenced data record. This feature is shown on the left side of Fig. 5, where the foreign key attributes `Capital_fk` (label: Capital), which reference the capital city of a country, is not displayed, but a selection box showing the current value and simply giving the option to change it using the `db-select` web-component described previously. If a renaming of the attribute identifiers of the database table is desired (as here, `Capital` instead of `Capital_fk`), this can be accomplished with CSS rules.

The right side of Fig. 5 shows, how the markup of the web-component, shown on the left, looks like. The parameter `key` expects the value of the primary key of the dataset. The name of the primary key attribute does not have to be specified as it is determined from the metadata of the table. If the attribute `key` is omitted, the component provides a form within which a new data record can be created.



Id:	70
Name:	France
Code:	F
Area:	547030
Population:	58561754
Capital:	Par

Overland Park
Paramaribo
Parana
Paranagua
Pardubice
Pare Pare
Paris
Parma
Parnaiba

```
<db-row
   table="country"
    display-key="on"
    mode="edit"
    key="70">
</db-row>
```

Figure 5. Web-component `db-row` in edit-mode using the predefined layout. Foreign keys (i.e., Capital) are represented by `db-select`-components, allowing to search and select a concrete dataset.

Another use of the `db-row` component is as a controller in a model-view-controller (MVC) setup (attribute `controller="true"`). In this case, the component is invisible and reads the parameters either from the GET parameters of the current URL or from a JSON string that has been passed to the `data`-attribute. Depending on whether the primary key value has been specified, an SQL update or insert operation is performed. Subsequently, the page specified by the `target-url` is loaded (or `error-url` in case of an error).

Table IV lists all possible attributes of the web component `db-row`. The `form` attribute, for example, is used to create your own HTML form layout and link it to the component, so that this form is used instead of the internal one. If the parameter `is-editable` is set to `false`, a predefined, read-only representation of the dataset is displayed. If this does not meet layout requirements, the `visible` attribute can be set to `false` and the layout can be defined using `db-field` components (see below).

E. Field-Component

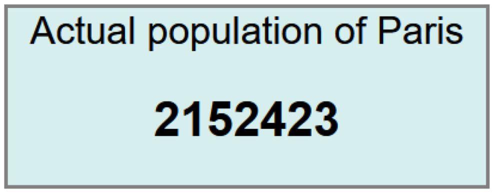
The `db-field` component handles exactly the value of one column of a dataset. It gets its value from a `db-row` component and returns the value within a `<span>` element. If only one `db-row` is specified on the HTML-page, this is used automatically, otherwise the desired `db-row` must be specified with the parameter `dataset`. Fig. 6 gives a minimal example, displaying the name and population of the city with the primary key value of 2037 (Paris). In the upper part, you see the result in the browser, at the bottom the corresponding declaration of the elements `db-row` and two `db-field` elements in the HTML page. Note that in the `db-row` component, the attribute `refresh-rate` is set to the value of 10. This ensures that the dataset is reloaded every 10 seconds so that the actual number of inhabitants is always displayed.

TABLE IV
`db-row` ATTRIBUTES

Attribute	Description	Mandatory
table	Name of the table	yes
connection-id	Id of a db-connection web-component	no
visible	Specifies if the component should display the dataset on the page. If the component is used together with `db-field` components you typically set this parameter "off" (default: on)	no
is-editable	Specifies if the dataset should be editable. Possible values: yes/no(default: yes)	no
display-key	Flag, if the database key should be displayed or not	no
attribute-list	Comma separated list of attributes to be shown (default: all attributes)	no
key	The value of the primary key of a dataset in this table.	no
form	the id of a form-element, the `db-row` component should work with. Additionally a mapping between the form fields and the table columns can be specified using the *mapping* attribute (see below).	no
mapping	Mapping between form field name and table column name.	no
controller	Enable "controller-mode". If the parameter is set to "true", the parameter `action` must also be set (`store-from-get-request`, `store-from-data`).	no
action	See description above (controller-mode).	no
target-url, error-url	Page to load after dataset is written (controller-mode).	no
refresh-rate	Duration (in seconds) until the dataset is read again	no

F. Query-Component

The visual representation of the `db-query` component is similar to that of the `db-table` component in Fig. 3. The main difference is that no `table` parameter is specified, but an arbitrary SQL select-statement. Fig. 7 shows an example in which the SQL statement determines the number of borders and overall border length for all countries having more than 5 neighbors, sorted by decreasing number of neighbors and decreasing border length. Table V provides a list of all possible attributes.

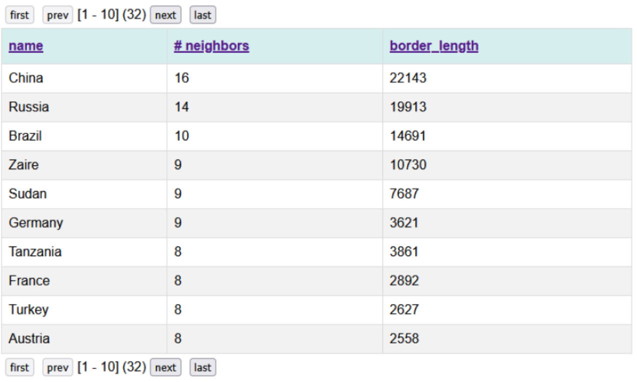


Actual population of Paris

2152423

```
<db-row table="city" visible="off" key="2037"
        refresh-rate="10">
</db-row>
<div class="large-box">
    Actual population of <db-field attribute="Name">
                         </db-field>
    <p/>
    <db-field class="bold-huge" attribute="Population">
    </db-field>
</div>
```

Figure 6. Two `db-field` components with an invisible `db-row` component



first prev [1 - 10] (32) next last

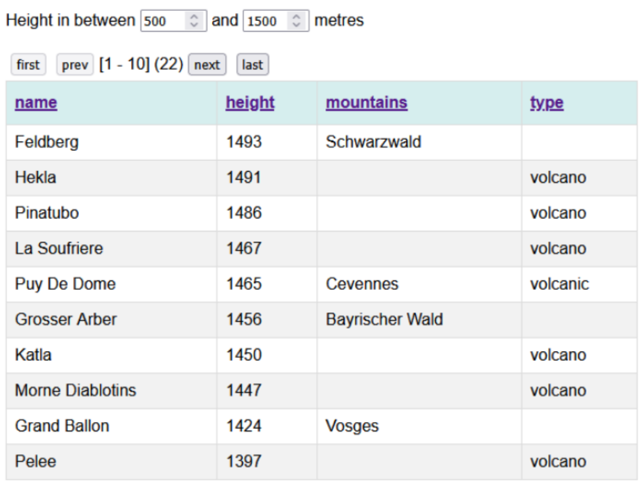
name	# neighbors	border_length
China	16	22143
Russia	14	19913
Brazil	10	14691
Zaire	9	10730
Sudan	9	7687
Germany	9	3621
Tanzania	8	3861
France	8	2892
Turkey	8	2627
Austria	8	2558

first prev [1 - 10] (32) next last

```
<db-query
   sql="select c.name, count(*) as &quot;# neighbors&quot;,
               round(sum(length)) as border_length
          from country c
          left join borders b
            on b.Country1_fk=c.Id or b.Country2_fk=c.Id
         group by c.Id, c.Name
        having count(*) > 5
         order by 2 desc, 3 desc"
   pagesize="10"
   actions="paging,sort">
</db-query>
```

Figure 7. Web-component `db-query` executing a complex SQL statement. The result datasets can be scrolled and ordered by the different columns of the result.

One of the most important enhancements since the version of our components described in [1] is the support of parameterizable SQL-statements. Instead of values, it is possible to define placeholders in the SQL-statement in the form of question marks (?), which get their value from other HTML elements of the website. This can be, for example, the value of an input field, or the currently selected value of an HTML-selectbox or the *db-select* (see Section V-C) component implemented by us. Fig. 8 shows this functionality, where the SQL-statement contains two parameters that take their values from the two input fields that specify the minimum and maximum height of the mountains to be displayed. The mapping between the input fields and the parameters in the SQL statement is realized by the attribute `params` of the component `db-query`. The values of the attribute specify the id of the HTML components and the property to be read. In the case of the HTML input element, this is the property `value`.



Height in between 500 and 1500 metres

first prev [1 - 10] (22) next last

name	height	mountains	type
Feldberg	1493	Schwarzwald	
Hekla	1491		volcano
Pinatubo	1486		volcano
La Soufriere	1467		volcano
Puy De Dome	1465	Cevennes	volcanic
Grosser Arber	1456	Bayrischer Wald	
Katla	1450		volcano
Morne Diablotins	1447		volcano
Grand Ballon	1424	Vosges	
Pelee	1397		volcano

```
<form>
  Height in between
  <input type="number" id="low-limit"  value="500"> and
  <input type="number" id="high-limit" value="1500">
 metres
</form>
<p>
<db-query sql="select name, height, mountains, type
                 from mountain
                where height >= ?
                  and height <= ?
                order by height desc"
          params="low-limit.value high-limit.value"
          pagesize="10">
</db-query>
```

Figure 8. Web-component `db-query` with a parameterized SQL-statement, reading the values from two input fields.

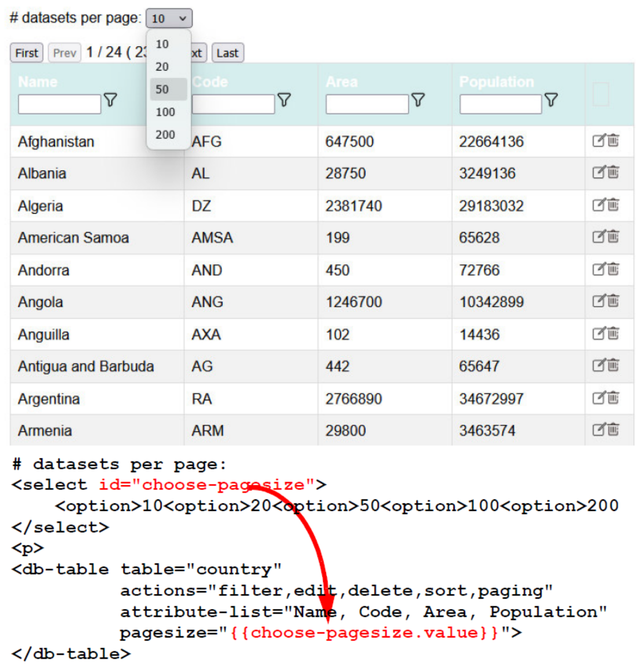
By specifying the two input elements from which the values for the parameterized SQL-statement are read, event handlers are also registered, which inform the `db-query` component about changes to the values for the input fields, so that the component executes the SQL-statement again when the values change.

TABLE V
`db-query` ATTRIBUTES

Attribute	Description	Mandatory
sql	SQL Statement to be executed	yes
pagesize	Maximum number of datasets on a page	no
page	Page to display	no
connection	Id of a `db-connection` web-component. If the attribute is not set, the default server component is chosen	no
refresh-rate	Duration (in seconds) until the query is resubmitted	no
actions	List of possible values: sort, paging	no

VI. Parameter Binding

Analogous to the behavior of parameterized SQL statements, we have implemented a mechanism that allows the values of the parameters of our components to be read from other components. This is achieved by a special syntax, in which instead of the value for an attribute, the id of the HTML component followed by the concrete property (dot-notation) is specified within double curly brackets. Fig. 9 demonstrates this using the example of the `pagesize` attribute of the `db-table` component. Instead of specifying the value as fixed, the `{{..}}` notation is used here to reference the property `value` of the select element with the ID `choose-pagesize` (`{{choose-pagesize.value}}`).



Name	Code	Area	Population	
Afghanistan	AFG	647500	22664136	
Albania	AL	28750	3249136	
Algeria	DZ	2381740	29183032	
American Samoa	AMSA	199	65628	
Andorra	AND	450	72766	
Angola	ANG	1246700	10342899	
Anguilla	AXA	102	14436	
Antigua and Barbuda	AG	442	65647	
Argentina	RA	2766890	34672997	
Armenia	ARM	29800	3463574	

```
# datasets per page:
<select id="choose-pagesize">
    <option>10<option>20<option>50<option>100<option>200
</select>
<p>
<db-table table="country"
          actions="filter,edit,delete,sort,paging"
          attribute-list="Name, Code, Area, Population"
          pagesize="{{choose-pagesize.value}}">
</db-table>
```

Figure 9. Parameter binding: Mapping attribute values to other HTML-elements.

VII. Example Application

In this section, the use of the components we have developed will be demonstrated using a small example application. The components `db-connection`, `db-table` and `db-row` will be used. The aim of the application is a simple task list in which tasks to be completed can be entered and managed with a priority and optional deadline. The application consists of two HTML pages, a view and a controller page. The view is shown in Fig. 10 and contains a HTML-form for entering a new task and below it a table with the tasks already created. Beside the form-element with the input fields for the new task, the page contains the web-components `db-connection` and `db-table`.

When the submit button ("add task") is pressed, the form data is send to the page `iaria-demo.controller.html` (see Fig. 11) via the GET-method and this page acts as a controller like in a MVC setup, storing the passed values and after that reload the calling page. The complete code of the controller file `iaria-demo.controller.html` is shown in Fig. 11. In line 2 and 3, our database web-components are loaded. Line 4 and 5 sets up the component `db-connection` to establish a connection to the database with the name `iaria-demo`. If the attribute `url` is omitted, it is expected that the endpoint resides at the same address, from where the components were loaded (see line 2). In lines 6 to 8, the component `db-row` is configured as a controller (`controller="true"`), reads the parameters passed from the previous page (the view with the form from Fig. 10) and inserts the dataset into the `todo_list` table. Since no `target-url` or `error-url` parameter are specified, the previous form page is reloaded after the insert operation is executed.

VIII. Discussion, Further Challenges

Our web components for database developers serve as a proof of concept, demonstrating that lightweight, framework-independent database interaction is feasible. However, we must work towards a low-code or no-code solution to transition from a developer-focused prototype to a solution for no-coders. A visual tool allowing users to connect components dynamically could improve accessibility - such as a split-screen UI with commands on the left and live interaction on the right.

Challenges such as validation, scalability, and security are critical for real-world usage. Performance testing is needed to ensure components handle large datasets and concurrent users efficiently, while usability testing will determine if non-developers can use the system effectively. Hosting components on a CDN and optimizing the backend REST API will enhance scalability. Security remains a primary concern, requiring role-based access control (RBAC) to restrict data access. Additionally, input sanitization is necessary to prevent SQL injection and XSS attacks.

To address these challenges, we will:

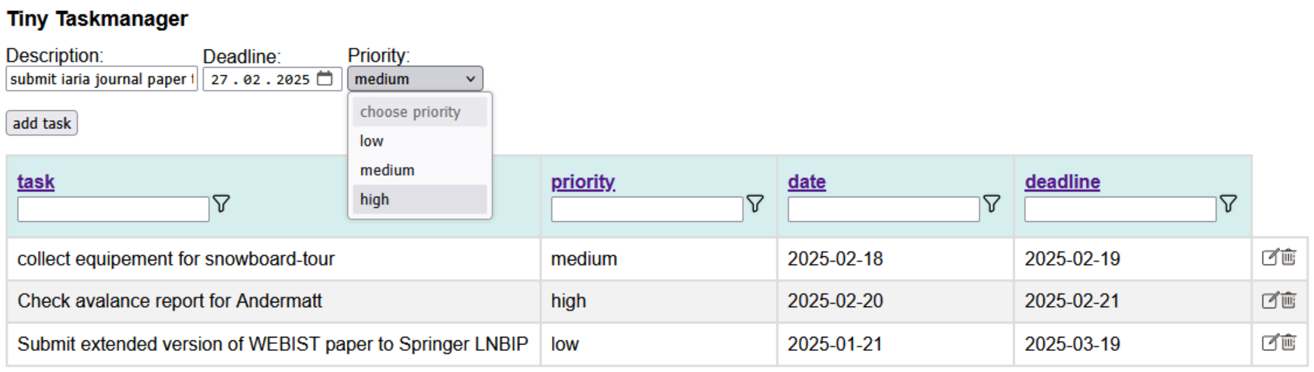
1) Develop a wizard-based UI for configuring components without coding.
2) Implement authentication and authorization mechanisms for secure user access on the World Wide Web.
3) Integrate components into third-party web applications for real-world test settings to evaluate arising problems.

Our database components could enhance scalability, security, and usability and serve as an alternative to traditional methods, effectively connecting developer-driven tools and no-code solutions.

IX. Conclusion and Outlook

We have implemented the first prototype of our web components and are actively working on expanding their functionality. One key improvement is evaluating all available database metadata directly within the components. This metadata is already visible in Fig. 5, where a selection box displays a dereferenced value (e.g., "Paris") instead of the foreign key for the capital.

Security remains a significant area for future work. Actually we allow the restriction to only acces certain tables inside a database and support authentication based on keycloak via JWT token forwarding. For future iterations we plan to implement a role-based access control (RBAC) meachanism. Using our web components in another web application with RBAC makes


Tiny Taskmanager

Description: submit iaria journal paper
Deadline: 27 . 02 . 2025
Priority: medium

task	priority	date	deadline	
collect equipement for snowboard-tour	medium	2025-02-18	2025-02-19	
Check avalance report for Andermatt	high	2025-02-20	2025-02-21	
Submit extended version of WEBIST paper to Springer LNBIP	low	2025-01-21	2025-03-19	

Figure 10. Example Application: View with form and db-table component.

```
<!-- controller file: iaria-demo.controller.html -->
<script type="module" src="http://localhost/dbwc/db.js">
</script>
<db-connection database="iaria-demo">
</db-connection>
<db-row controller="true" table="todo_list"
          action="store-from-get-request">
</db-row>
```

Figure 11. db-row component, acting as a controller (complete code).

the integration possibilities interesting and the security aspects challenging.

Several functional enhancements are also in progress. The db-select component will be extended to support multiple selections, making handling $n : m$ relationships easier. The db-table component currently does not resolve foreign keys, but this can be worked around using db-query with SQL-join operations. In the future, native support for foreign key dereferencing in the db-table will be added as an optional feature.

For future work, we focus on four key areas:

1) **Security and Authentication** – Adding built-in authentication and authorization mechanisms.
2) **Scalability and Performance** – Optimizing data handling for large datasets and concurrent users.
3) **No-Code Accessibility** – Developing a visual configuration wizard for non-developers.
4) **Evaluation** - Evaluation of our framework in terms of the time required to develop a specific application. This should include a comparison with traditional software development approaches as well as with modern existing low-code solutions.

Future work will also involve real-world testing and integration into enterprise applications to gather feedback and further refine the components. Our components will enable business developers to create new business cases with software support without needing software engineers.

References

[1] A. Schmidt and T. Münch, “Web components for database developers”, in *Proceedings of the Sixteenth International Conference on Advances in Databases, Knowledge, and Data Applications*, 2024, pp. 20–22.

[2] H. Chen, R. H. Chiang, and V. C. Storey, “Business intelligence and analytics: From big data to big impact”, *MIS quarterly*, pp. 1165–1188, 2012.

[3] R. Vyas, “Comparative analysis on front-end frameworks for web applications”, *International Journal for Research in Applied Science and Engineering Technology*, vol. 10, no. 7, pp. 298–307, 2022.

[4] J. Hassan, “The effects of architectural design decisions on framework adoption: A comparative evaluation of meta-frameworks in modern web development”, Ph.D. dissertation, May 2024. DOI: 10.13140/RG.2.2.10552.97287.

[5] C. Shapiro, *Information rules: A strategic guide to the network economy*. Harvard Business School Press, 1999.

[6] J. Eckstein and B. R. Schultz, *Introductory relational database design for business, with Microsoft Access*. John Wiley & Sons, 2018.

[7] K. Almaree *et al.*, “The usefulness of cash budgets in micro, very small and small retail enterprises operating in the cape metropolis”, *Expert Journal of Business and Management*, vol. 3, no. 1, 2015.

[8] M. Skare, M. d. l. M. de Obesso, and S. Ribeiro-Navarrete, “Digital transformation and european small and medium enterprises (smes): A comparative study using digital economy and society index data”, *International journal of information management*, vol. 68, p. 102 594, 2023.

[9] K. Rokis and M. Kirikova, “Challenges of low-code/no-code software development: A literature review”, in *International conference on business informatics research*, Springer, 2022, pp. 3–17.

[10] T. Münch, “Vanilla js-design and implementation of a progressive web application from scratch”, in *International Conference on Web Engineering*, Springer, 2024, pp. 461–464.

[11] D. Glazkov and H. Ito, *Introduction to web components*, https://www.w3.org/TR/components-intro/, [Online; accessed 2025-05-27].

[12] A. Khalili, A. Loizou, and F. van Harmelen, “Adaptive linked data-driven web components: Building flexible and reusable semantic web interfaces: Building flexible and reusable semantic web interfaces”, in *The Semantic Web. Latest Advances and New Domains: 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29–June 2, 2016, Proceedings 13*, Springer, 2016, pp. 677–692.

[13] “Web components - specifications”, [Online; accessed 2025-05-27], 2015, [Online]. Available: https://www.webcomponents.org/specs.

[14] M. Delisle, *Mastering phpMyAdmin 3.1 for Effective MySQL Management*. Packt Publishing, 2009, ISBN: 1847197868.

[15] E. Boese, *An Introduction to Programming With Java Applets*. Jones and Bartlett Publishers, 2009.

[16] G. Reese, *Java Database Best Practices: Persistence Models and Techniques for Java Database Programming*. O'Reilly, 2009.

[17] S. Pemberton, "The one hundred year web", in *Companion Proceedings of the ACM Web Conference 2023*, ser. WWW '23 Companion, Austin, TX, USA: Association for Computing Machinery, 2023, pp. 642–647, ISBN: 9781450394192. DOI: 10.1145/3543873.3585578.

[18] M. Hanus, "Lightweight declarative server-side web programming", in *Practical Aspects of Declarative Languages: 23rd International Symposium, PADL 2021, Copenhagen, Denmark, January 18-19, 2021, Proceedings*, Copenhagen, Denmark: Springer-Verlag, 2021, pp. 107–123, ISBN: 978-3-030-67437-3. DOI: 10.1007/978-3-030-67438-0_7.

[19] E. Elshan, E. Dickhaut, and P. Ebel, "An investigation of why low code platforms provide answers and new challenges", in *Hawaii International Conference on System Sciences (HICSS)*, (Maui, Hawaii), Maui, Hawaii, 2023.

[20] N. Prinz, C. Rentrop, and M. Huber, "Low-code development platforms-a literature review.", in *AMCIS*, 2021.

[21] R. Arora, N. Ghosh, and T. Mondal, "Sagitec software studio (s3)-a low code application development platform", in *2020 International Conference on Industry 4.0 Technology (I4Tech)*, IEEE, 2020, pp. 13–17.

[22] Caspio, *Caspio: Low-Code Platform - Build Online Database Apps*, https://www.caspio.com/, (Accessed on 2025-05-23), 2024.

[23] Budibase, *Github: Budibase/budibase*, https://github.com/Budibase/budibase, (Accessed on 2025-05-23), 2024.

[24] Webflow, *Webflow: Create a custom website | Visual website builder*, https://webflow.com/, (Accessed on 2025-05-23), 2024.

[25] Bubble, *Bubble: The full-stack no-code app builder*, https://bubble.io/, (Accessed on 2025-05-23), 2024.

[26] *WHATWG. HTML Living Standard*, https://html.spec.whatwg.org/multipage/, [Online; accessed 2025-05-27].

[27] M. van der Schee, *PHP-CRUD-API*, https://github.com/mevdschee/php-crud-api, Last accessed 17.2.2025, 2025.

[28] S. Thorgersen and P. I. Silva, *Keycloak-identity and access management for modern applications: harness the power of Keycloak, OpenID Connect, and OAuth 2.0 protocols to secure applications*. Packt Publishing Ltd, 2021.

[29] W. May, "Information extraction and integration with FLORID", Last accessed 17.2.2025, 1999, [Online]. Available: http://dbis.informatik.uni-goettingen.de/Mondial.

[30] A. Schmidt and T. Münch, "Enable Business Users to Embed Dynamic Database Content in Existing Web-Based Systems Using Web Components and Generic Web Services", in *Proceedings of the 20th International Conference on Web Information Systems and Technologies - WEBIST*, 2024, pp. 296–306. DOI: 10.5220/0013000000003825.

A Modular Approach for ABM/LMM Models: Specification of Reusable Building Blocks Centred on the Economic Concepts of WTA and WTP

Eric Innocenti *, Dominique Prunetti *, Marielle Delhom †, Corinne Idda *
*UMR CNRS 6240, LISA Research Laboratory, University of Corsica Pasquale Paoli, Corte, France
e-mail:{eric.innocenti | prunetti_d, | idda_c}@univ-corse.fr
†UMR CNRS 6134, SPE Research Laboratory, University of Corsica Pasquale Paoli, Corte, France
e-mail: marielle.delhom@univ-corse.fr

***Abstract*—Agent-Based Models focusing on land markets provide a computational framework to simulate socio-economic dynamics in land and real-estate markets. In this paper, we introduce the 5-Step Simulation Iterative Modelling Process method, an iterative, five-step modelling and simulation decomposition approach specifically designed to structure the development of Agent-Based Land Market Models. We describe how implementing reusable building blocks—conceptual, computational, and executable—enhances modularity and fosters reusability of both theoretical concepts and software code. An illustrative example, applied to land and real-estate markets in Corsica, concretely demonstrates the application of the method and the creation of these reusable components. The integration of the economic concepts of Willingness To Accept and Willingness To Pay into the design of an Agent-Based Land Market Model exemplifies how these building blocks contribute to market dynamics formation. Finally, we highlight the potential of this approach to strengthen computational simulation, support socio-economic analysis, and promote sustainable land management.**

Keywords-Agent-Based Modelling; Land and Real Estate Markets; Reusable Building Blocks; Socio-Economic Dynamics; Sustainable Land Management.

I. Introduction

This article presents an extended version of the international conference paper titled "*Reusable Building Blocks for Agent-Based Simulations: Towards a Method for Composing and Building ABM/LUCC*", which was presented during SIMUL 2024 [1]. Computer simulations of *Agent-Based Models of Land Use and Cover Change* (*ABM/LUCC*) constitute a relatively recent interdisciplinary research domain within computational economics, positioned at the intersection of computer science, software engineering, economics, geography, and social sciences. *ABM/LUCC* models represent an increasingly specialised branch of *Agent-Based Models* (*ABM*), originating from *Multi-Agent Systems* (*MAS*) within the broader field of *Distributed Artificial Intelligence* (*DAI*). In this work, we argue that modelling *complex socio-economic systems* must follow a progressive and iterative approach, characterised by an iterative *composition-decomposition* process, extending from the formulation of a *conceptual model*—defining key study elements—to the development of executable, *modular*, and *adaptive* computer code. The *5-Step Simplified Iterative Modelling Process* (*5-SSIMP*) guides this modelling procedure from initial concept formulation to the execution of computer-based simulation experiments. The process is structured around producing intermediate models (Phase A), categorised as abstract and concrete, with the ultimate objective of delivering a modular and adaptive executable model. This approach involves specifying *Reusable Building Blocks* (*RBB*) [1], [2]. [2] advocates structuring agent-based models around atomic, clearly documented, and context-specific *RBB*s. Building upon their approach, our work expands this concept by specifically defining and integrating economic building blocks centred on *Willingness To Accept* (*WTA*) and *Willingness To Pay* (*WTP*). These model entities integrate theoretical concepts (conceptual model), computational components (computational model), and executable code modules (executable model). Such entities must be designed generically to ensure their reusability across diverse computational simulation projects. Ultimately, this methodology aims to reduce development costs and time, ensure code reliability, and facilitate integration with artificial intelligence systems. The *modelling phase A* of the *5-SSIMP* specifically targets the creation of three types of *RBB*: conceptual-RBB (*conRBB*), computational-RBB (*comRBB*), and executable-RBB (*exeRBB*). In this article, we illustrate how *RBB* are produced according to phase A of the *5-SSIMP* methodology. We detail the construction of *RBB* using the generic economic concepts of *WTA* and *WTP*, commonly employed in *ABM/LUCC* of the *Agent-Based Land Market Model (ABM/LMM)* type. These represent specialised socio-economic *MAS* designed to study and analyse spatial-economic phenomena driven by agent interactions within market frameworks [3]–[5]. The *WTA* and *WTP* concepts are indispensable for characterising agent decision-making processes within *ABM/LMM*.

In the second part, we clarify the theoretical and methodological framework that structures this interdisciplinary research. We present in detail the generic economic concepts of *WTA* and *WTP*, specifying their roles and integration within the conceptual model.

The third part provides a concise overview of the *5-SSIMP* method, briefly revisiting its primary phases and steps. We define the three types of *RBB*s—conceptual (*conRBB*), computational (*comRBB*), and executable (*exeRBB*)—, which must be specified during phase A of the modelling process. We also describe the fundamental characteristics of these blocks, along with their structural roles in progressively developing intermediate abstract and concrete models.

In the fourth part, we practically illustrate this methodological approach through an application to land and real estate markets in Corsica. We demonstrate how the economic concepts of

WTA and *WTP* can be specified and implemented as *RBB*s within an *ABM/LMM*, tailored to the unique socio-economic characteristics of this tourist region. This case study also serves as an illustrative example of modular structuring within the computational model, highlighting methodological advantages associated with the use of objects, design patterns and *RBB*s.

The fifth part engages, in an in-depth discussion of the essential roles played by *WTP* and *WTA* concepts within agent decision-making processes in the *ABM/LMM*. Using preliminary simulation results, we illustrate how this modular approach, based on *RBB*s, effectively explores simulation data, addressing specific sustainable land management challenges in Corsica.

Finally, the sixth part concludes by summarising the major contributions of this work and outlines future research perspectives. We envisage further refinements of the Corsican *ABM/LMM* and its integration into dedicated software Python infrastructure, facilitating broader generalisation and reusability across larger different territorial and thematic contexts.

II. Theoretical foundation

A. Willingness To Pay and Willingness To Accept

First introduced over a century ago for the former concept [6], *WTP* and *WTA* are fundamental in economics for understanding consumer and producer decision-making processes. *WTP* is defined as the maximum amount a consumer is willing to pay to acquire a good or service. Conversely, *WTA* represents the minimum amount an individual is willing to accept to give up a good or forego a service. These concepts are grounded in the *theory of subjective value*, where the value of a good is determined by the perceived utility it provides. *WTP* is often employed in market studies to estimate potential demand and set optimal prices. It can be measured using various methods, such as *contingent valuation* or *conjoint analysis*, though these techniques are sometimes biased by strategic or hypothetical factors [7]. In theory, *WTP* and *WTA* should be equivalent according to the *standard model of neoclassical economics*. However, in practice, they often differ, with this gap known as the *endowment effect*. This effect highlights that individuals tend to place a higher value on what they already own. The discrepancy is attributed to psychological factors such as loss aversion, perceptions of relative value, and cognitive biases [8], [9]. Practically, *WTP* helps determine consumer surplus, i.e., the difference between what a consumer is willing to pay and the actual price. Meanwhile, *WTA* assists in estimating producer surplus, i.e., the difference between the price received and the minimum acceptable amount. These indicators are essential for evaluating economic welfare, whether in contexts such as differential pricing, environmental valuation, or public policy analysis [10]. Ultimately, *WTP* and *WTA* are crucial tools for specifying agent decision-making processes and pricing behaviours in an *ABM/LMM*. During the *simulation data analysis phase*, the comparison of *WTP* and *WTA* serves as a *key indicator*, reflecting the potential for a convergence of interests between buyers and sellers within the markets of an *ABM/LMM*. These concepts thus provide a rigorous framework for studying socio-economic dynamics in spatially explicit environments, such as those modelled in *ABM/LMM*s.

B. Context of the Case Study

The primary empirical application of our work concerns the land and housing markets in Corsica, which, due to their dynamics and the scarcity of land available for residential use, are characterised by significant pressure on prices and conflicts over land use. Corsica is a geographically defined territory, surrounded by the Mediterranean Sea. It is one of the least densely populated Mediterranean regions, with the majority of its population concentrated along the coastline, with only the town of Corte, located inland, standing as an exception due to the presence of the island's sole university within its territory [11]. Since the 1970s, Corsica has experienced the development of a tourism-based economy, which today represents the island's main economic sector. This tourism expansion has led to the construction of infrastructure for mass tourism, as well as a significant rise in the number of second homes, a trend that continues to grow. By 2015, there were over 90,000 second homes in Corsica, accounting for more than 37% of the regional housing stock [12], a proportion nearly four times higher than the national average. Consequently, in cities such as Ajaccio and Bastia, as well as other tourist areas, securing affordable accommodation has become increasingly difficult. These findings raise important questions, which we aim to address through the development of a virtual computational simulation environment based on an *ABM/LMM* model. Our objective is, first, to determine how investments in properties intended for short-term tourist rentals affect the availability of housing at affordable prices for local residents. Secondly, we seek to examine, from a public policy perspective, which measures could be implemented to address these issues (e.g., zoning regulations, investment taxes, or the creation of a permanent resident status).

III. Method

A. 5-Step Simulation Iterative Modelling Process

To structure this computational modelling and simulation work within an interdisciplinary context, we employ the iterative composition-decomposition method known as the *5-Step Simulation Iterative Modelling Process* (*5-SSIMP*)[1]. This approach ensures a logical and coherent progression, from the concepts formulated in intermediate models to the validation of results obtained through simulation experiments. Its iterative nature allows for continuous adjustments to the modelling components. As illustrated in Figure 1, the *5-SSIMP* method consists of two main phases (*A.-Modelling, B.-Simulation*), and five stages (*A.1-Conceptualisation*, *A.2-Integration*, *A.3-Implementation*, *B.1-Experimentation*, and *B.2-Data Analysis*. This ensures a smooth transition from conceptual theory to operational simulation. The principle of the *5-SSIMP* method is illustrated in Figure 2. During the *A.1-Conceptualisation* stage, modellers simplify the real-world system by formulating a *conceptual model*, identifying key components and their relationships. In the *A.2-Integration stage*,

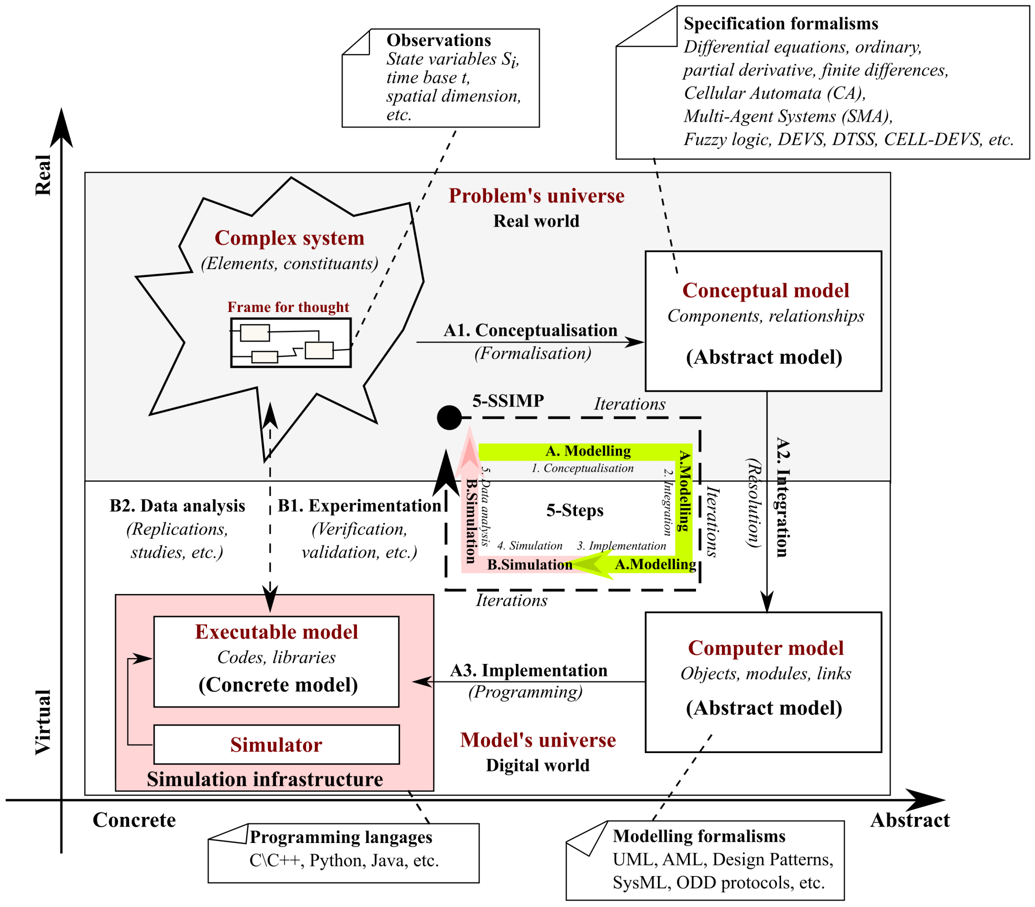


Figure 1. The *5-SSIMP* method is an iterative composition-decomposition approach for the modelling and simulation process.

modellers translate the *conceptual model* into a *computational model*, defining *objects* (classes, interfaces), *modules* (groups of objects), and appropriate *links* (relationships). In the *A.3-Implementation* stage, the *computational model* is transformed into an *executable model* (executable code modules). This process is based on *Object-Oriented Programming* (*OOP*) and *Generic Programming* (*Template Programming*) techniques [13], as well as *Design Patterns* (*DP*) [14]. During the *B-Simulation* phase, in the *B.1-Experimentation* stage, modellers must validate and verify the *executable model* through rigorous testing to ensure the accuracy and reliability of the simulated data obtained, comparing it, where possible, with real-world data. In the *B.2-Data Analysis* stage, the executable model is finalised for routine practical use, ensuring that it remains reliable, robust, and efficient over time and across evolving conditions. As illustrated in Figure 1, the *RBB* components of the *executable model* are continuously refined and validated through the iterative cycle of continuous adjustments within the *5-SSIMP* method. These iterations enable feedback loops and the integration of new elements derived from field survey data and questionnaire responses. The *5-SSIMP* method effectively structures the iterative phases of development, testing, and refinement involved in constructing *ABM/LUCC* models.

B. Reusable Building Blocks

TABLE I
LINKS BETWEEN INTERMEDIATE MODELS (ABSTRACT AND CONCRETE) AND REUSABLE BUILDING BLOCKS (RBB) IN THE 5-SSIMP METHOD.

Intermediate model	**RBB**	**Specification Tools**
A1. Conceptual	*conRBB*	Mathematics, ODEs, PDEs, etc. Formalisms, DTSS, DEVS, etc.
A2. Computational	*comRBB*	Standards, norms, UML, AML, ODD, SysML, etc.
A3. Executable	*exeRBB*	OOP, Template programming, DP, etc.

The *5-SSIMP* method is based on the creation of *Reusable Building Blocks* (*RBB*), which ensure the modularity and reusability of various modelling components. Each of these components is designed independently, facilitating its evolution from conceptual specification to implementation in an object-oriented programming language. Within this framework, generic building blocks (*conRBB*, *comRBB*, *exeRBB*) are defined, integrated, and implemented in a way that allows them to be reused and adapted for other simulation projects. The *5-SSIMP* method significantly reduces development time and associated costs by optimising collaboration among team

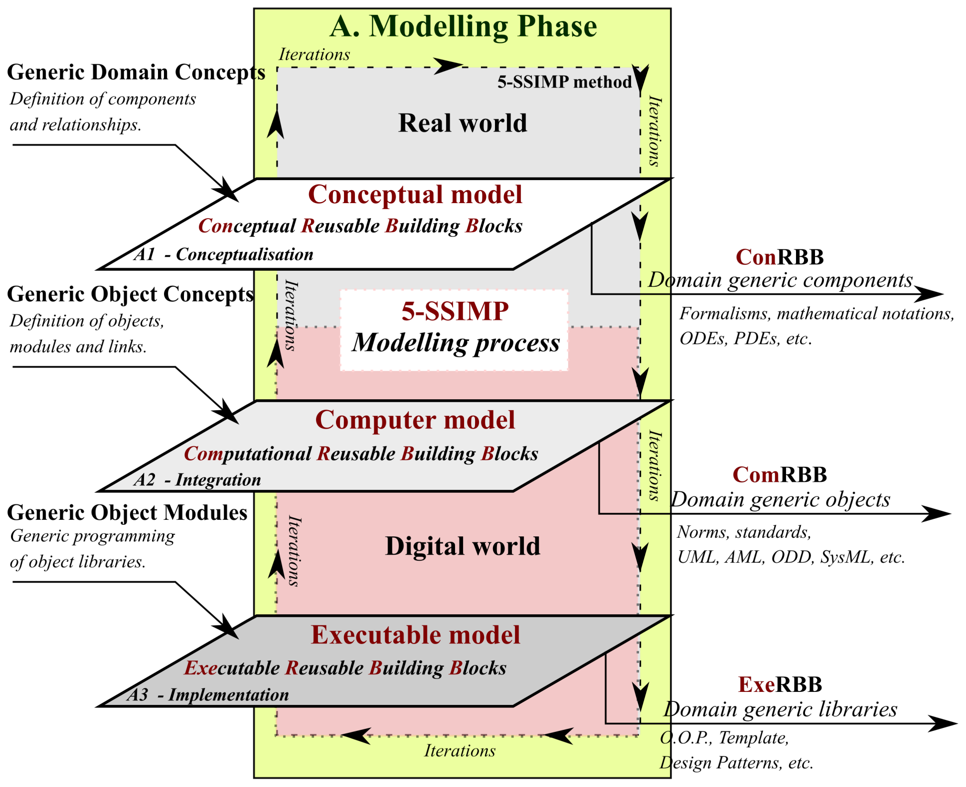

A. Modelling Phase
Iterations
5-SSIMP method
Real world
Generic Domain Concepts
Definition of components and relationships.
Conceptual model
Conceptual Reusable Building Blocks
A1 - Conceptualisation
ConRBB
Domain generic components
Formalisms, mathematical notations, ODEs, PDEs, etc.
Generic Object Concepts
Definition of objects, modules and links.
5-SSIMP
Modelling process
Computer model
Computational Reusable Building Blocks
A2 - Integration
ComRBB
Domain generic objects
Norms, standards, UML, AML, ODD, SysML, etc.
Generic Object Modules
Generic programming of object libraries.
Digital world
Executable model
Executable Reusable Building Blocks
A3 - Implementation
ExeRBB
Domain generic libraries
O.O.P., Template, Design Patterns, etc.

Figure 2. The core concept of *Reusable Building Blocks* (*RBB*) in the *A-Modelling* phase of the *5-SSIMP* method.

members involved in the modelling process. In this paper, the concepts of *WTP* and *WTA* serve as representative examples to illustrate the relevance and efficiency of the *5-SSIMP* method. These concepts are specified in the form of *RBB* to structure their modular integration into the intermediate models of the *5-SSIMP* method. Furthermore, this approach facilitates their future application in various other contexts related to land and housing markets. The production of *RBB* is embedded within the *A.1-Conceptualisation*, *A.2-Integration*, and *A.3-Implementation* stages of the *A–Modelling* phase in the *5-SSIMP* method. During these stages, modellers identify and isolate key elements characterising the socio-economic system under study, defining corresponding intermediate models, both *abstract* and *concrete*. These models incorporate both generic and specific characteristics, which must be characterised separately. Elements with generic properties are grouped into *Reusable Building Blocks* (*RBB*) within the intermediate models, serving to modularly organise structures and functionalities that can be reused in other *ABM/LUCC* models. In this context, each *RBB* represents a generic aspect of the intermediate model (either *abstract* or *concrete*), such as generic agent behaviours, generic social interactions, or generic economic dynamics. The structuring of intermediate models into *Reusable Building Blocks* (*RBB*) facilitates the iterative process by enabling:

- updates, replacements, maintenance, and evolution of the executable model (*modularity*);
- the creation and updating of code libraries, allowing validated code to be shared and reused across different simulation projects (*reusability*);
- collaboration between modellers in economics and computer science, supporting the development and evolution of new generic modules tailored to specific requirements.

C. Conceptual Reusable Building Block

A *Conceptual Reusable Building Block* (*conRBB*) is a type of *RBB* produced in the *A.1-Conceptualisation* stage of the *5-SSIMP* method. *conRBB* are formulated using specification formalisms (e.g., rules, notations, formal languages, etc.), mathematical notations, ordinary differential equations (ODEs), and system specifications (e.g., DEVS [15]). These *conRBB* structure and organise generic key components and their relationships within abstract conceptual models in a clear and structured manner.

D. Computational Reusable Building Block

A *Computational Reusable Building Block* (*comRBB*) is another type of *RBB*, produced in the *A.2 Integration* stage of the *5-SSIMP* process. *comRBB* ensure the consistency, standardisation, and reusability of generic objects or object groups across different simulation projects. They can be reused in various fields of study. *comRBB* should adhere to recognised computational norms and standards, such as *Unified Modelling Language* (*UML*) [16], *Agent Modelling Language* (*AML*) [17], *Overview*, *Design concepts*, and *Details* (*ODD*) [18], or *Systems Modeling Language* (*SysML*) [19], among others. To achieve this, modellers rely on *Object-Oriented Programming* (*OOP*) [20], *Template Programming* [13], and *Design Patterns* (*DP*) [14]. Recent studies demonstrate that *DP* significantly enhance the *modularity* and *reusability* of models in computational simulations [21]–[23].

E. Executable Reusable Building Block

An *Executable Reusable Building Block* (*exeRBB*) is the type of *RBB* produced in the *A.3 Implementation* stage of the *5-SSIMP* modelling process. These building blocks consolidate the generic code components of the executable model. This involves implementing, within a programming language, the object groupings defined in the computational model, incorporating *Templates* and *Design Patterns* where applicable. *exeRBB* are essential for computational replications carried out in the *B.1-Simulation* and *B.2-Data Analysis* stages of the *5-SSIMP method*. They consist of reusable code modules that can potentially be adapted for use in other computational simulation projects. These software libraries are designed generically, providing a significant advantage in computational simulation projects that require high productivity and software reliability.

IV. Implementation

A. Conceptual Model

The *ABM/LMM* presented as an example in this study comprises two economic markets: a *Land Market* (*LM*) and a *Housing Market* (*HM*). This work builds upon the research we initiated in 2021, focusing on the economic study and analysis of a tourist region, Corsica [24]. In this section, we provide a concrete example of modelling the decision-making processes and price formation mechanisms of agents within an *ABM/LMM*, following the implementation of the *5-SSIMP* method. We also present the *RBB*s that we have developed to represent the *WTA* and *WTP* formulated within the *ABM/LMM*.

1) Land Market - LM: In the *LM*, household agents who own land interact as sellers (`LndHse-agents` - `LM-sellers`) with real estate developer agents acting as buyers (`Dev-agents` - `LM-buyers`).

The concepts of *WTP* and *WTA* are employed to model the decision-making behaviours of economic agents involved in both the *LM* and the *HM*. These concepts serve to specify the price-setting decision processes and behavioural mechanisms of the agents within the *ABM/LMM* we are constructing. More specifically, in the *LM*, *WTA* represents the minimum amount that a `LndHse-agent` household owner is willing to accept to sell a plot of land on the *LM*. This *WTA* corresponds to the average past sale price of similar plots with identical characteristics. Conversely, *WTP* represents the maximum amount that a `Dev-agent` is willing to pay to acquire land on the *LM*. This *WTP* is determined by the `Dev-agent` real estate developer, based on the anticipated selling price of the house they intend to build on the land, while seeking to achieve a target profit margin on the total cost of the house. Interactions between the `LndHse-agents` as sellers and the `Dev-agents` as buyers in the *LM* are illustrated in Figure 3.

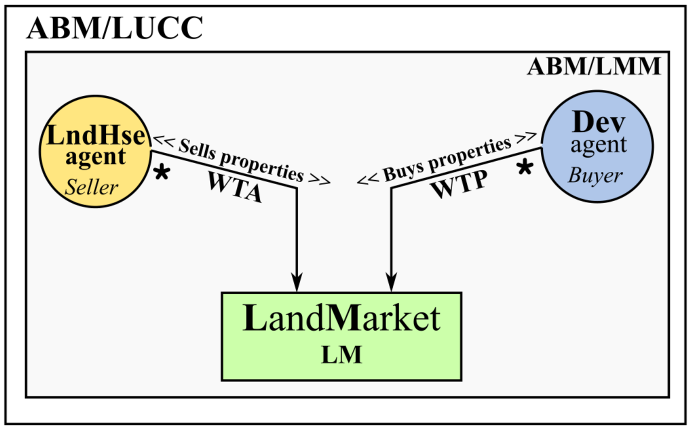


Figure 3. Interactions between household landowner agents (`LndHse-agents`) as sellers and real estate developer agents (`Dev-agents`) as buyers in the ABM/LMM (*LM*).

2) Housing Market - HM: In the *HM*, `Dev-agents` (real estate developers) resell land with newly built houses. Their *WTA* corresponds to the amount required to generate a profit margin on the total costs associated with producing the housing unit (construction costs and the price paid for the land). Within this *HM*, there are two categories of buyers. `Hse-agents` are Households seeking accommodation, whose *WTP* is determined through the maximisation of a quasi-linear utility function under budget constraints.

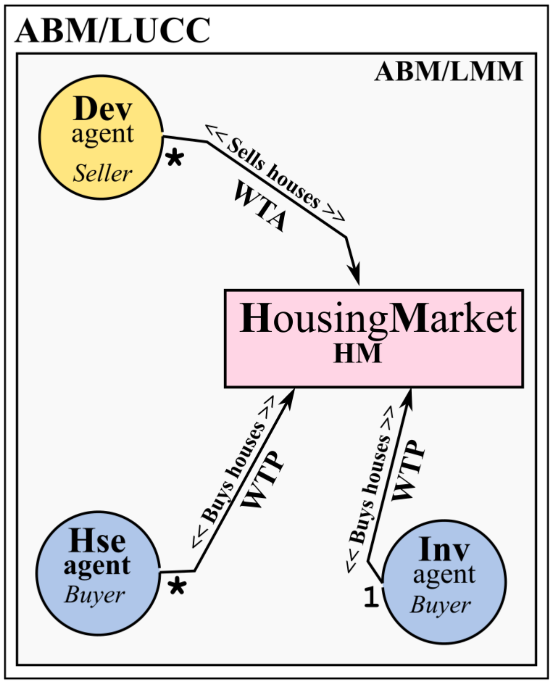


Figure 4. Interactions between real estate developer agents as sellers (`Dev-agents`), household agents as buyers (`Hse-agents`), and the single investor agent as a buyer (`Inv-agent`) in the ABM/LMM (HM).

`Inv-agent` is the single representative investor who purchases houses and rents them as tourist accommodations.

For the `Inv-agent`, *WTP* is based on the opportunity cost of investing in a tourist rental property versus an alternative financial investment of the same amount. In other words, this *WTP* follows an objective criterion whereby the expected return from purchasing a house at a given price must be equal to or greater than the return from an equivalent investment in the financial market.

3) *Agent's Interaction:* In terms of interactions, `Dev-agents` purchase land assets (plots) sold by `LndHse-agents`, with the objective of constructing a house. For the sake of conceptual model simplification, each real estate developer is specialised in constructing only one category of house $m = \{1, ..., M\}$ (e.g., detached house, chalet, bungalow, etc.), which is subsequently sold on the *HM*. In the *HM*, real estate developer agents who previously acted as buyers in the *LM* become sellers (`Dev-agents - HM-Sellers`) and interact with household agents as buyers (`Hse-agents - HM-buyers`), as well as with the single representative investor agent (`Inv-agent - HM-buyer`). Whether in the *LM* or the *HM*, the price formation processes of agents in the model are based on the key economic concepts of *WTA* and *WTP*. These two fundamental concepts are represented as *Conceptual Building Blocks* (*conRBB*) within *ABM/LMM* models.

4) *Price Formation in the LM:* Household landowner agents (`LndHse-agents`) make their decisions based on their *WTA*, represented as a *Conceptual Reusable Building Block* (*conRBB*). This decision depends on various factors, such as opportunity costs, market conditions, and the financial objectives of the agents. The *WTA* of `LndHse-agents` in the *LM* is given by Equation (1).

$$WTA_{LndHse}\left(z_i\right) = P_{t-1}\left(z_i\right) \tag{1}$$

The vector z_i represents the characteristics of the land asset being sold, while the past average sale price $P_{t-1}(z_i)$, calculated using a spatial regression model, is estimated based on the historical sale prices of similar plots with comparable characteristics z_i. Interviews conducted with real estate professionals (three estate agents and one property developer) revealed that an implicit rule prevails across various segments of the Corsican housing market: sellers and buyers tend to maintain a negotiation margin when finalising a transaction. For sellers, this margin manifests as an attempt to secure a transaction price above their reserve price, whereas for buyers, it reflects an effort to negotiate a transaction price below their maximum *WTP*.

a) $Askprice_{\text{Lwd-Agent}}$: According to our interviewees, this negotiation margin is approximately 7% of the sale price. We denote this margin a ψ, a value specific to each individual, which plays a role in the negotiation process between seller agents (`LndHse-agents`) and buyer agents (`Dev-agents`) during land transactions. It is incorporated into the calculation of the asking price set by sellers in the *LM*, $Askprice_{Lwd\text{-}Agent}$, as defined in Equation (2).

$$Askprice_{Lwd\text{-}Agent} = (1 + \psi_{Lwd\text{-}Agent}) \times WTA_{Lwd\text{-}Agent} \tag{2}$$

b) *WTP* - `Dev-agents`: The purchasing decisions of real estate developer agents (`Dev-agents`) in the *LM* are based on their *WTP*, represented as a *conRBB*, as defined in Equation (3).

$$WTP_m\left(z_i\right) = \frac{P_{H,i,t-1}\left(z_i, LivA_m\right)}{1+\pi_m} - C_{LivA}\left(z_i\right) LivA_m \tag{3}$$

$LivA_m$ represents the surface area of a house belonging to category m. π_m denotes the anticipated profit margin of the real estate developer agent. $P_{H,i,t-1}$ is the average past price of a house built on a plot of land, both of which have equivalent characteristics i corresponding to z_i. $C_{LivA}\left(z_i\right)$ represents the construction cost per unit of surface area required to build the house.

c) $Bidprice_{\text{Dev-agent}}$: Taking into account the negotiation margin ψ_m of a `Dev-agent` and in an attempt to secure a transaction price lower than their *WTP*, the purchase price ($Bidprice_m$) is given by Equation (4).

$$Bidprice_m = (1 - \psi_m) WTP_m(z_i) \tag{4}$$

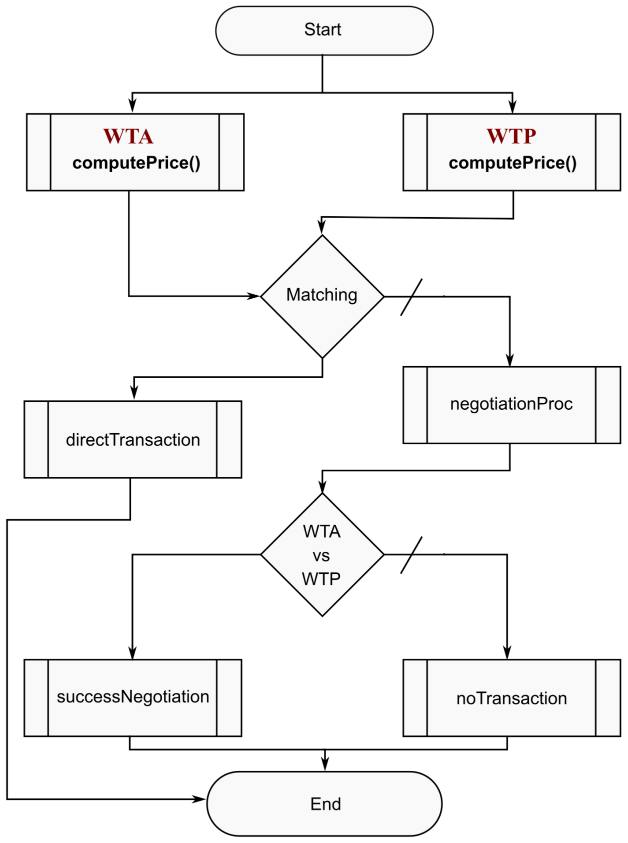


Figure 5. Flowchart of the Negotiation and Transaction Process in the LM Algorithm.

5) Negotiation and Transaction in the LM: Once all buyer agents in the *LM* have set their bid prices ($Bidprice_{m-\texttt{Dev-agent}}$) and seller agents have determined their asking prices ($Askprice_{-\texttt{Lwd-Agent}}$), the matching and exchange mechanism is executed. `Dev-agent` submit offers for available plots of land while considering their construction capacity constraints, whereas `LndHse-agents` evaluate the received offers. Seller `LndHse-agents` review all proposals and identify the highest bid ($Bidprice_{m-\texttt{Dev-agent}}$). Matching then occurs between the `Dev-agent` buyer who submits the highest offer and the corresponding `LndHse-agent` seller. Once this matching is completed, the negotiation and transaction process can begin.

a) Direct Transaction Process - DTP: If the highest $Bidprice_{max}$, submitted by the `Dev-agent`, is greater than or equal to the asking price ($Askprice_{\texttt{Lwd-Agent}}$) set by the household landowner agent, the transaction is completed immediately without negotiation. In this case, the transaction price ($\bar{P}(z_i)$) is set at the asking price ($Askprice_{\texttt{Lwd-Agent}}$) determined by the selling household agent.

b) Transaction Process with Negotiation - TDN:

$$\bar{P}(z_i) = WTA_{Hse}(z_i) + \frac{ND_i}{\chi}\left[WTP_{Max} - WTA_{Hse}(z_i)\right] \tag{5}$$

If the asking price ($Askprice_{\texttt{LndHse-agent}}$) set by the household landowner agent is strictly higher than the maximum bid price $Bidprice_{max}$ offered by the real estate developer agent (`Dev-agent`), a negotiation process may be initiated, provided that the buyer's *WTP* (`Dev-agent`) is greater than or equal to the seller's *WTA* (`LndHse-agent`). The transaction price $\bar{P}(z_i)$ is determined by Equation (5). ND_i represents the number of expressed bids for the property, while χ corresponds to the total number of `Dev-agents` for which $n_{m'} \geq 0$, with $m\prime = \{1, ..., \chi\}$. In Equation (5), the distribution of the difference between the buyer's *maximum WTP* WTP_{max} and the seller's *WTA* $WTA_{Hse}(z_i)$ depends on market conditions and the relative bargaining power of both parties. Thus, the higher the number of expressed bids (ND_i) for a property, the stronger the seller's bargaining power, leading to an increase in the transaction price. Conversely, when the number of expressed bids is low, the buyer's bargaining power increases, resulting in a lower transaction price.

c) No Transaction: Finally, if the asking price ($Askprice_{\texttt{LndHse-agent}}$) set by the household landowner agent strictly exceeds the maximum bid price ($Bidprice_{max-\texttt{Dev-agent}}$) and the *WTA* ($WTA_{LndHse}$) of the landowning household is greater than the *WTP* (WTP_m) of the `Dev-agents`, no transaction can take place for this plot.

6) Price Formation in the HM: The *HM* represents the interaction between "*real estate developer agents*" `Dev-agents`, who become sellers of their properties, "*household agents*" `Hse-agents` acting as buyers, and a single "*investor agent*" buyer *`Inv-agent (HM-buyer)`*.

In this market, the *WTA* (*ConRBB*) of the *Dev*-agents selling their houses is given by Equation (6):

$$WTA_{i,m}(z_i, LivA_m) = (1+\pi_m)\left(\overline{P_i}(z_i) + C_{LivA}LivA_m\right) \tag{6}$$

where $\overline{P_i}(z_i)$ represents the price paid for the parcel i with characteristics z_i.

A negotiation margin similar to that used by sellers and buyers in the *LM* is also applied by economic agents in the *HM*. Sellers initially propose an asking price above their *WTA*, attempting to achieve their individual negotiation margin. Similarly, buyers initially propose a bid price below their *WTP*, based on their individual negotiation margin.

To avoid unnecessarily complicating the exposition of the conceptual model, the explicit relationships linking sellers' *Ask prices* to their *WTA* and buyers' *Bid prices* to their *WTP* in the *HM* are not presented here.

Following the maximisation of a quasi-linear utility function under budget constraints, the *WTP* for household agents seeking accommodation is defined by Equation (7):

$$\text{WTP}_c(\eta) = \frac{(1+\delta_c)\left[(1+\delta_c)^T - 1\right]}{\delta_c}\theta_c(\eta) \tag{7}$$

where δ_c is the household agent's discount rate, T is the average duration of a real estate loan, and $\theta_c(\eta)$ is the function defined by Equation (8), based on [25]:

$$\theta_c(\eta) = \frac{YD_c\left(V_c^{Max}\right)^2}{b^2 + \left(V_c^{Max}\right)^2} \tag{8}$$

where:

- YD_c represents the monetary amount allocated by the household to housing each period;
- b represents the slope of their bid function.

7) HM: Purchase Process of the `Inv-agent`: The purchasing decision-making process of the `Inv-agent` buyer relies on the *WTP* (*conRBB*) described by Equation (9):

$$\text{WTP}_I(\eta) = \frac{1-(1+r)^{-T}}{r(1-\rho)(1+r)^{-T} + rT - 1 + (1+r)^{-T}}\gamma\zeta\varphi(\eta) \tag{9}$$

where:

- η: vector of the house characteristics;
- $r \in [0,1]$: interest rate in the financial market;
- $\gamma \in [0,1]$: coefficient of net rental yield (net of maintenance costs);
- ζ: number of days in the tourist season;
- $\varphi(\eta)$: average daily revenue of a tourist residence with similar characteristics;
- $\rho \in [0,1]$: residual value coefficient of the house after T years.

The *conRBB* of the `Inv-agent` involves a limited budget, denoted as Ω, representing the total investment in the *HM* over a given period. The agent can choose between two investment types: a financial market investment or a real estate investment.

8) Negotiation and Transaction in the HM: The negotiation and transaction algorithm of the *HM* is similar to that of the *LM*. To avoid unnecessarily complicating our presentation, and due to the similarity of these two algorithms—particularly their reliance on the *WTA* and *WTP* of seller and buyer agents—this algorithm is not detailed here.

For a more detailed description of the *conRBB* used in this article's conceptual model, the interested reader can refer to [26] and [24].

B. Computer Model

In this section, we present the computational integration of the *ABM/LMM* and its *computational Reusable Building Blocks* (*comRBB*) related to the generic concepts of *WTP* and *WTA*, previously formulated as conceptual *Reusable Building Blocks* (*conRBB*).

1) Design patterns: In computational modelling, the number of objects can rapidly become very large, especially when multiple organisational levels are involved. In such contexts, modellers must address the challenge of defining numerous objects and relationships.

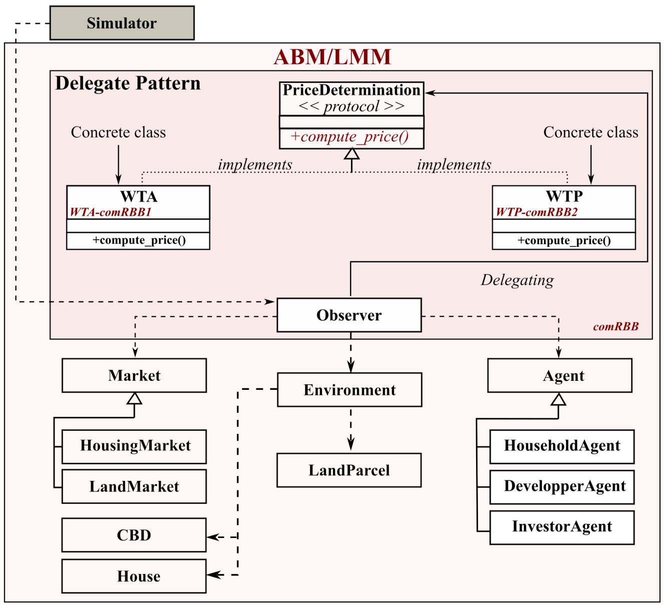


Figure 6. In the computer model, *comRBB* related to *WTP* and *WTA* are implemented following the *Delegation* design pattern.

The use of *design patterns* helps manage this complexity, allowing structurally well-defined and organised *comRBB* to be implemented and improved independently. The concept of design patterns originates from the seminal work *A Pattern Language: Towns, Buildings, Construction* [27] by architects *Christopher Alexander*, *Sara Ishikawa*, and *Murray Silverstein*. During the 1970s, these authors defined a pattern as: "*Each pattern describes a problem, which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice*". This statement defines a design pattern as a recurring problem with its associated solution, a specific context, an *architecture*, and the expression of the associated *generic solution*. Subsequently, the concept of design patterns was introduced into computer science during the 1970s with object-oriented programming (OOP), notably through the renowned *Model-View-Controller* (MVC) pattern [28]. The MVC pattern was initially proposed by *Tryve Reenskaug* as a generic solution for complex data-handling problems [29]. Towards the late 1980s, the contributions of *Reid Smith* also supported their use in computer science [30], alongside the influential article by *Kent Beck* and *Ward Cunningham*, which proposed adapting design patterns to OOP through five interface-design compositions [31].

TABLE II
CHARACTERISTICS OF A *DESIGN PATTERN* IN COMPUTER SCIENCE ACCORDING TO THE "*GoF*" FORMALISM [32].

Vocabulary Element	**Meaning**
Name	Name of the design pattern.
Problem	Description of the problem addressed by the *Design Pattern*.
Initial context	The context to which the pattern applies.
Forces	Description of situations where the *Design Pattern* is applicable.
Solution	Components of the solution and their relationships.
Examples	Examples of application.
Consequences	Description of how the *Design Pattern* achieves its goal.
Logic	Description of the logic implemented.
Related patterns	Closely related *Design Patterns* referenced here.

However, it was not until 1995 that the use of *Design Patterns* became widespread within OOP, particularly due to the influential work of the "*GoF*" (Gang of Four)—*Erich Gamma*, *Richard Helm*, *Ralph Johnson*, and *John Vlissides* [32]. This book standardised precise vocabulary and formalisation for design patterns, now commonly adopted in computer science literature (see Table II). Since then, *Design Patterns* have progressively promoted unambiguous modular and reusable code expression, facilitating best practices and expertise dissemination among designers and developers [28], [33], [34].

Recognising the valuable knowledge encapsulated in expert-driven design patterns, we propose using them to integrate the *comRBB* of the ABM/LMM into the computer model, ultimately aiming at implementing modular and reusable *exeRBB* in the executable model.

Additionally, using *Design Patterns* in computational simulation promotes reproducibility by precisely and completely describing generic object-based model elements. Consequently, the implemented *exeRBB* code is easily understandable and replicable by the scientific community. Using established *Design Pattern* names helps modellers clearly explain, justify, and communicate the structure of their *comRBB* and *exeRBB*, ensuring reproducibility in executable implementations from the conceptual stage onwards.

2) NetLogo Design Pattern: In an interdisciplinary scientific context, the *NetLogo* simulation platform is preferred for initial prototyping of *reusable building blocks* (*RBB*) within agent-based *ABM/LMM* models.

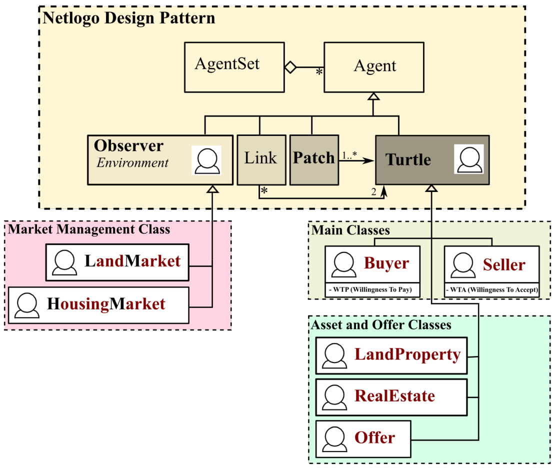


Figure 7. The *NetLogo Design Pattern* allows rapid construction of an *ABM/LMM* prototype using only four generic types of components (*comRBB*): an omniscient *observer*, *patches*, *turtles* (mobile agents), and relational links (*links*) between agents [1].

NetLogo simplifies modelling by providing four generic types of reusable components (*comRBB*): an omniscient observer (*Observer*), spatially explicit fixed cells (*Patches*), mobile agents (*Turtles*), and social connections (*Links*). These components significantly facilitate the initial modelling stage of *ABM/LMM* by enabling rapid prototyping and systematic testing of generic behaviours implemented within executable reusable building blocks (*exeRBB*). This structuring of the *NetLogo* simulation platform (cf. Figure 7), as proposed by *Seth Tisue* [35], facilitates modelling through clear modular organisation, thereby promoting rapid prototyping and verification of *RBB*s. Nevertheless, we restrict the use of *NetLogo* to the initial prototyping phase of *RBB*, given its limitations when dealing with intensive real-world data processing tasks, for which higher-performance object-oriented languages, such as Python or C++, are more suitable.

3) Delegation Design Pattern: In computational integration phases of an *ABM/LMM*, the description of numerous objects and relationships can quickly increase complexity, exacerbated by extensive inheritance use. However, inheritance creates strong coupling among objects, limiting modularity, evolvability, and reusability. To ensure optimal computer model structuring and avoid such restrictive couplings, an alternative delegation-based approach is necessary. The *Delegation Design Pattern* precisely addresses this challenge, allowing an object to delegate a specific part of its behaviour to another specialised object without direct inheritance, thus reducing strong coupling. In this work, the architectural choice of delegation, outlined in Figure 6, enables the creation of modular, maintainable, and easily evolvable *comRBB* for *WTA* and *WTP*. The characteristics of the delegation design pattern are summarised in Table III. In Figure 6, the price determination classes `WTA` and `WTP` both implement the abstract method `compute_price()` defined in the common interface *PriceDetermination*. The abstract class `PriceDetermination` specifies a common interface protocol for different price determination strategies within the *ABM/LMM* model. Concrete classes `WTA` and `WTP` represent specialised objects responsible for implementing their specific strategies to determine asset prices, thereby encapsulating the model's economic concepts of *WTA* and *WTP*. This approach notably prevents coupling between objects of the final executable model and those from the simulation infrastructure. The resulting object-oriented code is significantly more modular and reusable; dependencies between objects of the executable model and those of the simulation infrastructure are thus substantially reduced [36]. Thus, it is essential to design the components of the computer model with design patterns in mind, to ensure modularity and reusability.

TABLE III
CHARACTERISTICS OF THE *DELEGATION* DESIGN PATTERN ACCORDING TO THE *GoF* FORMALISM.

Vocabulary item	**Meaning**
Name	*Delegation*
Problem	Ensuring that an object can transfer or delegate part of its behaviour to another object without resorting to inheritance.
Initial context	When a component must execute an operation, but the processing varies depending on context. It is desirable to avoid strong coupling or a rigid class hierarchy.
Forces	- Reduction of coupling between classes. - Easier code reuse. - Improved maintainability.
Solution	Introduce a *delegate* object responsible for the desired behaviour. The primary class redirects the call to this object, allowing the behaviour to be specialised or modified dynamically.
Examples	- Implementation of multiple behaviours in agent-based simulations. - Management of different calculation strategies (pricing, behaviour, etc.) via a separate object.
Consequences	- Enables combining functionalities without multiplying subclasses. - Makes the system more flexible and modular. - Simplifies future modifications or extensions.
Logic	A primary object holds a reference to a secondary object (the delegate). Calls related to specific behaviour are transferred to this delegate rather than being implemented directly.
Related patterns	*Strategy* (for algorithm specialisation), *Adapter* (for interface compatibility), *Decorator* (for dynamic addition of responsibilities).

C. Executable Model

The *delegation* design pattern is implemented in accordance with the organisation of the *WTA-comRBB* and *WTP-comRBB* components of the computational model shown in Figure 6. The code is structured into abstract classes (`PriceDetermination`) to facilitate the implementation of the *delegation* design pattern and to implement the *WTA-exeRBB* and *WTP-exeRBB*. As a reminder, *abstract classes* are fundamental tools in object-oriented programming, allowing code to be structured in a modular and reusable way. They define a common interface that multiple subclasses can implement, thus ensuring a consistent organisation of the code and facilitating its extension. Their use in the executable model promotes separation of responsibilities, with each subclass being responsible for implementing its own behaviours while maintaining compatibility with the overall model. Abstract classes also enable the addition of new functionalities without altering the existing object structure, thereby enhancing the flexibility and scalability of the model. In this perspective, as illustrated in Figure 6, the creation of agents within the computational model follows a modular approach, externalising decision-making strategies related to *WTP* and *WTA* into specialised object codes (*exeRBB*). This design avoids direct implementation of price formation calculations within agent classes, delegating this responsibility to dedicated objects. The implemented *exeRBB* thus allows agents to adopt various pricing strategies without altering their internal structure. Furthermore, this approach facilitates the addition or modification of strategies without requiring a complete overhaul of agent class codes, while ensuring the reusability of these modules in other modelling contexts.

Listing 1. The abstract class (`PriceDetermination`) implements the *delegation* design pattern in *Python*.

```python
from abc import ABC, abstractmethod
class PriceDetermination(ABC):
 """
 Abstract class defining a common interface for price
 decision strategies related to WTA and WTP.
 Abstract methods:
  compute_price(): Method to calculate the price based on
  the agent’s attributes and asset characteristics.
  """
 def __init__(self,delegate, **kwargs):
  """
  Abstract constructor that accepts a variable number of
  keywords.
  arguments to provide flexibility for derived
  strategies.
  Args:
     **kwargs:  Dictionary containing named arguments
     specific to derived classes (here WTP or WTA RBBs).
  """
  self.__kwargs = kwargs
  self.delegate = delegate
  # Additional initialisation specific to subclasses can
  be added here.
  @abstractmethod
  def compute_price(self):
   """
   Abstract method to calculate price (i.e., WTP or WTA).
   Returns:
   float: Computed price according to the decision-making
   strategy implemented by subclasses.
   """
   pass
```

In order to implement this object-oriented architecture, the `PriceDetermination` module presented in Listing 1 is defined as an abstract class serving as a common interface for economic agents’ decision-making processes. This interface requires implementation of the abstract method `compute_price()`, which models the price-formation calculation according to market dynamics. Thus, the `PriceDetermination` object in Listing 1 provides a common basis for implementing decision-making strategies related to WTA and WTP.

Listing 2. Implementation of the *exeRBB* `WTA` in *Python*.

```python
from PriceDetermination import PriceDetermination
import math
class WTA(PriceDetermination):
 """
 Represents the minimum acceptable price (WTA) for
 selling an asset. This class computes the WTA using
 a regression model where the natural logarithm
 of the price is a function of various parameters
 and regression coefficients.
 """
 def __init__(self, **kwargs):
  """
  Initialise a WTA instance.
  Keyword arguments include:
  - past_land_price: Historical land price.
  - regressCoeff_k0 to regressCoeff_kN:
  Regression coefficients for the model.
  - parcel_surface: Surface area of the parcel.
  - beach_distance: Distance to the beach.
  - cbd_distance: Distance to the central business
  district.
  - sea_viewIndex: Index representing the quality of
  the sea view.
  - currentTime: The current time variable.
  """
  # Initialise the superclass with the provided keyword ↪
     arguments.
  super().__init__(self, **kwargs)
  # Dynamically extract regression coefficient keys
  from kwargs.
  # This ensures that the number of coefficients is
  # determined by the input.
  coeff_keys = sorted([key for key in kwargs if key.↪
     startswith("regressCoeff_")], key=lambda x:int(x.↪
     split("_k")[1]))
  # Create a dictionary mapping ’k{i}’ to its
  # corresponding coefficient.
  self.coeffs = {f’k{int(key.split("_k")[1])}’: kwargs.↪
     get(key, 0.0) for key in coeff_keys}
  # Retrieve additional parameters with default values if
  # not provided.
  self.past_land_price = kwargs.get(’past_land_price’↪
     ,0.0)
  self.parcel_surface = kwargs.get(’parcel_surface’,0.0)
  self.beach_distance = kwargs.get(’beach_distance’,0.0)
  self.cbd_distance = kwargs.get(’cbd_distance’,0.0)
  self.sea_viewIndex = kwargs.get(’sea_viewIndex’,0.0)
  self.currentTime = kwargs.get(’currentTime’,0.0)

  def compute_price(self):
   """
   Compute the minimum acceptable price (WTA) using
   the regression formula: ln(price) = k0 +
   k1 * log(parcel_surface) +
   k2 * (log(parcel_surface))^2 +
   k3 / beach_distance + k4 / cbd_distance +
   k5 * sea_viewIndex + k6 * currentTime
   Returns:
   The computed price, obtained as the exponential of:
   ln(price).
   In the event of a division by zero, 0.0 is returned.
   """
   try:
    # Compute the natural logarithm of the parcel’s
    # surface area.
    log_parcel = math.log(self.parcel_surface)
    # Calculate the natural logarithm of the price based
    # on the regression model.
    ln_price = (
    self.coeffs.get('k0', 0.0) +
    self.coeffs.get('k1', 0.0) * log_parcel +
    self.coeffs.get('k2', 0.0) * (log_parcel ** 2) +
    self.coeffs.get('k3', 0.0) / self.beach_distance +
    self.coeffs.get('k4', 0.0) / self.cbd_distance +
    self.coeffs.get('k5', 0.0) * self.sea_viewIndex +
    self.coeffs.get('k6', 0.0) * self.currentTime
    )
    # Return the computed price by exponentiating
    # ln(price).
    return math.exp(ln_price)
  except ZeroDivisionError as e:
    # Error message in case a division by zero occurs.
    print("Error computing WTA: division by zero.", e)
    return 0.0
```

In Listing 2, the code excerpt demonstrates how the *exeRBB* `PriceDecision` specifically implements the `compute_price()` method, which is used to calculate the minimum acceptable price for a seller agent's *WTA* in the *ABM/LUCC*.

Similarly, the `WTP` class shown in Figure 6 implements the *WTP* strategy for buyer agents by determining the maximum price a buyer agent is prepared to offer. This amount is calculated based on the agent's available budget, anticipated selling price, estimated construction costs, and desired profit margin. The modules `WTA` and `WTP` *exeRBB* inherit from the `PriceDetermination` class and implement their own computational logic through the methods `compute_price()`, tailored to the specific strategies of seller and buyer agents, respectively. Thus, agents within the model can instantiate a `WTA` or `WTP` object and delegate the price calculation to it, without directly handling the complexity of these decisions. This organisation, structured into increasingly specialised implementations of `PriceDetermination`, promotes a modular, scalable, and reusable architecture, ensuring a clear separation of responsibilities and enhanced flexibility in adapting the model to various economic scenarios. The *exeRBBs* thus defined not only contribute to the modularity and clarity of the executable model's code but also facilitate its adaptation and reuse in other *ABM/LMM* modelling contexts. This enhances the modularity, reusability, and flexibility of the model by allowing the dynamic customisation of behaviours within the *exeRBBs*.

V. Discussion and preliminary results

Owing to its modular architecture based on *RBBs*, the developed *ABM/LMM* exhibits high flexibility, enabling extensive testing, fine-tuning, and comparative evaluation of various simulation scenarios. Within this context, initial simulation experiments focus on analysing the behaviour of previously developed *RBBs*. To this end, several land and real estate policy tests scenarios were considered to evaluate how the *WTA* and *WTP RBBs* influence price formation and market dynamics. Examples of these test scenarios, preliminarily validated through prototyping in *NetLogo* [35], for the validation of *RBBs* [1], are presented and discussed in detail in Table IV.

Each scenario is based on varying configurations of the *WTA* and *WTP RBBs*, enabling a detailed analysis of economic,

TABLE IV
Example scenarios of land and real estate policies simulated in Corsica.

Scenario	Description
BAU	This baseline scenario extrapolates current trends, serving as a comparative reference to evaluate the effects of more interventionist policies.
BTRI	This scenario explores the impact of a total prohibition of new tourist rental investments, reflecting drastic yet plausible policy measures to curb tourist accommodation saturation.
TTRI	A tax varying from 0.01 to 0.5 is imposed on revenues derived from tourist rental investments. This measure aims to reduce profitability, limiting investment expansion and influencing *LM* dynamics.
CZ	This scenario mandates a minimum compulsory distance (between 1 and 50 distance units) from the coastline for any new tourist rental investment. The goal is to protect coastal zones by reducing development concentration near sensitive habitats.
CBD-Z	Similar to coastal zoning, this policy imposes a mandatory minimum distance (1 to 50 distance units) from the city centre (*CBD*) for new tourist rental investments. Its objective is to redistribute tourist accommodations across the territory, alleviating urban centre pressures and encouraging development in less exploited areas.

TABLE V
Fixed Parameters

Input	Value
Percentage of low revenue households	54%
Percentage of middle revenue households	20%
Percentage of high revenue households	26%
Percentage of households that own their home	52%
Percentage of households that own a buildable parcel of land	87%
Share of households income spent on housing	0.3
Time Horizon (year)	20
Transport cost (€/km)	0.5404
Number of buildable parcels of land	2,500
Number of resident households	5,625
Number of developers	4
Number of Rent days	220
Slope of the bid function	0.8

regulatory, and behavioural parameters influencing agents' price-setting processes. These variations allow a nuanced examination of public policy impacts on land and real estate dynamics, particularly regarding price setting, market pressures, and housing accessibility. Table V summarises the fixed parameters used throughout simulation experiments, which remain constant to provide reference points.

Table VI presents parameters randomly assigned values following a uniform distribution, introducing variability into the simulations.

Data sources for these simulation experiments include the detailed *PERVAL* database provided by the Chamber of Notaries, offering various land and property market indicators, and *AirDNA*, which supplies data on short-term holiday rentals to estimate average daily revenue from tourist properties. Additional datasets from the *INSEE* (French National Institute of Statistics and Economic Studies) provide demographic and economic parameters. Supplementary data sources include references [37] for construction sector data and [38] for transportation costs to the CBD, alongside valuable insights from discussions with real estate agents and property developers.

As an example, we present a selection of preliminary simulation results obtained using the parameters listed in Table VII.

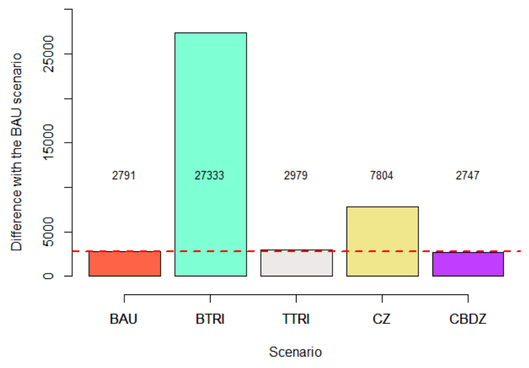


Figure 8. Number of developer bankruptcies by scenario.

Analysing these results, we extensively examine the impacts of agent interactions on Corsica's land and real estate dynamics. Each simulated policy scenario illuminates potential public policy effects on urban development and *HM* evolution. Detailed examination of developer bankruptcies, market price variations, distances to strategic interest points such as the *Central Business District* (*CBD*) and coastline, as well as the *Mean Sea View Index*, provided particularly insightful findings. These indicators offer valuable interpretations of the economic, social, and environmental consequences of each policy. Specifically, we detail developer bankruptcies and average price trends observed across both studied markets for each simulated scenario.

TABLE VI
RANDOM UNIFORM VALUE PARAMETERS

Input	Value
Land Area (m^2)	$[500, 2500[$
Sea view Index	$[0.75, 23.25[$
Disposable budget of low revenue households	$[6200, 23200[$
Disposable budget of middle revenue households	$[10000, 40000[$
Disposable budget of high revenue households	$[11500, 71500[$
Disposable budget of Investor	$[900000, 1100000[$
Interest rate	$[0.02, 0.04[$
Discount rate	$[0.01, 0.11[$
Leeway	$[0.05, 0.09[$
Net return of rental	$[0.65, 0.75[$
House loss value	$[0.08, 0.12[$
Margin rate	$[0.02, 0.024[$
Cost Parameter (€/m^2)	$[750, 1000[$
Max Building sites	$[1, 3[$
Number of Bedrooms	$[1, 6[$
House surface : $a + b(\text{N. of Bedrooms} - 1)$	$a : [18, 31[, a : [24, 33[$
Building time : $8 + BTSup$	$BTSup : [0, 4[$

TABLE VII
EXPERIMENTAL PARAMETERS AND DATA SOURCES.

Aspect	Details
Initial	Experiments start with an initial seed of 0 to ensure reproducibility.
Replications	Simulations ran for a total of 2,500 replications over 40 time steps, with each step representing a semester.
Environment	The world is a grid of 501×501 = 251,001 patches, with the *CBD* located at coordinates (20, 50).
Data	The *ABM/LUCC* heavily relies on real-world data for accuracy, including results from four spatial regressions and a hedonic model categorising households by income classes.

A. Developer Bankruptcies

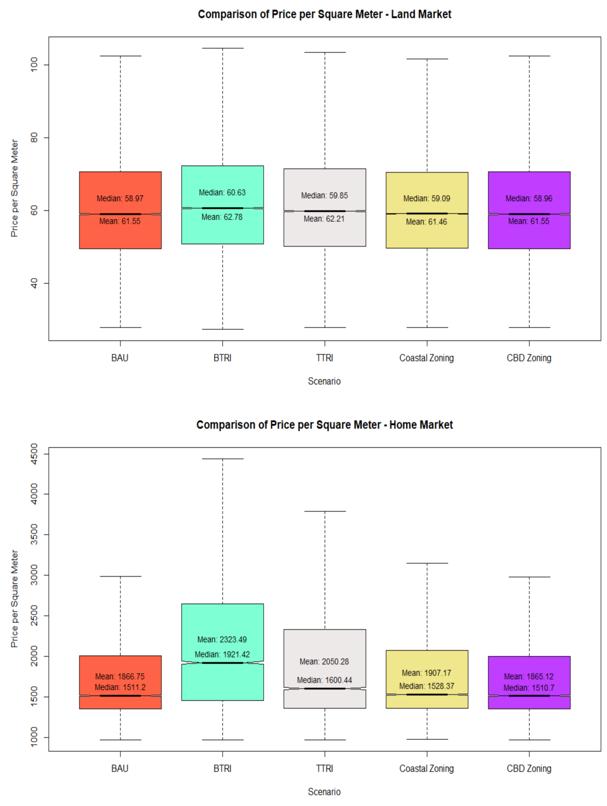


Figure 9. Dispersion of price per square meter for *LM* and *HM*.

Figure 8 reveals significant variation in developer bankruptcies across scenarios. Developer bankruptcies totalled 2,791 under the *BAU* scenario, escalating sharply to 27,333 under the *BTRI* scenario. Scenarios *TTRI*, *CZ*, and *CBD-Z* demonstrate relatively stable or mildly favourable conditions for developers, highlighting the delicate balance required between stringent regulation and market economic sustainability.

B. Market Price Variations

Figure 9 demonstrates *LM* resilience, displaying only minor price fluctuations across scenarios.

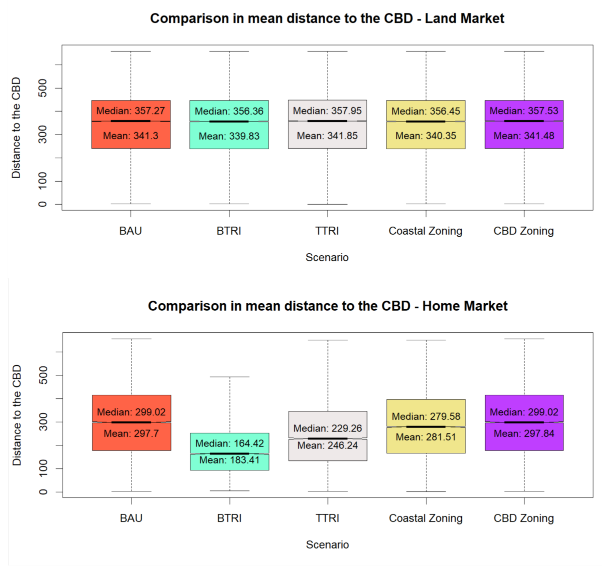


Figure 10. Comparison with the mean distance to the *CBD* in the *LM* and *HM* markets.

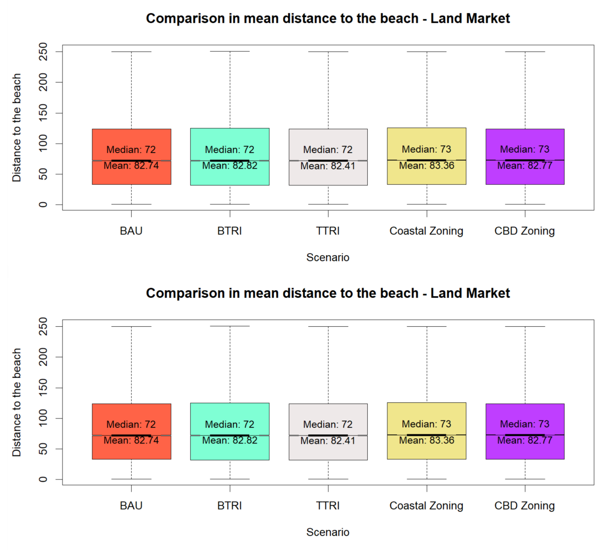


Figure 11. Comparison with the mean distance to the Beach in the *LM* and *HM* markets.

While the *LM* generally withstands simulated policies, the *HM* reacts significantly to restrictive scenarios, especially *BTRI*, with notably higher price increases. This pattern aligns with typical market responses under restrictive conditions, underscoring substantial impacts of such policies on supply-demand dynamics.

C. Influence of Central Business District (CBD)

Figure 10 assesses the scenarios' effects on average transaction distances to the CBD. The *BTRI* scenario notably induces a shift in investor-to-household purchasing, reducing average distances to the CBD, thus indicating household-driven spatial realignment towards urban centres.

Investor-driven submarkets show less sensitivity to CBD-based zoning policies, highlighting investors' resistance to such spatial regulation.

D. Distance to the Coastline

Figure 11 analyses average distance variations to the coastline across scenarios, revealing increased distances in the *CZ* scenario, especially among investors. Conversely, *BTRI* triggers decreased coastal distances in the *HM*, suggesting spatial substitution and a shift favouring household proximity to coastal areas. These observations illustrate the spatial behavioural impacts regulatory policies exert on economic agents, shaping territorial development dynamics.

VI. Conclusion and Future Work

In this article, we have presented the *5-SSIMP* modelling method, based on the design, integration, and implementation of *RBB*s. We specifically focused this study on the economic concepts of *WTA* and *WTP*. By applying the iterative composition-decomposition modelling method *5-SSIMP* to the development of an *ABM/LMM*, we demonstrated how these modular blocks (conceptual, computational, and executable) effectively structure price formation dynamics and economic agents' decision-making processes. Applying this method to the case study of *LM* and *HM* in Corsica, concretely illustrated the benefits brought by the use of the *Delegation Design Pattern* in the development of *exeRBB*s in Python. This architectural choice significantly improves the modularity, flexibility, and reusability of the executable model, while simplifying the integration of complex economic concepts such as *WTA* and *WTP*. The modular approach proposed here offers several key advantages: it reduces development time and associated costs; ensures enhanced robustness of the software code; and facilitates reproducibility of simulation experiments. Furthermore, it promotes effective collaboration between economists and software development specialists, enabling easier adaptation of models to different territorial contexts or new research objectives. In terms of future research, this work opens several promising avenues. Firstly, it would be valuable to extend experimentation by diversifying public policy scenarios further, aiming to better understand their impacts on socio-economic and environmental dynamics. Secondly, integrating *exeRBB*s as modules within a dedicated Python-based software infrastructure, would facilitate the construction, parametrisation, and simulation of these models across various geographical and thematic contexts. Lastly, this approach also opens promising interdisciplinary perspectives by fostering dialogue between economic and computer science modellers.

Our findings rely on calibrating the model to a single island—Corsica—which constitutes one of the limitations of our study. Addressing this constraint will be the focus of future work, notably by applying the model to other territories across the Mediterranean basin to assess its external validity.

References

[1] E. Innocenti, D. Prunetti, M. Delhom, and C. Idda, "Reusable Building Blocks for Agent-Based Simulations: Towards a Method for Composing and Building ABM/LUCC", in *The 16th International Conference on Advances in System Modeling and Simulation (SIMUL 2024)*, 2024, pp. 28–33.

[2] U. Berger *et al.*, "Towards Reusable Building Blocks for Agent-Based Modelling and Theory Development", *Environmental Modelling & Software*, vol. 175, pp. 1–12, 2024.

[3] D. C. Parker, "An Economic Perspective on Agent-Based Models of Land Use and Land Cover Change", in *The Oxford Handbook of Land Economics*, J. Duke and J. Wu, Eds., Oxford University Press, 2014, ch. 16, pp. 402–429.

[4] D. C. Parker, D. G. Brown, J. G. Polhill, P. J. Deadman, and S. M. Manson, "Illustrating a New "Conceptual Design Pattern" for Agent-Based Models of Land Use via Five Case Studies—the MR POTATOHEAD Framework", in *Agent-Based Modelling in Natural Resource Management*, Universidad de Valladolid, 2008.

[5] D. C. Parker *et al.*, "MR POTATOHEAD: Property Market Edition | Development of a Common Description Template and Code Base for Agent-Based Land Market Models", in *15th Annual Social Simulation Conference (SSC 2019)*, 2019.

[6] H. J. Davenport, "Proposed Modifications in Austrian Theory and Terminology", *The Quarterly Journal of Economics*, vol. 16, no. 3, pp. 355–384, 1902.

[7] J. Schmidt and T. H. A. Bijmolt, "Accurately Measuring Willingness to Pay for Consumer Goods: A Meta-Analysis of the Hypothetical Bias", *Journal of the Academy of Marketing Science*, vol. 48, pp. 499–518, 2020.

[8] D. Kahneman, J. L. Knetsch, and R. H. Thaler, "Experimental Tests of the Endowment Effect and the Coase Theorem", *Journal of Political Economy*, vol. 98, no. 6, pp. 1325–1348, 1990.

[9] W. L. Adamowicz, V. Bhardwaj, and B. Macnab, "Experiments on the Difference between Willingness to Pay and Willingness to Accept", *Land Economics*, vol. 69, no. 4, pp. 416–427, 1993.

[10] C. A. Vossler, S. Bergeron, M. Doyon, and D. Rondeau, "Revisiting the Gap between the Willingness to Pay and Willingness to Accept for Public Goods", *Journal of the Association of Environmental and Resource Economists*, vol. 10, no. 2, pp. 413–445, 2023.

[11] A. Huyssen and R. Malleville, "En Corse, 86 % de la Population Vit dans l'Aire d'Attraction d'une Ville", *Insee Flash Corse*, vol. 54, 2020, https://www.insee.fr/fr/statistiques/4808381.

[12] H. Touzani, "Des Résidences Principales et Secondaires en Forte Croissance", *Insee Flash Corse*, vol. 32, 2018, https://www.insee.fr/fr/statistiques/3571002.

[13] D. Vandevoorde and N. M. Josuttis, *C++ Templates: The Complete Guide*. Addison-Wesley Professional, 2002, ISBN: 978-0201734843.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Abstraction and Reuse of Object-Oriented Design", in *ECOOP'93—Object-Oriented Programming: 7th European Conference, Kaiserslautern, Germany*, Springer, 1993, pp. 406–431.

[15] M. J. Blas, S. Gonnet, and B. P. Zeigler, "Towards a Universal Representation of DEVS: A Metamodel-Based Definition of DEVS Formal Specification", in *2021 Annual Modeling and Simulation Conference (ANNSIM)*, IEEE, 2021, pp. 1–12.

[16] I. Jacobson, J. Booch, and G. Rumbaugh, *The Unified Modeling Language Reference Manual*. Addison-Wesley, 2021.

[17] R. Cervenka and I. Trencansky, *The Agent Modeling Language—AML: A Comprehensive Approach to Modeling Multi-Agent Systems*. Springer Science & Business Media, 2007.

[18] V. Grimm *et al.*, "The ODD Protocol for Describing Agent-Based and Other Simulation Models: A Second Update to Improve Clarity, Replication, and Structural Realism", *Journal of Artificial Societies and Social Simulation*, vol. 23, no. 2, 2020.

[19] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, 2014.

[20] M. Aniche, J. Yoder, and F. Kon, "Current Challenges in Practical Object-Oriented Software Design", in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, IEEE, 2019, pp. 113–116.

[21] M. Najdek, M. Paciorek, W. Turek, and A. Byrski, "Three New Design Patterns for Scalable Agent-Based Computing and Simulation", *Informatica*, vol. 35, no. 2, pp. 379–400, 2024.

[22] M. Rudolph, S. Kurz, and B. Rakitsch, "Hybrid Modeling Design Patterns", *Journal of Mathematics in Industry*, vol. 14, no. 1, p. 3, 2024.

[23] L. Serena, M. Marzolla, G. D'Angelo, and S. Ferretti, "Design Patterns for Multilevel Modeling and Simulation", in *2023 IEEE/ACM 27th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, IEEE, 2023, pp. 48–55.

[24] D. Prunetti *et al.*, "ABM/LUCC of a Complex economic system of land and home markets facing an intense residential development", in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2021, pp. 1–8.

[25] T. Filatova, D. C. Parker, and A. Van der Veen, "Agent-Based Urban Land Markets: Agent's Pricing Behavior, Land Prices and Urban Land Use Change", *Journal of Artificial Societies and Social Simulation*, vol. 12, no. 1, p. 3, 2009.

[26] E. Innocenti, C. Detotto, C. Idda, D. C. Parker, and D. Prunetti, "An Iterative Process to Construct an Interdisciplinary ABM Using MR POTATOHEAD: An Application to Housing Market Models in Touristic Areas", *Ecological Complexity*, vol. 44, p. 100 882, 2020.

[27] C. Alexander, *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[28] E. M. Buck and D. A. Yacktman, *Les Design Patterns de Cocoa*. Pearson Education France, 2010.

[29] T. M. H. Reenskaug, "User-Oriented Descriptions of Smalltalk Systems", *Byte*, vol. 6, no. 8, 1981.

[30] R. Smith, "Panel on Design Methodology", in *ACM SIGPLAN Notices*, ACM, vol. 23 (5), 1987, pp. 91–95.

[31] W. Cunningham and K. Beck, "Using Pattern Languages for Object-Oriented Programs", *Proceedings of OOPSLA*, vol. 87, 1987.

[32] R. Johnson, E. Gamma, R. Helm, and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", *Boston, Massachusetts: Addison-Wesley*, 1995.

[33] J. Hunt, *Scala Design Patterns—Patterns for Practical Reuse and Design*. Springer, 2016.

[34] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Wiley, 1996.

[35] S. Tisue and U. Wilensky, "NetLogo: A Simple Environment for Modeling Complexity", in *International Conference on Complex Systems*, Citeseer, vol. 21, 2004, pp. 16–21.

[36] E. M. Buck and D. A. Yacktman, *Cocoa Design Patterns für Mac und iPhone*. mitp Verlags GmbH & Co. KG, 2010.

[37] RF, "Plateforme Ouverte des Données Publiques Françaises", 2024, [Online]. Available: https://www.data.gouv.fr/ (visited on 05/27/2025).

[38] RF, "Impots.gouv.fr", 2024, [Online]. Available: https://www.impots.gouv.fr/ (visited on 05/27/2025).

Model-Based Development of Code Generators for Use in Model-Driven Development Processes

Hans-Werner Sehring
Department of Computer Science
NORDAKADEMIE gAG Hochschule der Wirtschaft
Elmshorn, Germany
e-mail: sehring@nordakademie.de

***Abstract*—Model-driven software development is gaining attention as a software engineering approach due to the various benefits it offers. Typical approaches start with the modeling of the application domain at hand and continue with the specification of the software to be developed. Results are documented by specific software engineering artifacts. Especially in model-driven approaches, these artifacts are formal or semi-formal models. Model transformations are applied to develop and refine artifacts. In a final step, code is generated from the models. Code can be source code written in specific programming languages, configuration files, and the like. Practice shows that model-based code generators have to bridge a rather large gap between the most refined software models and the compilable code that implements these models. This makes the development of code generators itself an expensive task. In this article, we discuss ways to break down the development of code generators into smaller steps. Our discussion is guided by both principles of compiler construction and by an application of model-driven development itself. Using a modeling language, we demonstrate how code generation can be organized to reduce development costs and increase reuse. In addition, program code becomes part of the model transformation sequence, allowing code changes to be automated and model elements to be referenced from code.**

Keywords-software development; software engineering; symbolic execution; top-down programming.

I. Introduction

Software construction requires methods and processes that guide development from an initial problem statement through all stages of the software lifecycle, culminating in the implementation, testing, rollout, operation, and maintenance of the software.

Model-Driven Software Engineering (*MDSE*) strives to support such development processes by making explicit

- the artifacts created at each stage and possibly intermediate results
- the decisions that lead to the development of each artifact.

Ideally, MDSE supports the entire software lifecycle from requirements engineering and domain concepts through software architecture, design, and programming to software operations.

Figure 1 outlines some typical artifacts of software engineering processes. While many of them can be handled in MDSE processes, executable code must be generated for a particular target platform, such as a *Programming Language* (*PL*), software libraries, a runtime environment, and a target infrastructure. Later stages that depend on the code, such as operations tasks, must also be considered in code generation. This prepares the code for activities such as maintenance, monitoring, etc.

The support provided by MDSE approaches has advantages in many application areas. Models of sufficient formality can be checked for completeness or correctness to a certain extent. Traceability between artifacts allows understanding of design decisions and model transformation steps during software maintenance. A final step of automated generation of executable code can save development costs during the implementation phase. Fully automated generation allows incremental development through model changes if the software is generated in an evolution-friendly manner.

Therefore, code generation from software models allows to take advantage of the potential benefits of modeling in MDSE. However, experience shows that generator development tends to be complex and costly. We see several reasons for this.

- The abstraction that models provide over programming language expressions requires code generators to deal with a higher level of abstraction than compilers for PLs.
- Implementation details that are not reasonably part of software models must be added in code during generation.
- Various non-functional requirements of professional software development must be satisfied by generated code in addition to the requirements explicitly reflected by the software models. A code generator must add code for these as cross-cutting concerns.

Furthermore, these aspects of code generation typically require the development of project-specific generators.

Code generators are similar to compilers for high-level PLs. From this point of view, a model-driven process can be divided into a *frontend* and a *backend* part. In this logical division, the frontend deals with the more abstract models of the application domain and software design in *model-to-model transformations* (*M2MTs*). These early phases are covered by MDSE approaches. The backend activities of code generation, optimization, and target platform considerations are often hidden in implementations of comprehensive *model-to-text transformations* (*M2TTs*).

In this article, we propose a structure for decomposing code generator development for easier development and a higher level of reuse.

This article extends the presentation of a conference paper on the topic [1].

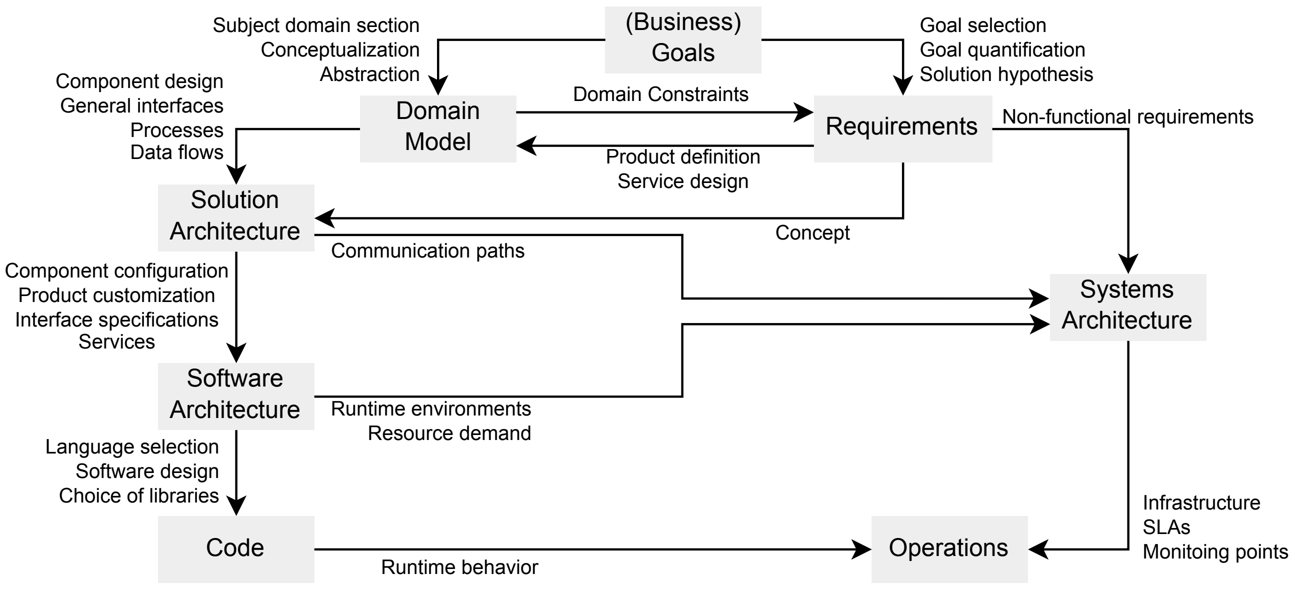


Figure 1. Typical software engineering artifacts.

The remainder of this article is organized as follows: In Section II, we review model-driven software engineering with a focus on the final step of code generation. A corresponding approach to code generation is outlined in Section III. We use a meta modeling language to present some models as experiments with the approach. This language is introduced in Section IV. In Section V, we illustrate the model-driven code generation approach with some sketches of code generation models. Some remarks on the derivation of code models from more abstract models are made in Section VI. The paper is concluded in Section VII.

II. Model-Driven Software Development

In this section, we revisit MDSE in general and code generation in particular in order to lay the foundation for the discussion of model-based code generation in the following sections.

A. Subject Domain Modeling

With few exceptions, the purpose of software is to solve some a real-world problem. For example, software is used to perform scientific calculations, to support a company's business processes, or to control hardware. As a result, a software project is typically embedded in a project with a broader scope. Therefore, the analysis and documentation of the task at hand begins with entities that lie outside the software domain, but within the broader *subject domain*, *application domain*, or simply *domain*.

The purpose of a domain model is twofold: it clarifies the terms and rules of the (real-world) application scenario and it defines a possible solution to the task at hand in terms of the application domain. As a model, it furthermore provides an abstraction by defining the section of the application domain that is considered by the software to be built.

Figure 1 presents typical stages of a project by naming the artifacts under consideration. Typically, a project begins by setting goals. Goals define success criteria for the overall project. Goals guide the choice of abstraction for the domain model, which is also defined in an early stage of a project. The same goes for the requirements (for the software), which are derived from both the goals and the domain conceptualization.

Subject domain modeling is beyond the scope of this article.

B. Software Modeling

Models of the early steps in the software lifecycle are formulated from the perspective of the application domain. From these, models of software from a technical perspective are derived. The solution architecture typically is the link between the domain perspective and the software perspective (see Figure 1).

Software description spans a series of models, starting from abstract ones to increasingly concrete ones. M2MTs are applied to derive models, even though mainly design decisions drive the modeling process.

When software models reach a sufficiently concrete level, code is finally emitted by M2TTs. The code generation and maintenance part is often not well represented in an MDSE process, though. On top of that, in most cases textual representations of code are decoupled from the models of earlier design phases.

Ideally, models of the software also support the operations and maintenance phase. In this case, they must be available at runtime [2].

The more abstract descriptions of software at the level of software architecture specifications are not in the focus of this article. However, it depends on the type of M2TTs or code generators at which point the transition from abstract software models to concrete source code takes place.

C. Cases of Code Generation

We see two application scenarios for software construction with MDSE:

1) approaches for systems of a given application class that share fixed functionality at some level of abstraction
2) approaches for application-specific functionality

A typical case of an application class with fixed functionality is the case of information systems, that typically provide only *Create, Read, Update, Delete* (*CRUD*) operations. Models of information systems, therefore, mainly represent domain entities and their relationships. Software generation is based on fixed patterns for code that provides CRUD functionality for the various entities.

Approaches that can be found in the class of generators that produce code with fixed functionality work with meta-programming [3][4], template-based approaches (see below), and combinations of these two [5]. Since generators for a specific target implementation can be built in a generic way, MDSE can be employed comparatively easy in this scenario.

In the general case of software containing custom business logic, software must be generated according to specified functionality. To automatically derive working software from specifications, MDSE approaches for application-specific business functionality must include formal models for precise definitions.

Means for deriving software from formal models are often built into editing tools for the respective formalisms. With respect to running software, formal models are typically used in one of two ways: Either code is generated from such models, or hand-written code is embedded in formal models at specific extension points.

For production-grade software systems, code generation is the only option in order to satisfy nonfunctional requirements. Depending on the modeling approach chosen, a model interpreter may not provide sufficient performance or scalability at runtime. The coexistence of the higher-level model and the lower-level PL code can increase maintenance complexity due to the different roles involved. Since changes to a model can affect the code [6], it is crucial to be able to trace of code design decisions.

A practical software system consists of different components, each of which is typically created by one generator each. Therefore, multiple code generators need to work in concert. To this end, different generator runs have to be orchestrated [5], and information exchange (for example, for identifiers used in different components) has to be managed [4].

D. Code Generation Techniques

Code is generated in a final step of an MDSE process, often based on M2TTs [7].

Special attention is paid to code generation, as this step can be well formalized in an MDSE process. There are several techniques for code generation, mainly generic code generators, meta-programming, and template-based techniques. Generative AI could be an alternative.

This way, there is reuse of software generators that translate formal specifications into code in a generic way. Typically, there is little or no way to direct the code generation for the case at hand [8]. Therefore, the generated code must be wrapped in order to be integrated into a production-grade software system, for example, to add error handling and additional code for monitoring.

a) Generic Code Generators: Custom functionality generally needs to be formulated in a Turing-complete formalism. Although the ability to verify such descriptions is limited, their expressiveness is required. Formal specifications of software functionality can be translated into working software by a code generator, that works like a compiler for a PL.

Code generators of modeling tools provide a well-tested and generally applicable translation facility. Specifications according to a given formalism are translated into a supported target environment. Examples include parser generators that generate code from grammars, software generators that take finite state machines as input [9], and those that use Petri Nets to execute code on firing transitions [10].

Generic code generators require significant development effort. But they can be developed centrally in a generic way. Therefore, there is a high degree of code reuse in the form of generators [11]. However, the models used as input are application-specific, and they must be more elaborate than the input for other forms of generators.

b) Meta-Programming: Programs that generate programs are an obvious means of generating software. Meta-programming is possible with PLs, that allow the definition of data structures that represent code and from which code can be emitted. Since many widely used languages do not include meta-programming facilities, this capability is added through software libraries or at the level of development environments.

Meta-programming provides maximum freedom in generating custom code. Consequently, results can be tailored to the application at hand, including specific business logic.

However, the development of such generators tends to be costly, depending on the degree of individuality of code [12]. This is due to the fact that meta-programs are harder to maintain and to debug due to their abstract nature. In addition, code reuse is very low for custom code.

c) Templates: Code with recurring structures can be formulated as templates with parameters for the variations of this uniform code. Code is generated by applying the templates with different parameter values.

A prominent example of a template-based approach is used for the *Model-Driven Architecture* (*MDA*). The *MOF Model to Text Transformation Language* [13] provides a means to define code templates based on (UML) models.

Templates are easy to write, depending on the degree of generics. They allow adaptation to the project at hand by making changes to templates. The degree of reuse of templates within a project can be high, depending of the structural similarities between parts of the code. Cross-project reuse can be expected to be quite low.

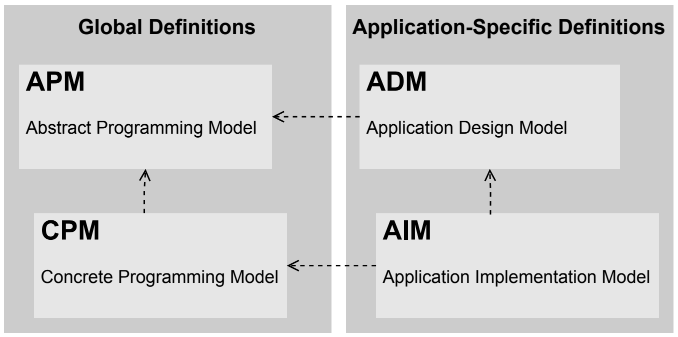

Global Definitions
Application-Specific Definitions
APM
Abstract Programming Model
ADM
Application Design Model
CPM
Concrete Programming Model
AIM
Application Implementation Model

Figure 2. Code models and their relationships.

d) Generative AI: The emerging generative AI approaches based on large language models provide another way to generate code from descriptions. Based on a library of examples, they allow the interactive generation of code from less formal descriptions, especially natural language expressions.

Generative AI can deal with complex requirements and rules. It has the advantage of being able to generate code in multiple PLs from (almost) the same descriptions.

There are indications that generative AI may be particularly well suited to producing code on a small scale, for example, individual modules [14]. Final quality assurance and assembly currently remains a manual task.

Instead of generating the actual software solution, generative AI can also be used to create code generators [7].

III. Model-Based Code Generation

In this paper, we discuss a way to construct code through a series of model refinement steps and final code generation. Thus, it follows the typical theme of M2MTs followed by an M2TT. However, our goal is to make the code generation step nearly trivial and fully automatic. To achieve this, we propose certain code models that bridge the gap between domain or solution models and executable code.

Our goal is to reduce the complexity of generators through abstraction and to reduce costs through reuse of *abstract code*.

Figure 2 gives an overview of the kinds of code models. Those in *Global Definitions* are provided centrally as a kind of modeling framework. Those in *Application-Specific Definitions* are models that are provided for each software project.

The four model boxes in the figure represent classes of models. There will be several concrete models for each of them.

We describe the models in the following subsections. Examples are given in the following main section.

The outline of the approach is as follows:

- Abstraction leads to a hierarchy of models.
- An *Abstract Program Model* (*APM*) provides a generic model of code.
- An *Application Design Model* (*ADM*) defines the functionality of a software system in terms of an APM.
- A *Concrete Program Model* (*CPM*) serves as a technology model; it maps an APM to a concrete implementation technology, such as a PL
- An *Application Implementation Model* (*AIM*) is used for code generation; it provides a project-specific association of the desired functionality and a technology model

With these models, some degree of reuse is achieved on the level of

1) programming models / building blocks of abstract programs
2) idioms and design patterns for refactoring and optimizing abstract programs
3) code generation from abstract representations of the constructs of a particular PL into code

A. Models of Programming

APMs serve as meta-models for abstract programs. Programming paradigms constitute a possible starting point for describing programming in general. Models of paradigms help to capture the essence of a class of PLs.

Properties of hybrid languages can be captured by combining models of programming paradigms. To this end, the modeling language used should allow models to be combined, and paradigm models must be set up to allow combinations.

There are differences between existing PLs that cannot be captured within one central model of programming. For example, object-oriented PLs have different ways of handling multiple inheritance. Therefore, there may be coexisting programming models, even for the same programming paradigm.

B. Assigning Functionality to Domain Models

In contrast to pure programming, program models in an MDSE process refer to more abstract models, especially those formulated from an application domain perspective. Program models result from M2MTs, or they refer to source models. Resulting model relationships are a basis for traceability [15].

Figure 3 illustrates a model relationship. A hypothetical domain model contains a *SalaryIncrease* concept with a *Raise* subconcept . This specification is to be implemented using an imperative programming language, so there is, for example, a *ConditionalStatement*. The resulting model of the code for the software is represented as an ADM with a procedure *CheckTargetSalary*.

Application design models are essentially attributed syntax trees. In a kind of “reverse programming”, we manually construct syntax trees and generate code from them. This is not the right level for manual development of software generators. But program models can be derived from domain models similar to template-based software generation. The example in Figure 3 can be viewed this way. The advantage of abstract models and model relations over code templates is the independence from concrete programming languages. This allows us to make an early connection between a domain and a code model while still having the option of choosing the implementation details, including the programming language or other implementation technologies to be used.

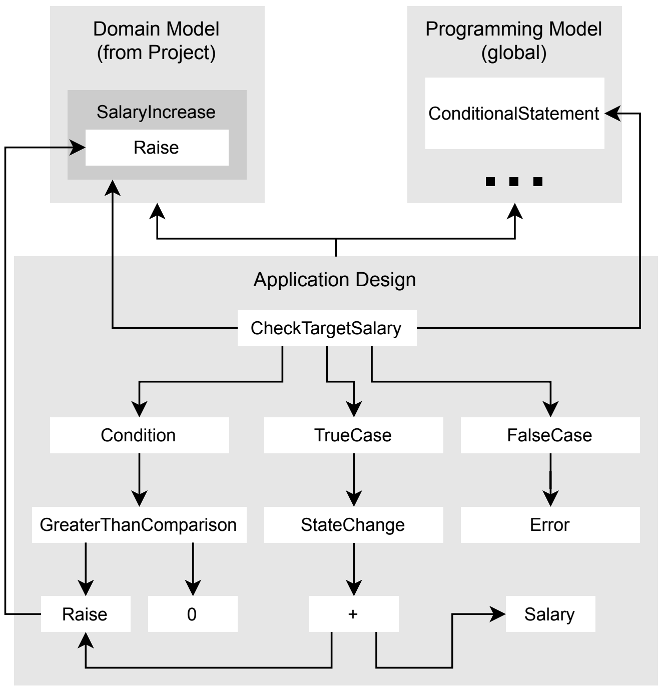


Figure 3. Typical software engineering artifacts.

C. Stepwise Refinement of Programming Models

Concrete models define which language constructs are available in a particular PL that is selected for implementation. For code generation, the abstract application code (given as an ADM) is combined with a CPM containing models of typical programming language constructs / idioms etc. in generalized form. M2MTs are applied to the combined model to transform it into an AIM that is suitable for code generation.

Model transformation consists of refining abstract program models with respect to a concrete PL or other implementation technology. There are several reasons why concrete models differ from abstract programming models. For example, there are different ways to implement abstract code in concrete PLs, similar PLs may have different best practices, they may have different constraints, and they may require different optimizations.

The transformation from an abstract to a concrete program need not be done in one step. For example, there is typically a hierarchy of abstractions, from abstract programs at the programming paradigm level, to classes of PLs and PL families, to concrete PLs, PL implementations or dialects, or even project-specific style guides.

D. Generating Code from Abstract Programs

An AIM is a model of a program that is suitable for code generation. This means that all parts of a model are assigned a concrete PL expression and thus a syntactic form.

With this model property, code can be generated by assembling pieces of code that each represent model entities.

IV. M^{3}L Overview

We use the *Minimalistic Meta Modeling Language* (*M^{3}L*) as our modeling notation. This language is briefly introduced in this section in order to discuss some model sketches in the remainder of the article.

A. Basic Concept Definitions

The M^{3}L is a (meta) modeling language that has been reported about. It is minimal in the sense that it is designed with a sparse syntax that is oriented towards metalogic and a simple semantics that basically breaks down to set theory.

A model in the M^{3}L consists of a series of definitions of *concepts*. A concept definition, in its simplest form, just names one concept:

```
A
```

Naming a concept leads to its evaluation (see Section IV-D below), if it exists, or to its creation otherwise. In the simplest form, just the name is introduced. Therefore, in either case, the concept A is defined after the above statement.

A concept can be defined as a *refinement* of another. For example, the following statement defines the concept A as a refinement of the concept B using the “is a” / “is an” clause.

```
A is a B
```

A is also called a *subconcept* of B, B the *base concept* or *generalization* of A. Multiple base concepts can be given at once:

```
A is a B, an E
```

Refinements *inherit* the definition from their base concept. This includes base concepts, content (see below), and rules (see below).

Using the “is the” clause instead defines a concept as the only specialization of its base concept.

```
C is the D
```

The concept C may have further base concepts, but D has no refinements other than C.

There may be multiple definitions of one concept. If definitions refer to the same concept name, their effect is cumulated.

```
A is a  B
...
A is an E
```

defines A as a subconcept of both B and E.

B. Contextual Concept Definitions

A concept C is defined in the *context* of a concept A by a definition of the form

```
A is a B {
  C is a D
}
```

C is part of the *content* of A. Each context defines a *scope*, and scopes are hierarchical. Concepts like A are defined in an unnamed top-level context. A concept A is *visible* in the context of a concept C if A belongs to the content of C or if there is a concept B such that A is visible in the context of B and C belongs to the content of B.

There can be multiple statements about a concept with a given name in different scopes. Contextual definitions are called *redefinitions*. All visible statements about a concept are cumulated. This allows concepts to be defined differently in different contexts. For example, the statements

```
A { C is a D }
C
```

define C as a specialization of D in the context of A, but without base concept in the topmost context.

A concept in a nested context is referenced as

```
C from A
```

There are well-defined names that refer to a part of a concept: "the concept" refers to the concept of a current definition, "the name" to its name, and "the content" refers to all content concepts of a concept. These are particularly used in refinements and redefinitions, such as

```
ListOfThings {
  Head is a Thing
  Tail is a ListOfThings
  AddToList {
    Elem is a Thing
  } is a ListOfThings {
    Elem is the Head
    the concept is the Tail
  }
}
```

This code allows adding a element to a singly linked list by using a current list as the tail of a new list (last line).

C. Semantic Rules

Semantic rules can be defined on concepts, denoted by a double turnstile ($\models$), in code written as "|=". A semantic rule references another concept, that is returned when a concept with a semantic rule is referenced. Like for any other reference, a non-existing concept is created on demand.

The scope of a semantic rule is specific: concepts referenced by the rule are resolved in the context of the concept to which the semantic rule belongs. If the rule leads to the creation of a new concept. then this concept is placed in the same context as the concept carrying the rule.

An example of a semantic rule is as follows:

```
A is a B {
  C is a D
} |= E { C }
```

In this example, E is defined by the semantic rule. Its content C is taken from the content of A. Because of the scoping rules, C must also be declared in the context of A. Otherwise, the rule is considered erroneous.

D. Concept Evaluation

Context, refinements, and semantic rules are employed for *concept evaluation*. When an existing concept is referenced, it is first evaluated, and the result of the evaluation is returned.

A concept evaluates to the result of its semantic rule, if defined, or else to its *narrowing* as defined below.

A concept's semantic rule is determined by the following steps:

1) If the concept's narrowing (see below) has a semantic rule, then this one is taken.
2) Else, if the narrowing inherits a semantic rule, then the rule of the closest ancestor is used.
3) Else, the set of *derived base concepts* (see below) is checked for semantic rules.
4) Else, there is no semantic rule.

A concept B is a narrowing of a concept A, if B is a transitive "is the" refinement of A.

A concept B is a derived subconcept of a concept A if

- the set of (transitive) base concepts of B is a superset of the set of (transitive) base concepts of A, and
- the content of A narrows down to content of B; this means that for every concept C in the content of A, there exists a concept D in the content of B such that D is C or one of its narrowings.

To evaluate a concept, syntactic rules and narrowing are applied repeatedly, until a concept evaluates to itself. Infinite evaluation loops are considered erroneous definitions.

An example of concept evaluation is provided by Figure 8 in Section V-B.

E. Syntactic Rules

Concepts can be marshaled/unmarshaled as text by *syntactic rules*, denoted by a turnstile ($\vdash$), in code represented by "|-". A syntactic rule names a sequence of concepts whose representations are concatenated, ended by a dot.

The representation of each concept is in turn defined by the syntactic rule of its evaluation. A concept without a syntactic rule is represented by its name.

Syntactic rules are used to represent a concept as a string as well as to create a concept from a string; they define both an output and a parser.

As an example, consider a definition

```
A is a B {
  C is a D
} |- the name says C .
```

When producing output for

```
John { hello is the C } is an A
```

it produces "John says hello", since *John* inherits the syntactic rule. Note that previously undefined concepts like *hello* and *says* are defined on spot, and that they are represented by their name.

```
TypedPL is a ProgrammingLanguage {
  Type

  Boolean is a Type
  True is a Boolean
  False is a Boolean

  Integer is a Type {
    Succ is a PositiveInteger {
      the concept is the Pred }
  }
  0 is an Integer
  PositiveInteger is an Integer {
    Pred is an Integer
  }
  1 is a PositiveInteger { 0 is the Pred }
}
```

Figure 4. Sample base model of typed programming languages.

V. Examples of Model Definitions for Code

We outline some models in order to illustrate the approach presented in Section III. We use the M^{3}L as introduced in the previous section as our modeling notation.

A. Example Programming Models

Sticking with the example of starting the modeling of programming with programming paradigms, there may be models that describe typical constructs of PLs of a particular paradigm. We give short outlines of PL base models for the most important programming paradigms that may provide concepts for APMs. Many details are omitted for the discussion in this article.

As the basis of programming models, assume base concept *ProgrammingLanguage*. Different aspects of programming are derived from that concept.

As a first specialization, Figure 4 sketches some basic concepts for typed programming languages. The concept *Boolean* represents a type for Boolean values; this type as the finite set of values *True* and *False*. The type *Integer* provides a definition of numbers based on the Peano axioms. Even though concrete programming languages offer a builtin type for integers that supports low-level operations provided by hardware, we want APMs to have provable properties independently of any concrete implementation. Please note that *Succ* is a method of an *Integer* object that answers the successor; it will be created if it does not exist. In contrast, *Pred* is an attribute that is set explicitly, for example, by *Succ*. Other types are omitted in this article, but may be defined accordingly.

1) Procedural Programming: Descriptions of some typical constructs of imperative PLs are shown in Figure 5. Typical control flow constructs, such as conditional statements and loops are given as M^{3}L concepts.

2) Functional Programming: Figure 6 outlines the basic definitions for functional PLs. Note that this model contains

```
ImperativeProgramming is a TypedPL {
  Variable {
    Name
    Type from TypedPL }

  Statement
  Sequence is a Statement {
    Statements is a Statement }
  CompoundStatement is a Sequence
  ParallelExecution is a Statement {
    ConcurrentStatements is a Statement }
  ConditionalStatement is a Statement {
    Condition     is a Boolean
    ThenStatement is a Statement
    ElseStatement is a Statement }
  Loop is a Statement {
    Body is a Statement }
  HeadControlledLoop is a Loop {
    Condition is a Boolean from TypedPL }
  CountingLoop is a Loop {
    Counter    is a  Variable {
      Integer is the Type }
    LowerBound is an Integer from TypedPL
    UpperBound is an Integer from TypedPL
    Step       is an Integer from TypedPL}
  VariableDeclaration is a Statement {
    Variable
    InitialValue is an Expression }

  Expression is a Statement
  Value is an Expression
  VariableReference is an Expression
  UnaryExpression is an Expression {
    Operand is an Expression }
  BinaryExpression is an Expression {
    Operand1 is an Expression
    Operand2 is an Expression }

  ...
}

ProceduralProgramming
  is an ImperativeProgramming
{
  Procedure {
    FormalParameter is a Variable
    Body            is a Statement }

  ProcedureCall is a Statement {
    Procedure
    ParameterBinding {
      FormalParameter from Procedure
      ActualParameter is an Expression } }

  ...
}
```

Figure 5. Sample base model of procedural programming.

```
FunctionalProgramming {
  Expression
  Value is an Expression
  ConditionalExpression is an Expression {
    Condition  is an Expression
    TrueValue  is an Expression
    FalseValue is an Expression
  }
  Function is a Value {
    FormalParameter
    FunTerm
  }
  FunCall is an Expression {
    Function
    ParameterBinding {
      FormalParameter from Function
      ActualParameter is an Expression
    }
  }
}
```

Figure 6. Sample base model of functional programming.

definitions that may not apply to all functional PLs, so other APMs exist.

The simple syntax of functional PLs makes the definition of this programming paradigm quite short. Note, however, that many properties of functional PLs are covered by the model. For example, partial function application is expressible since a function call (*FunCall*) is an *Expression*, and a *Function* is a *Value* which is also an *Expression*. So function calls can deliver functions. Likewise, higher-order functions are covered by the model.

The model in Figure 6 omits typical libraries of predefined functions, for example, the various kinds of recursive higher-order functions. These differ slightly between concrete PLs, though, so there will be variations.

3) Object-Oriented Programming: Only some base definitions for class hierarchies at the instance and class levels are sketched in Figure 7. The complete model is much more elaborate, and there are even more variants of PLs than in the other paradigms.

In particular, typical object-oriented PLs consist of two "sub-languages". In a declarative part, objects or classes are defined, method signatures (message formats) are declared, etc. Executable code is mainly found in method bodies, which are implemented depending on the kind of object-oriented PL used.

Statements as used in imperative programming are one way of implementing methods. In Figure 7, this is sketeched by the *OOP-imperative* concept. In particular, compound statements are typically used. In contemporary PLs, specific additions to the set of statements cater for object-oriented structures.

```
ObjectOrientedProgramming {
  MetaClass is an Object { Method }
  Method {
    Parameter is an Object
    Body
  }
  Classifier is a MetaClass
  Interface     is a Classifier
  AbstractClass is a Classifier
  ConcreteClass is a Classifier
  ObjectClass is a ConcreteClass
  Object is an ObjectClass
}
OOP-imperative
  is an ObjectOrientedProgramming,
     an ImperativeProgramming
{
  Method {
    LocalVariables is the Parameter,
                          a   Variable
    Body is a Statement
  }
}
OOP-functional
  is an ObjectOrientedProgramming,
     a  FunctionalProgramming
{
  Method {
    Body is a Function {
      FormalParameter is the Parameter
    }
  }
}
```

Figure 7. Sample base model of object-oriented programming.

A second option of implementing methods is in a functional way by assigning a function to a message. This is outlined by the concept OOP-functional in Figure 7. The parameter(s) of the function (*FormalParameter from Function*) that is used as a method body are passed from the parameters of the method (*Parameter from Method*).

B. Programming Language Semantics

For model checking or for model execution, language constructs as outlined in the previous subsection must be given semantics. The semantics of specific PLs is abstracted so that different generalized APMs can be defined to capture the various interpretations of PL constructs.

As an example, the behavior of the *ConditionalStatement* can be defined as shown in Figure 8. A concrete conditional statement as part of a program may look like

```
IfTrueStmt is a ConditionalStatement {
  True is the Condition
} |= ThenStatement

IfFalseStmt is a ConditionalStatement {
  False is the Condition
} |= ElseStatement
```

Figure 8. Semantics of conditional statements.

```
MyConditional is a ConditionalStatement {
  SomePredicate is the Condition
  Statement1    is the ThenStatement
  Statement2    is the ElseStatement
}
```

A concrete conditional statement like *MyConditional* must not have a semantic rule defined.

When evaluated, such a conditional statement will match (become a derived subconcept) of either *IfTrueStmt* or *IfFalseStmt*, depending on what *SomePredicate* evaluates to: *MyConditional*, like any refinement of *ConditionalStatement*, has a superset of the base concepts of both *IfTrueStmt* and *IfFalseStmt* since it is an explicit subconcept. The content of which one of them matches is determined by the evaluation of *SomePredicate* that should yield either *True* or *False*. Then, exactly one of *IfTrueStmt* or *IfFalseStmt* will be a derived base concept of *MyConditional* which will inherit the semantic rule that leads to the correct interpretation of the conditional statement.

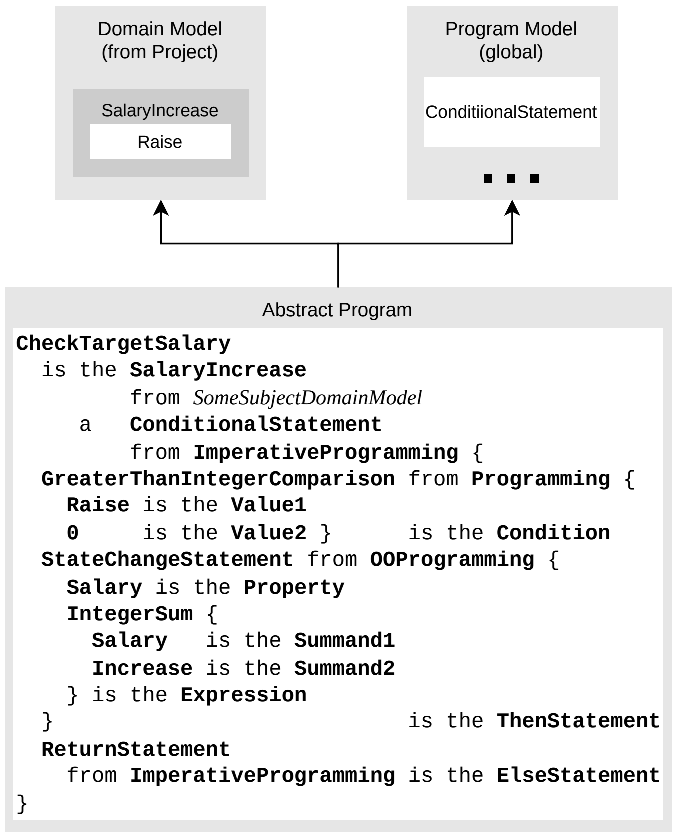
The semantic rule is inherited from the derived base concept, making the statement evaluate to either the “then branch” or the “else branch”.

This way of attaching semantics is typical for M^{3}L models; other modeling languages may have different ways of attaching semantics. We will not go into this in detail. However, it is an important part of the PL base models.

C. Abstract Programs

Based on the programming concepts outlined in the preceding subsections, abstract programs can be formulated using “instances” of these concepts. This means that refinements of the concepts of an APM form an ADM. Such a program is abstract in the sense that it is not written in a concrete PL. It is more like an attributed abstract syntax tree.

Figure 9 shows an example of an abstract code fragment for a sample ADM. The code is based on the model shown in Figure 3. An imperative object-oriented programming style is chosen. The entire code fragment represents a conditional statement that checks for a positive raise. The *ConditionalStatement* is outlined in Figures 10 and 8. The *GreaterThanIntegerComparison* may be a binary predicate that compares integers. It is used as the condition. The two branches of the conditional statement, *ThenStatement* and *ElseStatement*, are set to accordingly. On a positive raise, there is a state change, assuming that there is some employee object (omitted from the example; *Raise* and *Salary* may be declared outside the code fragment). On an invalid raise, the code is simply exited.



```
CheckTargetSalary
  is the SalaryIncrease
        from SomeSubjectDomainModel
     a  ConditionalStatement
        from ImperativeProgramming {
  GreaterThanIntegerComparison from Programming {
    Raise is the Value1
    0     is the Value2 }      is the Condition
  StateChangeStatement from OOProgramming {
    Salary is the Property
    IntegerSum {
      Salary   is the Summand1
      Increase is the Summand2
    } is the Expression
  }                            is the ThenStatement
  ReturnStatement
    from ImperativeProgramming is the ElseStatement
}
```

Figure 9. Typical software engineering artifacts.

This allows complete code bases to be formulated in an abstract way. Starting from a domain model, programming constructs can be introduced step by step to from an ADM. Using a CPM, an ADM can be transformed into an AIM.

D. Abstract Program Transformations

In our experimental setup with the M^{3}L, model transformations can be expressed by relating concepts to each other. In other modeling languages, the respective model transformation or model evolution facilities are used [16][17][18]. In the M^{3}L, M2MTs can be implemented by concept refinement, concept redefinitions, and semantic rules. M2TTs are expressed by syntactic rules in combination with concept evaluation.

Model transformations are, in the discussion of this article, transformations of abstract programs. The advantages of expressing code in abstract forms are manifold.

1) Stepwise Concretization: Many aspects of an implementation must be considered at once when there is just one code generation step. Code generation does not only have to produce code with the correct functionality. It must also respect nonfunctional requirements. On top of that, there are crosscutting concerns like error handling when producing executable code.

2) Higher Degree of Reuse: Abstract code has a possibly higher chance of being reusable. In particular, concepts may serve as prototypes that are redefined to concrete uses. On

```
Java is a ProgrammingLanguage {
  ConditionalStatement
  |- if ( Condition )
     ThenStatement
     ElseStatement .
}
Python is a ProgrammingLanguage {
  ConditionalStatement
  |- if Condition :
     " " ThenStatement
     else:
     " " ElseStatement .
}
```

Figure 10. A sample abstract program.

top of that, high-level designs, such as design patterns can be codified an applied where needed [19].

3) Roundtrip Engineering: When maintaining software, traceability from models to code is improved over M2TTs that generate code directly from more abstract models. If a need for change can be localized in the working software, it is potentially easier to trace it back to models.

4) Optimization: Code optimization is more effective at higher levels of abstraction. For example, algorithmic changes will usually have a greater impact than local code optimizations. With the ability to optimize code at each model layer, any generated code will benefit from optimizations without having to rely on PL-specific tools, such as compilers.

5) Cross-Platform Development: Finally, abstract programs allow target code to be generated in different PLs. This facilitates the development of distributed applications in heterogeneous environments, and the generation of code for different platforms from the same (abstract) code base.

E. Code Generation

The final M2TTs to produce source code are performed on models that combine an ADM with the abstract program for the problem at hand and a CPM that declares concrete PL constructs.

The CPM comes with predefined translation tables that are used to generate code. Such translation tables can be formulated by syntactic rules in the example of the M^{3}L.

For example, rules for language-dependent code generation for two different languages can be such as sketeched in Figure 10.

By separating APMs and CPMs, it is possible to generate different code from the same abstract program. In the M^{3}L, concepts can easily be redefined with different syntactic rules in the context of a PL. When generating code in such a PL context, the rules of all language constructs for that PL are used. Variations for language dialects can be handled in subcontexts where some rules are redefined.

```
Company {
  Person { Name Age }
}
CompanyImpl is a Company {
  PersonRecord is a Person,
                  a Record from TypedPL {
    Name is a  String  from TypedPL
    Age  is an Integer from TypedPL
  }
}
```

Figure 11. Sample of a first application design model for a domain model.

VI. Higher Programming Abstractions

In this article, we focus on models of programming language code. In an overarching modeling process, there are M2MTs that lead from domain models to software models and ones that lead from general software models to code models. Such M2MTs mark the start of a new development phase.

Depending on the models chosen, the gap between models of two subsequent phases may be rather large. Reasons are the change of concepts and the preferred structures that combine them. In this section, we briefly discuss two kinds of models that may bridge the gap more gently in the following two subsections.

Intermediate models that contain concepts from both the application domain as well as the software domain allow introducing a subset of the required technical concepts.

There are certain abstractions of code that provide a starting point for such intermediate models [3]. In particular, software design approaches for manual software development can be used as blueprints for the creation of initial code models.

A. Domain Models as Initial Application Design Models

An advantage of a consistent (meta) modeling approach like the one provided by the M^{3}L is the seamless transformation of domain-centric models towards code-centric models. This may be achieved by hybrid models, such as high-level code models that are based on domain models as data or object models and that constitute a first ADM. ADMs can be formulated in the M^{3}L as refinements of APMs. Implementation aspects are added stepwise to transform such a hybrid model into an adequate ADM.

Hybrid models provide more abstract software constructs in the sense of *Domain-Driven Development* or *Domain-Specific Languages* (*DSLs*) in which implementations relate to domain concepts directly [20]. They also provide a domain model that includes a connecting points to an implementation that may alternatively be added in a generic way. For information system, for example, typically CRUD operations are added for a domain-specific data model (see Section II-C). In such an approach, a first model provides consistent technical types for the attributes of (domain) entities and operations defined on them.

```
Company {
  OrgUnit is a Record {...}
  BusinessUnit is an OrgUnit {
    Departments is an Department
  }
  Department   is an OrgUnit {
    Teams is a Team
  }
  Team         is an OrgUnit {
    Members is a Person
  }
}

CompanyImpl is a Company

OrgUnitImplementation
  is a CompositePatternApplication
{
  CompanyImpl is the TargetModel
  OrgEntity   is the ComponentClass
  OrgUnit     is the CompositeClass
  Person      is the LeafClass
  Members is the AggregationRelationship
}
```

Figure 12. A sample domain model and design pattern application.

For an example, please consider a concept *Person* that is part of a domain model *Company*. Data about *Person*s shall be managed by an aggregated type that consist of a person's *Name* and *Age*. A first ADM for the example is outlined in Figure 11 as *CompanyImpl*, where persons are modeled as *Record*s composed of a *String* and an *Integer*. This type information is assigned on the basis of abstract type information as sketched in Figure 4.

Depending on the choice for a technology, there will be a direct mapping of the data types, and (CRUD) functionality can be added based in that mapping (see Section II-C).

B. Design Patterns

Design patterns provide proven solutions for specific tasks [21]. Additionally, they provide design standards that are well known amongst developers. This additional use of patterns may be codified in model transformations. There are patterns that are helpful in elaborating code models, and ones that help bridging the gap from more abstract to more concrete models.

As an example, assume organizational units in a company defined by a domain concepts in Figure 12. This sample company has three organizational layers, *BusinessUnit*, *Department*, and *Team*. A typical implementation will not reflect these domain concpts directly. Instead, there will probably be one implementation concept *OrgUnit* and the *Composite Pattern* [21] applied to it.

Figure 13 outlines a formalization of the pattern. A pattern defined this way can be applied by "instantiating" the pattern

```
PatternDefinitions {

  CompositePatternApplication {
    TargetModel
    ComponentClass
    CompositeClass
    LeafClass
    AggregationRelationship
  } |= TargetModel {
    ComponentClass is an AbstractClass
           from ObjectOrientedProgramming
    LeafClass      is a  Classifier
           from ObjectOrientedProgramming,
                      a  ComponentClass
    CompositeClass is a  Classifier
           from ObjectOrientedProgramming,
                      a  ComponentClass {
      AggregationRelationship
        is a ComponentClass
    }
  }

}
```

Figure 13. Layout of a pattern definition.

```
CompanyImpl {

  OrgEntity is an AbstractClass
           from ObjectOrientedProgramming

  Person    is a  ConcreteClass
           from ObjectOrientedProgramming,
               an OrgEntity

  OrgUnit   is a  Classifier
           from ObjectOrientedProgramming,
               an OrgEntity
  {
    Members is an OrgEntity
  }

}
```

Figure 14. Sample implementation resulting an application of the composite pattern.

concept, *CompositePatternApplication* in the example of the company organization as shown in Figure 12.

With the placeholders *TargetModel*, *ComponentClass*, *CompositeClass*, and *LeafClass* set by "is the", they each evaluate to the actual concepts given in the pattern application. Therefore, they will generate the model structure shown in Figure 14. Please note that the resulting implementation model *CompanyImpl* amends the definition in Figure 12, which leads to the concrete *OrgUnits* being available in the implementation, but now combinable via the *Members* relationship.

This resulting software model is not as accurate as the domain model, but better reflects the generalized way in which the model will be implemented.

VII. Conclusion and Future Work

Model-Driven Software Engineering is receiving a lot of attention for the benefits it brings to software engineering processes. While model-to-model and model-to-text transformations are being researched, in practice the final step of code generation from models is too costly to be applied in many application scenarios.

In this article, we propose an approach for defining code generators within the MDSE toolchain based on models. Generic models lay a foundation for all code generators. To this end, we studied models of typical programming paradigms with certain language-specific properties. Code generators consist of executable models that are derived from the base models. They should, therefore, be formulated in the same modeling language. If the models that define a code generator are also formulated in the same modeling framework as the models for earlier stages of the software engineering process, then models of the application domain and models of the software can be closely related. We use the M^{3}L as a consistent modeling framework.

The proposed approach allows us to achieve the goals of reduced development costs for code generators and of increased reuse, of both the base models and of parts of application-specific models. The use of multiple levels of abstraction makes each development step easier and less costly. Since the most abstract models can be applied in a generic way, they can be reused in different applications.

Future work includes experiments with real-world code models before pursuing new research directions. Since many important PLs are hybrid in nature, remaining issues with combined APMs need to be addressed, such as the mismatch between imperative and declarative PLs.

Acknowledgments

The publication of this work was made possible through the support of NORDAKADEMIE gAG.

References

[1] H.-W. Sehring, "Building model-based code generators for lower development costs and higher reuse", in *Proceedings Nineteenth International Conference on Software Engineering Advances*, ThinkMind, 2024, pp. 26–31.

[2] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap", in *Future of Software Engineering (FOSE '07)*, 2007, pp. 37–54.

[3] K. Czarnecki and S. Helsen, "Classification of model transformation approaches", in *Proceedings OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, vol. 45, 2003, pp. 1–17.

[4] H.-W. Sehring, S. Bossung, and J. W. Schmidt, "Content is capricious: A case for dynamic system generation", in *Proceedings Advances in Databases and Information Systems*, Springer, 2006, pp. 430–445.

[5] H. Mannaert, K. D. Cock, and J. Faes, "Exploring the creation and added value of manufacturing control systems for software factories", in *Proceedings Eighteenth International Conference on Software Engineering Advances*, ThinkMind, 2023, pp. 14–19.

[6] J. D. Rocco, D. D. Ruscio, L. Iovino, and A. Pierantonio, "Dealing with the coupled evolution of metamodels and model-to-text transformations", in *Proceedings of the Workshop on Models and Evolution*, ser. CEUR Workshop Proceedings, vol. 1331, CEUR-WS.org, 2014, pp. 22–31.

[7] K. Lano and Q. Xue, "Code generation by example using symbolic machine learning", *SN Computer Science*, vol. 4, 2023.

[8] T. Mucci, *What is a code generator?*, [Online] Available from: https://www.ibm.com/think/topics/code-generator. 2024.6.28. Think 2024, 2024.

[9] T. E. Shulga, E. A. Ivanov, M. D. Slastihina, and N. S. Vagarina, "Developing a software system for automata-based code generation", *Programming and Computer Software*, vol. 42, pp. 167–173, 2016.

[10] K. Radek and J. Vladimír, "Incorporating Petri nets into DEVS formalism for precise system modeling", in *Proceeding Fourteenth International Conference on Software Engineering Advances*, ThinkMind, 2019, pp. 184–189.

[11] K. Czarnecki, "Overview of generative software development", in *Unconventional Programming Paradigms*, Springer Berlin Heidelberg, 2005, pp. 326–341.

[12] S. Trujillo, M. Azanza, and O. Diaz, "Generative metaprogramming", in *Proceedings of the 6th International Conference on Generative Programming and Component Engineering, GPCE '07*, Association for Computing Machinery, 2007, pp. 105–114.

[13] Object Management Group, *MOF model to text transformation language, v1.0*, [Online] Available from: https://www.omg.org/spec/MOFM2T/1.0/PDF. 2024.7.4. OMG Document Number formal/2008-01-16, 2008.

[14] M. Harter, "LLM Assisted No-code HMI Development for Safety-Critical Systems", ThinkMind, 2023, pp. 8–18.

[15] S. Hajiaghapour and N. Schlueter, "Evaluation of different systems engineering approaches as solutions to cross-lifecycle traceability problems in product development: A survey", in *Proceedings International Conference of Modern Systems Engineering Solutions*, ThinkMind, 2023, pp. 7–16.

[16] A. Agrawal, "Metamodel based model transformation language", in *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Association for Computing Machinery, 2003, pp. 386–387.

[17] D. Song, K. He, P. Liang, and W. Liu, "A formal language for model transformation specification", in *Proceedings of the Seventh International Conference on Enterprise Information Systems - Volume 3: ICEIS*, INSTICC, SciTePress, 2005, pp. 429–433.

[18] A. P. Fontes Magalhaes, A. M. Santos Andrade, and R. S. Pitangueira Maciel, "Model driven transformation development (mdtd): An approach for developing model to model transformation", *Information and Software Technology*, vol. 114, pp. 55–76, 2019.

[19] A. Kusel et al., "Reuse in model-to-model transformation languages: Are we there yet?", *Software & Systems Modeling*, vol. 14, pp. 537–572, 2 2015.

[20] D. Thomas and B. M. Barry, "Model driven development: The case for domain oriented programming", in *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Association for Computing Machinery, 2003, pp. 2–7.

[21] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 1994.

Features, Practical Applications, and Validation of COSMOS Simulator: A Construction-Process Simulation Tool

Jirawat Damrianant
Department of Civil Engineering,
Faculty of Engineering, Thammasat School of Engineering,
Thammasat University,
Pathum Thani, Thailand
E-mail: djirawat@engr.tu.ac.th

Sakkaphant Meklersuewong
Department of Civil Engineering,
Faculty of Engineering, Thammasat School of Engineering,
Thammasat University,
Pathum Thani, Thailand
E-mail: sakkaphant@gmail.com

Abstract—**Computer-based simulation software is essential for efficiently simulating complex processes. COSMOS Simulator is a program developed specifically for simulating models created using COSMOS methodology, a modified Petri Net designed for simulating construction-based operations. However, unlike some existing Petri Net-based simulators, which may require a deep understanding of Petri Net theory, COSMOS is designed to be intuitive and accessible to construction professionals. Although previous studies have used the COSMOS Simulator to simulate various construction processes and documented its accuracy, no published work directly describes the simulator itself. This article aims to provide a detailed description and illustration of the COSMOS Simulator's features, especially its ability to model and simulate specific construction behaviours. In addition, this article offers further summaries and discussions of previous studies on the software's practical applications and validation. The paper provides a resource for researchers and practitioners interested in leveraging COSMOS for their construction modelling and simulation needs.**

Keywords-COSMOS; Construction Process Simulation; Domain-Specific Modelling; Petri Nets; Practical Applications.

I. INTRODUCTION

Process modelling and simulation are valuable approaches for construction engineering. However, a suitable software tool is necessary to simulate complex construction operations. The need for construction simulation software has been driven by the increasing complexity of construction projects and the need for effective planning and resource management tools. To address this need, the authors previously presented a detailed description and comprehensive illustration of the Construction Oriented Simulation MOdelling System (COSMOS) Simulator's features at the Sixteenth International Conference on Advances in System Modelling and Simulation (SIMUL 2024) in Venice, Italy, and received the Best Paper Award [1]. The present paper significantly expands on that earlier work, further elaborating the simulator's distinctive capabilities, practical applications, and validations.

Before discussing COSMOS in more detail, it is helpful to briefly review the historical context of simulation software development in construction. This overview will illustrate the evolution of such software and highlight key challenges encountered in the past, providing necessary background that explains the rationale for COSMOS's development. A comprehensive review of this historical context was previously provided in [2] and is briefly summarised as follows.

Early systems like the Micro-Computerised CYCLic Operation NEtwork (MicroCYCLONE) and the Dynamic Interface for Simulation of Construction Operations (DISCO) laid the groundwork for the field. Still, their adoption was often limited by the specialised knowledge required to use them. The emergence of object-oriented programming and discrete-event simulation paradigms led to the development of more user-friendly and versatile tools like the Construction Operation Simulation Tool (COST) and the Construction Object Oriented Process Simulation (COOPS) system. As construction projects grew in complexity in the 2000s, simulation tools like Simphony and STROBOSCOPE were developed to offer customisable and user-friendly platforms for modelling specific construction operations. However, the inherent complexity of construction processes, with their intrinsic uncertainties and dynamic interactions, continued to pose challenges for simulation modelling. In addition, many previously developed tools remained difficult to use, requiring substantial technical knowledge of simulation methodologies for construction practitioners.

These challenges led to the development of COSMOS, a simulation methodology that extends traditional Petri net frameworks with construction-specific modelling elements. As detailed in this paper, the COSMOS Simulator significantly advances construction process modelling and simulation. The software can simulate models created using the COSMOS methodology [3], a modified Petri Net designed to facilitate simulation modelling of construction-based operations. The methodology introduces new nodes, arcs, and attributes to capture complex construction behaviours, improving the ease and realism of modelling for simulation and analysis. Reference [3] details how these extended elements interact to represent various construction scenarios, showcasing their flexibility in handling the complexities of construction. However, unlike some Petri Net-based simulators that may demand a deep understanding of Petri Net theory, COSMOS is crafted to be easily accessible for construction professionals.

This article addresses a gap in the existing literature by providing a direct and detailed description of the COSMOS Simulator's features and capabilities. While previous studies have utilised the simulator for various construction simulations (some in Thai) [4]-[12], and the software's accuracy has also been confirmed and reported in several articles (some in Thai) [6][8][9][11], a dedicated publication outlining its functionalities was lacking. This paper fills that void. The article offers a detailed description and illustration of the distinctive features of the COSMOS Simulator, notably its capability to model behaviours not typically accessible in other Petri Net simulators, such as [13]-[17]. These features include elements like Header, Follower, Buffer, Pipe, End Arc, and DPA, which can manage continuous processes and dynamically progressive activities commonly encountered in specific construction processes. In addition, this article offers further summaries and discussions of previous studies on the software's practical applications and validation. The COSMOS Simulator's user interface and key components will be described in Section II. Section III presents practical implementations of the simulator along with validation results that attest to its accuracy and applicability. A discussion, conclusion, and suggestions for future work will be provided at the end of the article in Sections IV and V.

II. Descriptions of User Interface and Key Components

This section will review COSMOS's user interface and explain the essential components of the COSMOS Simulator. Figure 1 displays the homepage of the COSMOS Simulator's user interface, which can be accessed by selecting "Model" in the "view mode selector" panel. It should be noted that the Model mode is pre-selected by default. The system interface comprises several key components: the Menu Bar, Simulation-Run Controller Panel, Simulation Control Bar Properties Palette, Modelling Element Panel, Model Drawing Area, Status Bar, and View Mode Selector. The following subsections will comprehensively describe each of these significant components of the COSMOS Simulator.

A. Menu Bar

The menu bar in Figure 1 is divided into three tabs: Files, Settings, and Help. Each tab contains commands for manipulating files and software settings, such as creating a new file, opening an existing file, saving files, and changing font and grid settings.

B. Simulation-Run Controller Panel

To operate the simulation, users interact with the buttons on the "simulation-run controller panel". This panel contains several buttons as follows;

"Continuous Run" initiates a continuous simulation with animation as transitions fire and tokens move.

"Flash Run" simulates without displaying any animation, only providing the simulation's results unless the user specifies that animation should be shown.

"Pause" temporarily halts the simulation.

"Reset" brings the simulation back to its initial state.

"Previous Step" steps the simulation backwards by one step.

"Previous Event" steps the simulation backwards by one event.

"Next Event" steps the simulation forward by one event.

"Next Step" steps the simulation forward by one step.

See Figure 2 for the locations of these buttons in the user interface.

It is important to note that running the simulation by a step or by an event differs in terms of how the animation displays tokens residing in the places between adjacent transitions. When simulating by an event, the animation does not show tokens temporarily residing in the places, whereas simulating by a step does display these tokens.

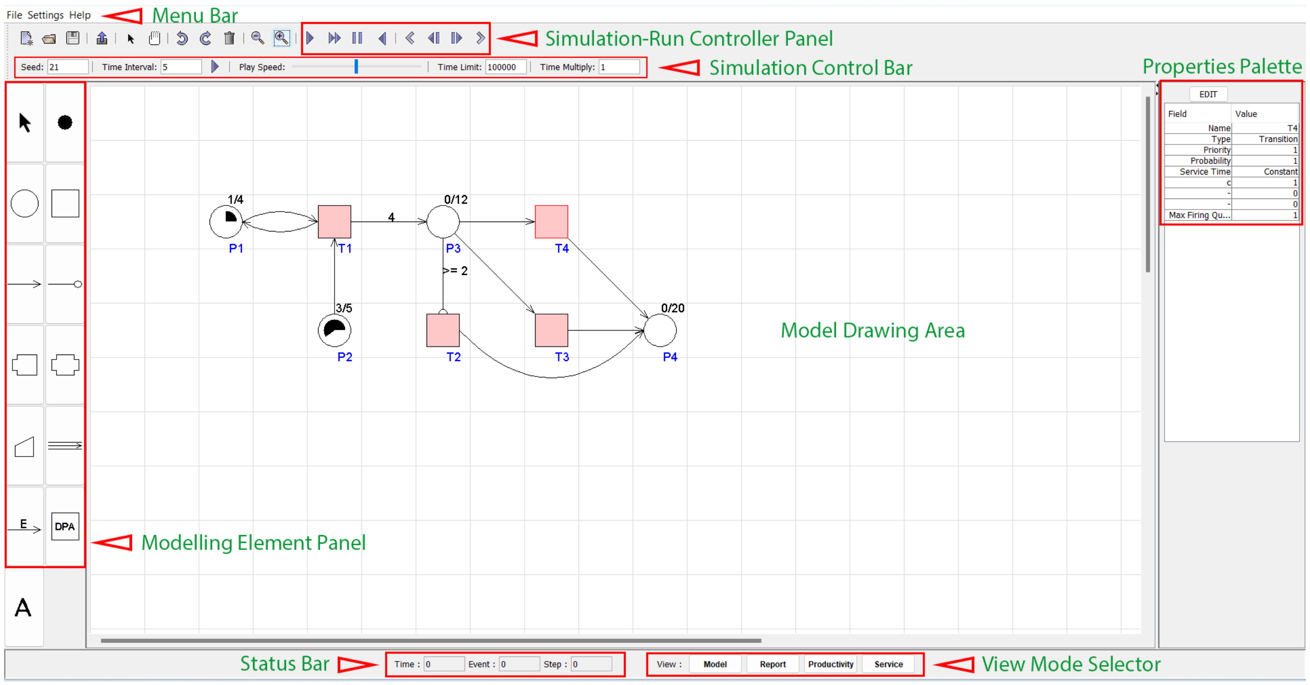
C. Simulation Control Bar

Before running a simulation, users can define a seed number in the "Seed" field of the "simulation control bar" (see Figure 1). The specified seed number is the initiator for generating a random number stream using the Linear Congruential Method. This stream is subsequently utilised to generate random samplings, including the firing duration, referred to as 'Service Time' within the COSMOS Simulator. Service time is sometimes stochastic; in such cases, the generated random numbers are used to determine the service time of the transitions each time they fire. These stochastic durations are governed by Probability Density Functions (PDF) specified by the users (see Figure 3). Additionally, the COSMOS Simulator utilises the stream to determine events for transition firings, whether they will fire or not. The determination is based on the probability ratios associated with transitions set by the users. These transition probabilities can be employed to resolve conflicts among transitions, should they arise.

The control bar offers additional functionalities. The Time Interval field allows users to specify the display frequency of the simulation run. For example, suppose the COSMOS simulation begins at time = 0, and the Time Interval is set to 5 minutes. In that case, the Simulator will visualise the run at 5, 10, 15, 20 minutes, and so on, showcasing the transition's firing and token movement animations at those time intervals. The simulation's speed can be adjusted using the Play Speed slider. Additionally, the Time Limit field allows users to define a specific time at which the simulation will be forced to terminate, even if its natural stopping conditions are not met.

D. Modelling Element Panel

The COSMOS modelling elements are located in the "modelling element panel", as shown in Figure 1. The panel contains various buttons representing different modelling element types, except for the top-left button, which serves as

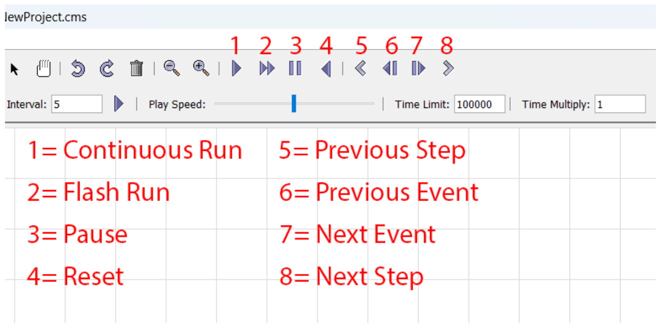


Seed: 21 | Time Interval: 5 | Play Speed: | Time Limit: 100000 | Time Multiply: 1

Field	Value
Name	T4
Type	Transition
Priority	1
Probability	1
Service Time	Constant
c	1
-	0
-	0
Max Firing Qu...	1

Time : 0 Event : 0 Step : 0

Figure 1. Homepage of the COSMOS Simulator's user interface.



Interval: 5 | Play Speed: | Time Limit: 100000 | Time Multiply: 1

1= Continuous Run	5= Previous Step
2= Flash Run	6= Previous Event
3= Pause	7= Next Event
4= Reset	8= Next Step

Figure 2. Simulation-Run Controller Panel

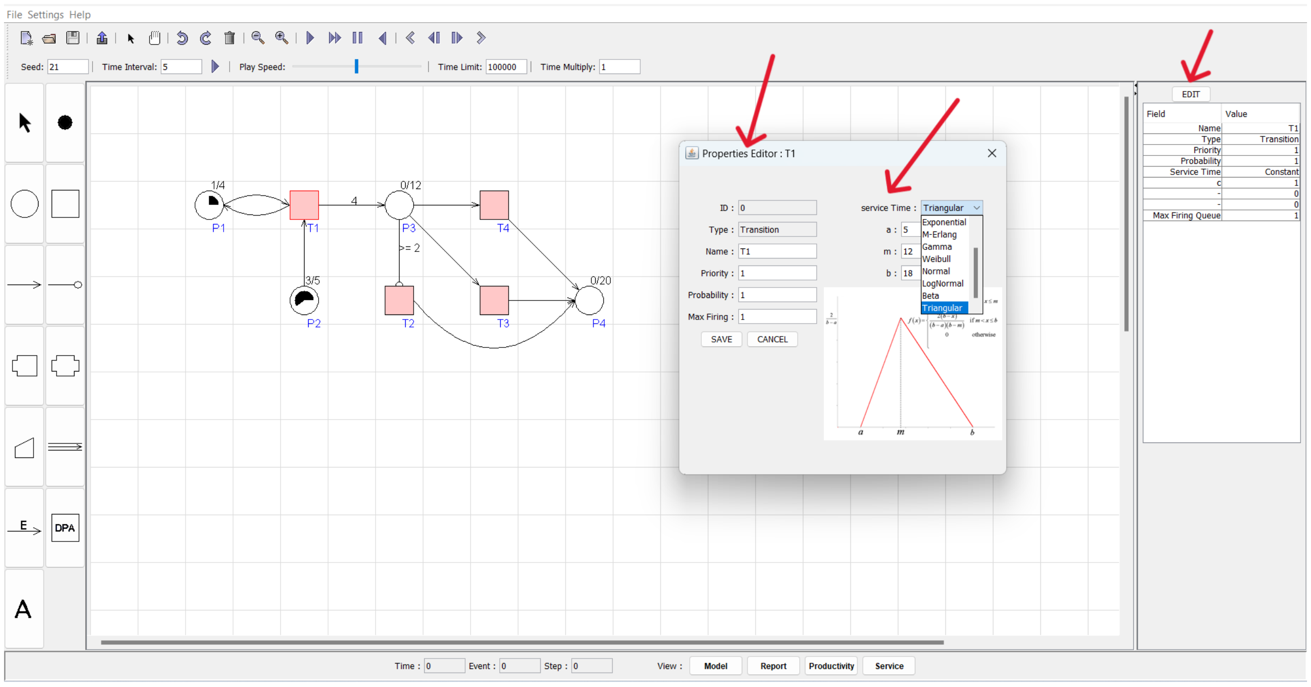
the selection mode. Clicking on any of these buttons allows users to enter the mode for placing the selected element type on the "model drawing area." The first four elements in the panel, located next to the selection mode, are the common Petri Nets elements: Token, Place, Transition, and Arc.

1) Place: A place element has two primary attributes: capacity and marking. Capacity refers to the maximum number of tokens that can be stored in a place at any given time, whereas marking indicates the current number of tokens present in the place. For instance, consider a Petri Net shown in Figure 1, where place P1 has a capacity of 4 tokens and currently contains one token. The current marking and capacity of the place are denoted by the numbers on the top-right corner as "1/4". A black area resembling a pie chart is used to visually represent the ratio between the marking and the capacity of the place.

2) Transition: Transitions in the COSMOS Simulator have several primary attributes that determine their behaviours during the simulation. These attributes include priority, probability, service time, and max firing queue. Figure 1 provides an example of a transition's properties palette (on the right-hand side of the figure), which displays its primary attributes. Priority and probability are used to resolve conflicts among transitions demanding tokens from the same place. Service time is the firing duration of the transition, which can be a constant value or a probability distribution. Users can change the firing duration type by clicking the "Edit" button in the properties palette. Figure 3 shows the properties editor for transition T1, which allows the user to specify the firing duration as a triangular distribution with minimum, mode, and maximum values of 5, 12, and 18 time units, respectively.

The term "max firing queue" refers to the maximum number of times a transition can fire simultaneously. This feature is handy for modelling certain construction behaviours. For example, when two loaders are working together to load three trucks, with each loader handling one truck at a time, there are instances when loading activities for two trucks occur simultaneously or overlap. The "max firing queue" feature can be used in this case.

Consider the initial state of a truck-loading model, as shown in Figure 4. Three trucks are located at P1, while two loaders are stationed at P2. By setting the maximum firing queue to 2, as shown in Figure 5, T1 can fire twice overlappingly. When firing, the number 2 displayed in the middle of T1 indicates that the transition handles two firings simultaneously. If the maximum firing queue were set to 1 (the default value), T1 could only fire once at a time. This scenario would not accurately reflect the real-world situation in which two loaders are available to handle the loading process simultaneously. Finally, the model in Figure 6 represents the circumstances when one truck is still being loaded while another truck has already finished loading. The number 1 displayed in the middle of T1 indicates that only one firing is being handled by T1 at this point in time.



Seed: 21 | Time Interval: 5 | Time Limit: 100000 | Time Multiply: 1

Properties Editor : T1

ID :	0
Type :	Transition
Name :	T1
Priority :	1
Probability :	1
Max Firing :	1
service Time :	Triangular
a :	5
m :	12
b :	18

SAVE CANCEL

$$f(x)=\begin{cases} \frac{2(x-a)}{(b-a)(m-a)} & \text{if } a\le x\le m \\ \frac{2(b-x)}{(b-a)(b-m)} & \text{if } m<x\le b \\ 0 & \text{otherwise}\end{cases}$$

EDIT

Field	Value
Name	T1
Type	Transition
Priority	1
Probability	1
Service Time	Constant
c	1
-	0
-	0
Max Firing Queue	1

Time : 0 Event : 0 Step : 0

Figure 3. Properties Editor of Transition.

3) Token and Arc: Tokens and arcs in the COSMOS Simulator serve the same function as those in common Petri Nets. In the current version of the simulator, all tokens and arcs are black and do not have any additional attributes or colours.

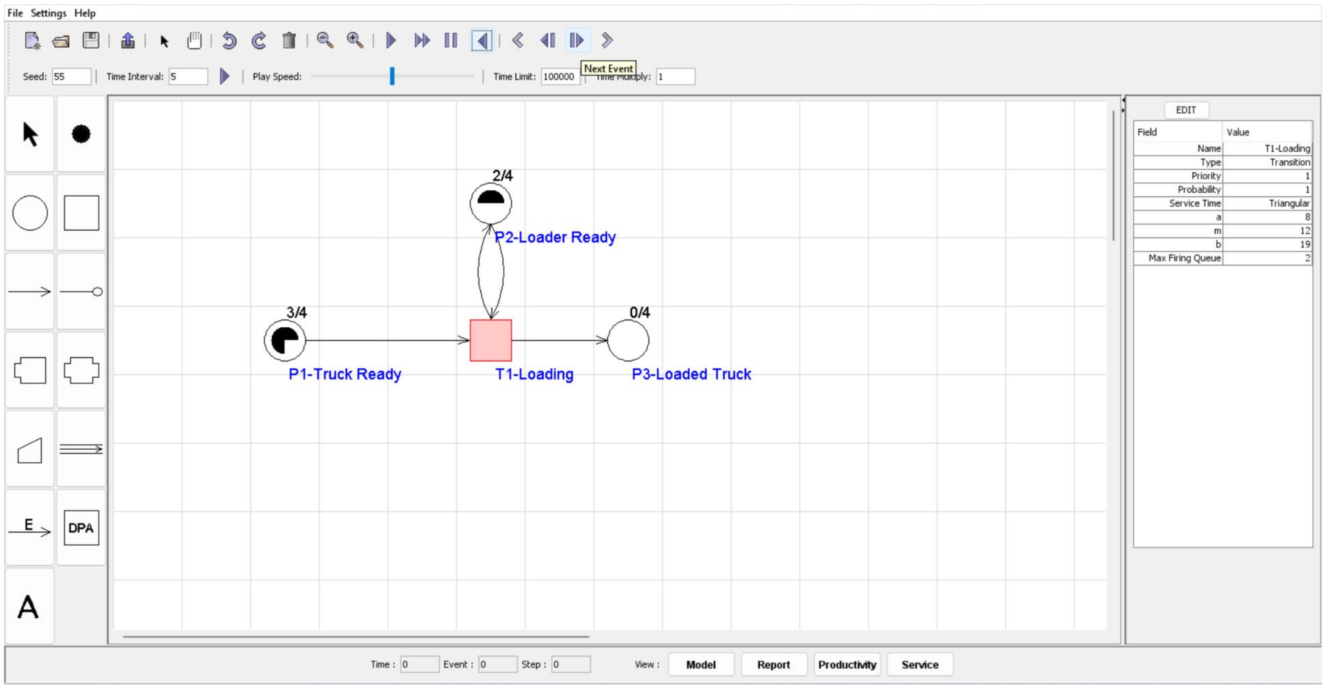
4) Condition Arc: Condition arcs in the COSMOS Simulator share similarities with inhibitor arcs found in modified Petri Nets, although substantial disparities exist between them. While the weight on a typical inhibitor arc is fixed at "equals zero," a condition arc possesses the flexibility to adopt any integer value as its weight, thereby enabling the expression of conditions in either equality or inequality formats. For instance, a condition arc's weight can be designated as "greater than or equal to 4." Additional instances illustrating the practical applications of condition arcs can be found in references [6][10] or a brief model delineated in Figure 7.

The model depicted in Figure 7 entails the transportation of 8 pieces of precast elements from a casting plant to a construction site. A loader situated at the plant facilitates the loading of precast elements onto a truck for transportation while also managing the unfinished precast elements within the plant. Nonetheless, the primary emphasis of this operation lies in the transportation of the eight precast elements. Consequently, the simulation of the process necessitates termination upon the completion of transporting the eight elements to the construction site and the subsequent return of the truck to the plant. In this model, a condition arc with a weight of ">= 1" (greater than or equal to one) is employed to govern the cessation of the process.

These features of condition arcs are handy for modellers who require control over specific logic or conditions in their construction process models. The features allow modellers to make their models more concise.

5) Header, Follower, Buffer, Pipe, and End Arc: Specific construction activities can only begin after their respective preceding activities have operated for a designated period. However, the completion of preceding activities is not mandatory before commencing the succeeding ones. When two or more activities share this interdependent relationship, they are classified as overlapping activities. To manage such overlapping activities, the COSMOS Simulator utilises five modelling elements: Header, Follower, Buffer, Pipe, and End Arc. Figure 8 displays the symbols of the five elements in the "modelling element panel" of the COSMOS Simulator.

A header is a unique transition type representing the first activity in a series of overlapping activities. Like a normal transition, it can be enabled and fired (shot). The primary function of a header is to convert discrete units of work into continuous units, represented as a percentage. When a header shoots, it sends portions of the work through pipes and a buffer to the next activity in the series.

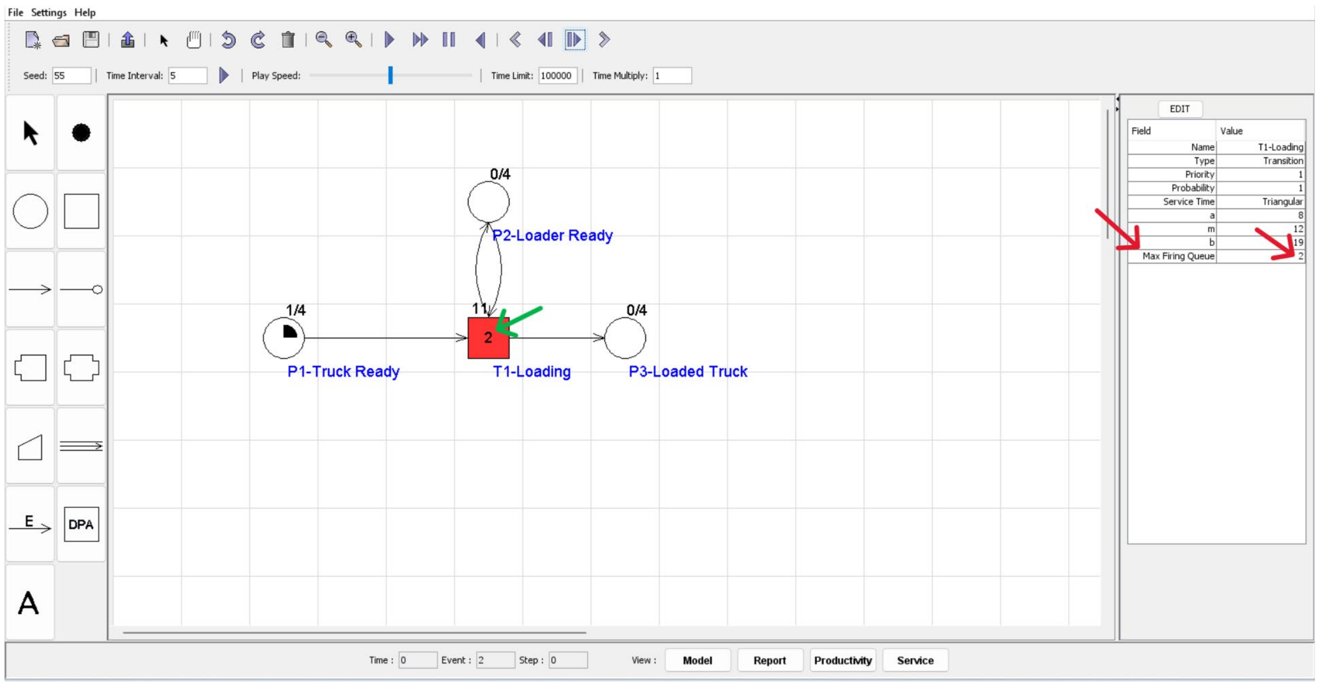


Seed: 55 | Time Interval: 5 | Time Limit: 100000 | Time Multiply: 1

Field	Value
Name	T1-Loading
Type	Transition
Priority	1
Probability	1
Service Time	Triangular
a	8
m	12
b	19
Max Firing Queue	2

Time : 0 Event : 0 Step : 0

Figure 4. Model illustrating “Max Firing Queue” feature (State 1).

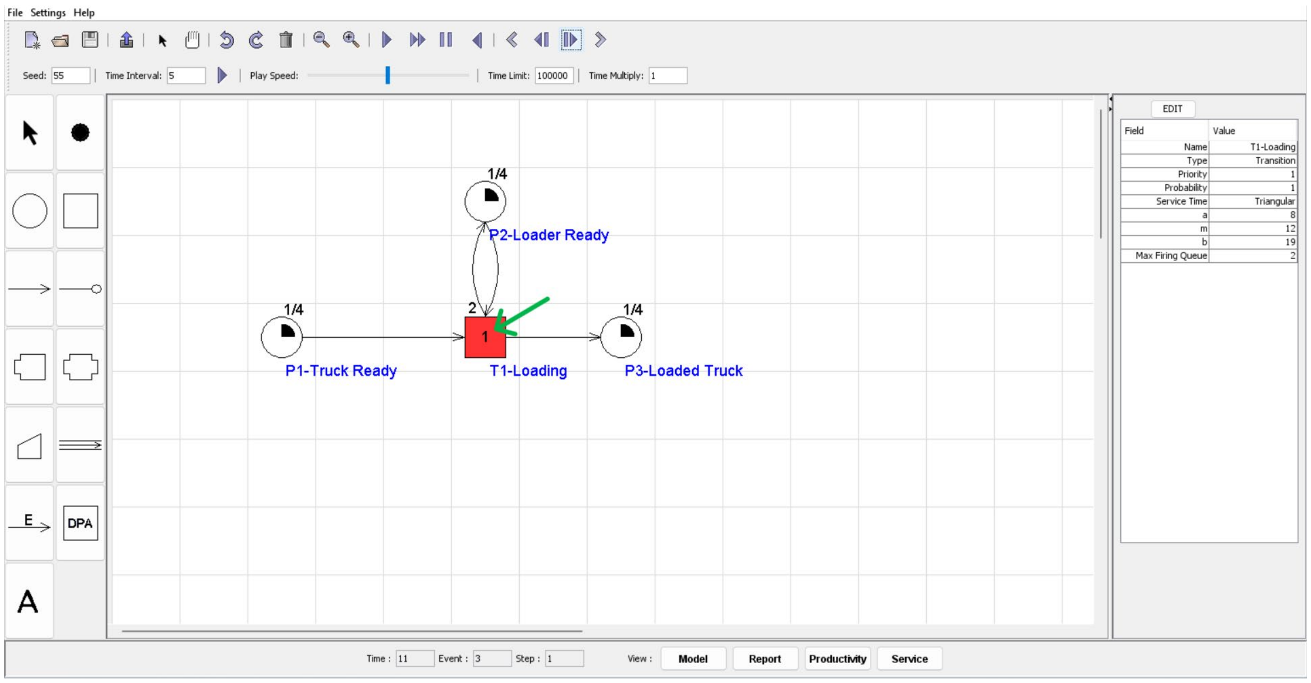


Seed: 55 | Time Interval: 5 | Time Limit: 100000 | Time Multiply: 1

Field	Value
Name	T1-Loading
Type	Transition
Priority	1
Probability	1
Service Time	Triangular
a	8
m	12
b	19
Max Firing Queue	2

Time : 0 Event : 2 Step : 0

Figure 5. Model illustrating “Max Firing Queue” feature (State 2).



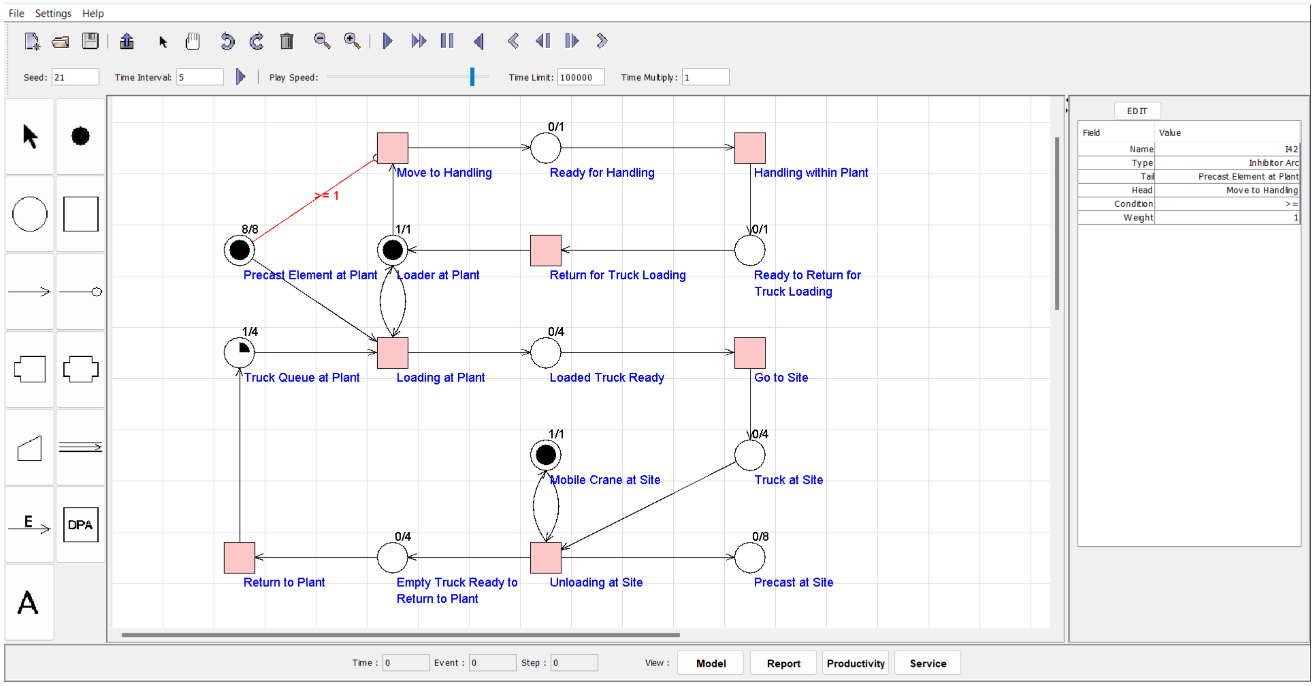
Seed: 55 | Time Interval: 5 | Time Limit: 100000 | Time Multiply: 1

Time : 11 Event : 3 Step : 1

EDIT

Field	Value
Name	T1-Loading
Type	Transition
Priority	1
Probability	1
Service Time	Triangular
a	8
m	12
b	19
Max Firing Queue	2

Figure 6. Model illustrating “Max Firing Queue” feature (State 3)



Seed: 21 | Time Interval: 5 | Time Limit: 100000 | Time Multiply: 1

Time : 0 Event : 0 Step : 0

EDIT

Field	Value
Name	I42
Type	Inhibitor Arc
Tail	Precast Element at Plant
Head	Move to Handling
Condition	>=
Weight	1

Figure 7. Model illustrating a sample application of "Condition Arc".

.

Additional details regarding the shooting mechanism and the functionality of headers can be found in [4].

A follower can be regarded as a particular type of transition, similar to a header. However, followers represent subsequent activities instead of representing the first activity in a series of overlapping activities. Like headers, followers release portions of continuous work through shootings. The quantity of work released from each shooting of a follower is equal to the shooting percentage specified in the header of the series. The shooting criteria for a follower are the same as those for a normal transition, with the additional condition that the released quantity of work from the preceding element (either a header or another follower) must be available in the input buffer of the follower. Further details on the functionality of followers can be found in [4].

A buffer is a special type of place where portions of the quantity of work released from headers or followers are stored. Buffers are connected to headers or followers via pipes. It's important to note that tokens cannot reside in buffers, and buffers have an unlimited capacity.

A pipe is a particular type of arc used to represent the flow of work released from headers or followers. In other words, pipes are used to send portions of work resulting from shootings of headers or followers. Pipes can only connect headers or followers to buffers and buffers to followers.

The COSMOS Simulator utilises an "end arc" to conclude overlapping series when the shooting percentage of the final follower reaches 100%. Once this threshold is met, the end arc sends a token or tokens to the connected outgoing place, with the number of tokens depending on the weight of the arc. This mechanism effectively terminates the series and ensures proper execution of the simulation.

Reference [11] demonstrates the use of the five elements (header, follower, buffer, pipe, and end arc) in a sample application to simulate overlapping activities in a concreting and waste-handling operation. The COSMOS model of the operation is shown in Figure 9.

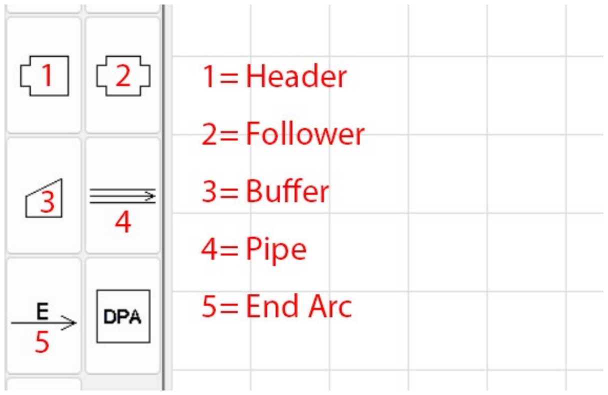
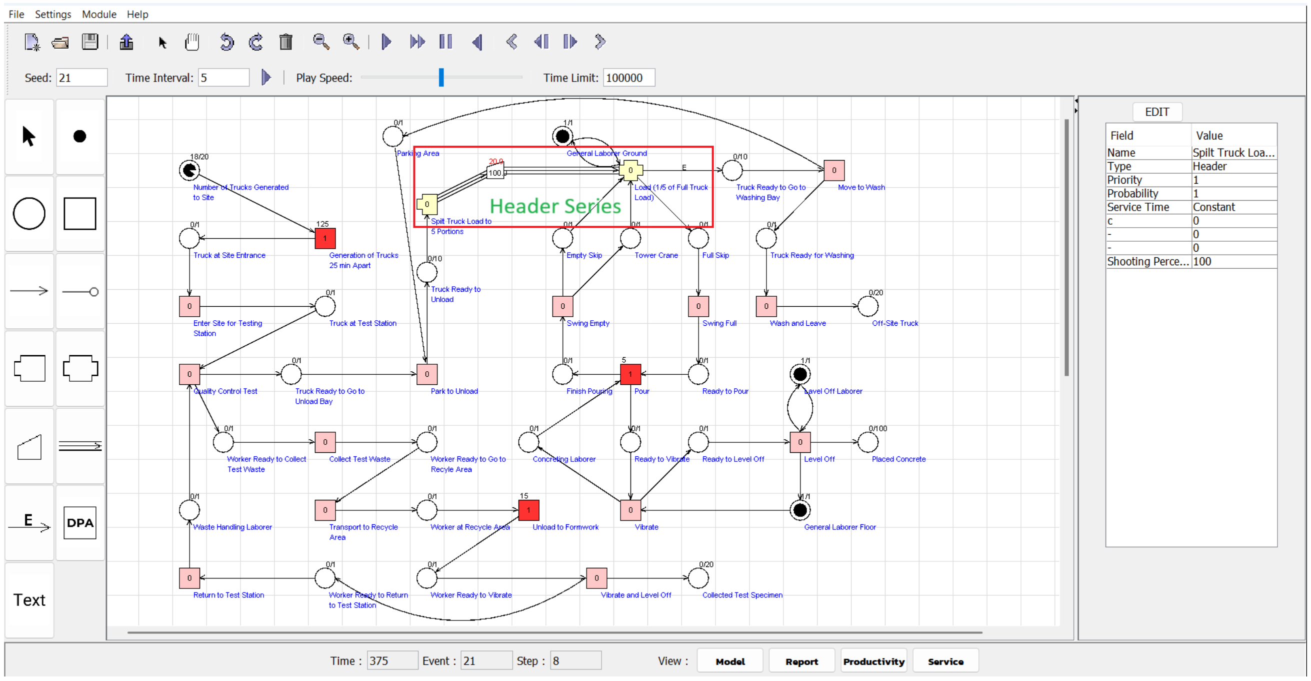


Figure 8. Header, Follower, Buffer, Pipe, and End Arc in Modelling Element Panel of the COSMOS Simulator.



Seed: 21 Time Interval: 5 Time Limit: 100000

Field	Value
Name	Spilt Truck Loa...
Type	Header
Priority	1
Probability	1
Service Time	Constant
c	0
-	0
-	0
Shooting Perce...	100

Time : 375 Event : 21 Step : 8 View :

Figure 9. COSMOS model with Header, Buffer, Follower, Pipes, and End Arc.

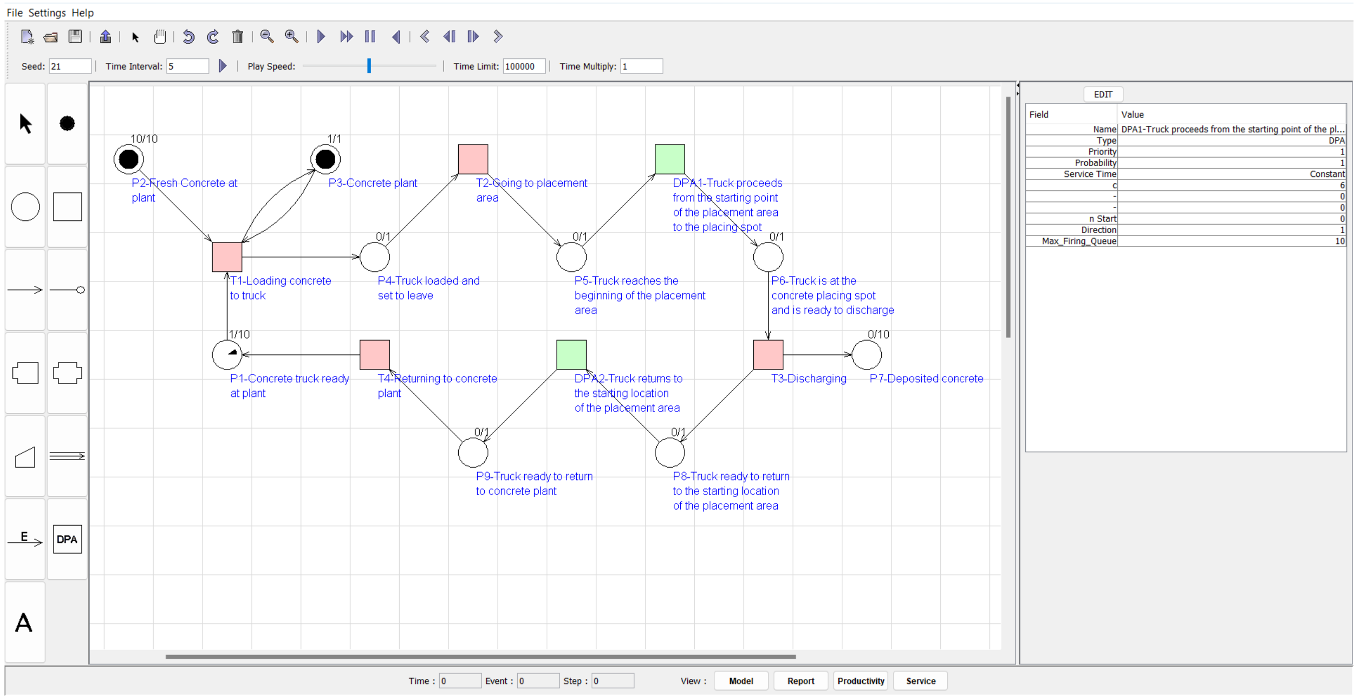
6) Dynamically Progressive Activity (DPA): Dynamically Progressive Activity (DPA) is defined in COSMOS as an activity whose duration varies due to the increase in the amount of work for each iteration. DPAs commonly occur in linear construction processes such as road construction and drainage pipe installation. For example, in reinforced-concrete road construction, the "moving to placing spot" activity will have a longer duration as the length of the road being constructed increases with each iteration of the placement. This is because the starting point of the placement area remains stationary while the placing spots get further away for each round of the placement. As a result, the distances between the beginning of the placement zone and the placing spots increase, thereby increasing the duration of the "moving to placing spot" activity performed by ready-mixed concrete trucks.

If a DPA's working rate and amount of work are known, its activity duration can be calculated. For instance, in reinforced-concrete road construction, suppose a concrete truck moves between the starting point of the placement area and a placing spot at an average speed of 10 km/hr or 166.67 m/min (this represents the working rate), and the distance between the beginning of the placement zone and the placing spot is 100 m (this represents the amount of work). In this case, the duration required for the truck in the "moving to placing spot" activity will be 0.6 minutes, indicating that, on average, the truck can cover a distance of 100 m within 0.6 minutes. Therefore, for distances of 200 m, 300 m, and 400 m, the truck will require 1.2, 1.8, and 2.4 minutes, respectively, to complete the activity.

After determining the duration of a DPA, users can input this information into the corresponding activity within the COSMOS Simulator. Subsequently, the simulator will calculate the duration of each iteration of the DPA by incrementally advancing the amount of work completed and using these values to simulate the process.

Figure 10 presents a concrete-road placement model, representing an operation similar to the abovementioned process. The model showcases the implementation of the DPA concept. Notably, a DPA element in the COSMOS Simulator is a unique type of transition that features a dynamically progressive firing duration. In the figure, the elements labelled "DPA1-Truck proceeds from the starting point of the placement area to the placing spot" and "DPA2-Truck returns to the starting location of the placement area" represent DPAs. When DPA1 fires for the first time, its firing duration will be zero since a truck can discharge concrete immediately upon reaching the starting point of the placement area without needing to move further forward. In the subsequent three iterations, the firing durations will be 0.6 minutes, 1.2 minutes, and 1.8 minutes as the placing points for the truck will be located 100 meters, 200 meters, and 300 meters away, respectively, from the beginning of the placement zone.

These modelling features collectively enable users to construct detailed and realistic simulations of construction processes.


File Settings Help
EDIT
10/10
P2-Fresh Concrete at plant
1/1
P3-Concrete plant
T2-Going to placement area
DPA1-Truck proceeds from the starting point of the placement area to the placing spot
T1-Loading concrete to truck
0/1
P4-Truck loaded and set to leave
0/1
P5-Truck reaches the beginning of the placement area
0/1
P6-Truck is at the concrete placing spot and is ready to discharge
1/10
P1-Concrete truck ready at plant
T4-Returning to concrete plant
DPA2-Truck returns to the starting location of the placement area
T3-Discharging
0/10
P7-Deposited concrete
0/1
P9-Truck ready to return to concrete plant
0/1
P8-Truck ready to return to the starting location of the placement area
DPA
A
View :
Model
Report
Productivity
Service

Seed: 21 | Time Interval: 5 | Play Speed: | Time Limit: 100000 | Time Multiply: 1

Field	Value
Name	DPA1-Truck proceeds from the starting point of the pl...
Type	DPA
Priority	1
Probability	1
Service Time	Constant
c	6
-	0
-	0
n Start	0
Direction	1
Max_Firing_Queue	10

Time : 0 Event : 0 Step : 0

Figure 10. Dynamically Progressive Activities (DPAs) in a concrete-road placement model.

While the COSMOS methodology is based on an extended Petri Net framework, its implementation in the COSMOS Simulator intentionally abstracts away much of the theoretical and mathematical complexity commonly associated with Petri Nets. Users are not required to understand formal definitions such as marking functions or matrix-based calculations, although a basic understanding of how tokens move and how transitions fire remains necessary to develop valid models. Importantly, users do not need to interact directly with abstract Petri Net syntax. Instead, COSMOS offers domain-specific visual blocks—such as headers, followers, buffers, pipes, and dynamically progressive activities—that closely represent real construction operations. As long as users can identify construction activities and define logical relationships between them (e.g., precedence, concurrency, or dependencies), they can effectively create simulation models. This design lowers the entry barrier for construction professionals, ensuring practical usability while retaining the analytical power of a Petri Net-based system.

III. Practical Applications and Validation of Cosmos Simulator

The COSMOS Simulator has been used to simulate various construction processes, and its accuracy has been demonstrated. This section will provide further summaries and discussion of previous studies on the software's practical applications and validation. These aspects will further reinforce confidence in the accuracy and reliability of the COSMOS software and system, which is crucial for its widespread adoption. Greater utilisation of the software will, in turn, facilitate continuous improvements and advancements in the COSMOS Simulator, enhancing its capability not only in construction process simulation but also in modelling other process-driven operations. Six cases from "Energy Reduction" to "Concreting and Waste-Handling Operation" will be discussed in this section.

A. Energy Reduction

The COSMOS Simulator was applied in the study to analyse and optimise the utilisation of heavy equipment fleets in an asphaltic-concrete road construction project in Ratchaburi, Thailand [5]. The research aimed to minimise energy consumption by identifying inefficiencies in the construction process and improving resource allocation. The study involved modelling and simulating the construction operations using the COSMOS system, which is based on Petri Nets.

The analysis covered seven key construction procedures, including clearing, levelling, excavation, embankment construction, base preparation, prime coating, and pavement finishing. Data on activity durations, equipment usage, and energy consumption were collected from an actual construction site. The simulation identified significant idle and waiting times within the equipment fleet, which contributed to excessive fuel consumption. Specifically, the original arrangement of one truck, one water truck, and one asphalt truck resulted in a total process duration of 5,653 minutes, with substantial waiting times for key equipment.

Process improvements were implemented by increasing the number of trucks and asphalt trucks from one to three while keeping the number of water trucks constant. The revised simulation showed a substantial reduction in process duration to 2,969 minutes and significantly decreased idle times. As a result, energy consumption was reduced by 26.8%, lowering diesel fuel usage from 523 litres to 383 litres.

These findings demonstrate the effectiveness of the COSMOS Simulator in optimising construction operations through systematic modelling and simulation. The study highlights the potential of process adjustments in achieving energy efficiency in road construction projects. However, the research was limited to a specific project scope and construction setting, and further studies could explore the broader applicability of the approach to different project types and conditions.

B. Optimisation of Supply Trains in Tunnel Boring Operation

The COSMOS Simulator was employed in this study to optimise the operation of supply trains in tunnel boring using tunnel boring machines (TBMs) [6]. The research focused on the Beung-Nongbon drainage tunnel project in Bangkok, Thailand, which required an efficient muck evacuation system due to the unusual tunnel length of 5.5 km without an intermediate vertical shaft. The challenge was to determine the optimal number of supply trains needed to synchronise with the TBM's excavation process.

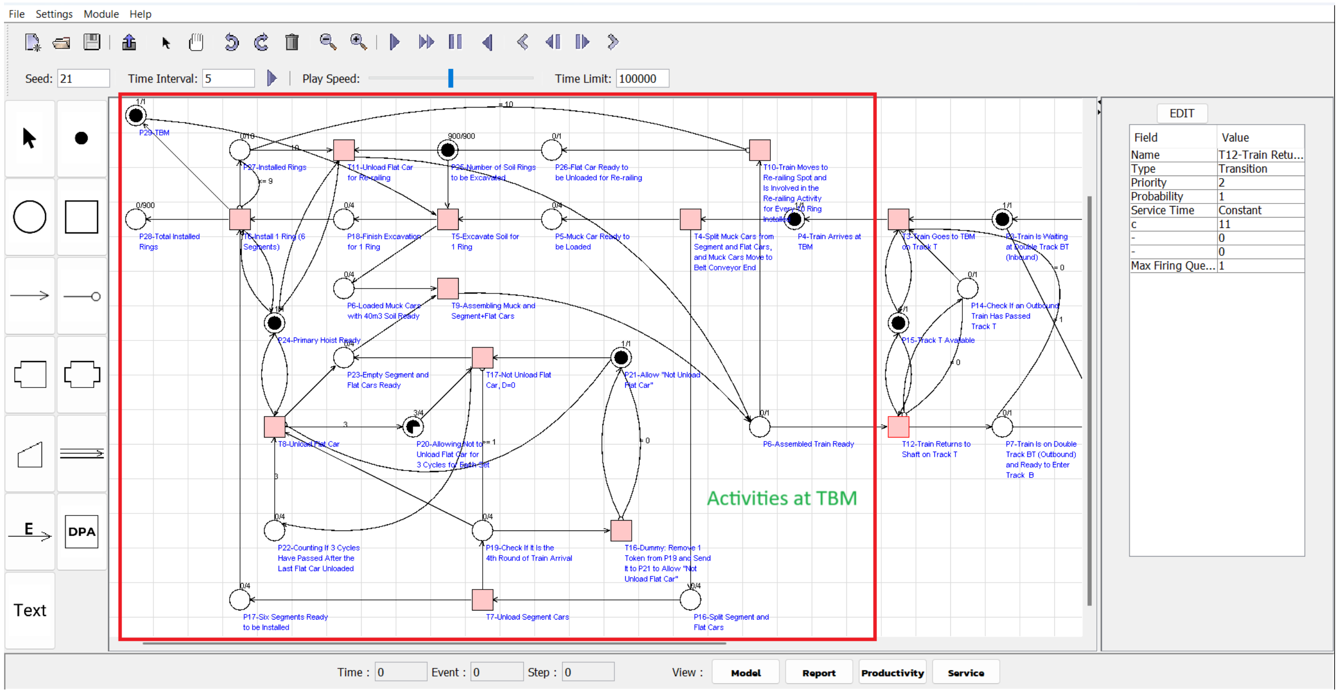
A COSMOS (Petri Net-based) model of the tunnelling operation was developed and simulated using the COSMOS system (see Figures 11 and 12). The model accounted for key activities, including muck evacuation, tunnel segment transportation, and rail relocation. The results revealed that the number of supply trains required varied based on the tunnel length. For example, the optimal number of supply trains was found to be two for tunnels up to 0.9 km, three for 0.9–2.7 km, four for 2.7–4.5 km, and five for 4.5–5.5 km. The study also determined the optimal placement of double track points to avoid bottlenecks, suggesting their positioning at 1.8, 3.6, and 4.5 km for tunnels longer than 4.5 km.

A critical finding was that deadlock situations could occur if supply trains were not strategically positioned at double-track points. The simulation also established that the maximum permissible single-track length (Track T) between the last double track point and the TBM should not exceed 0.9 km, ensuring continuous TBM operation with minimal delays. The maximum allowable distance between adjacent double track points was found to be 2.3 km to maintain optimal productivity. The study confirmed that the highest achievable tunnelling rate was 27.8 rings per day, aligning with historical data from similar projects.

These findings demonstrate the effectiveness of the COSMOS Simulator in optimising TBM operations by improving synchronisation between excavation and muck evacuation processes. However, the study's limitations include the assumption of ideal conditions without equipment breakdowns or unforeseen operational delays. Future research could incorporate these uncertainties to refine the optimisation approach further.

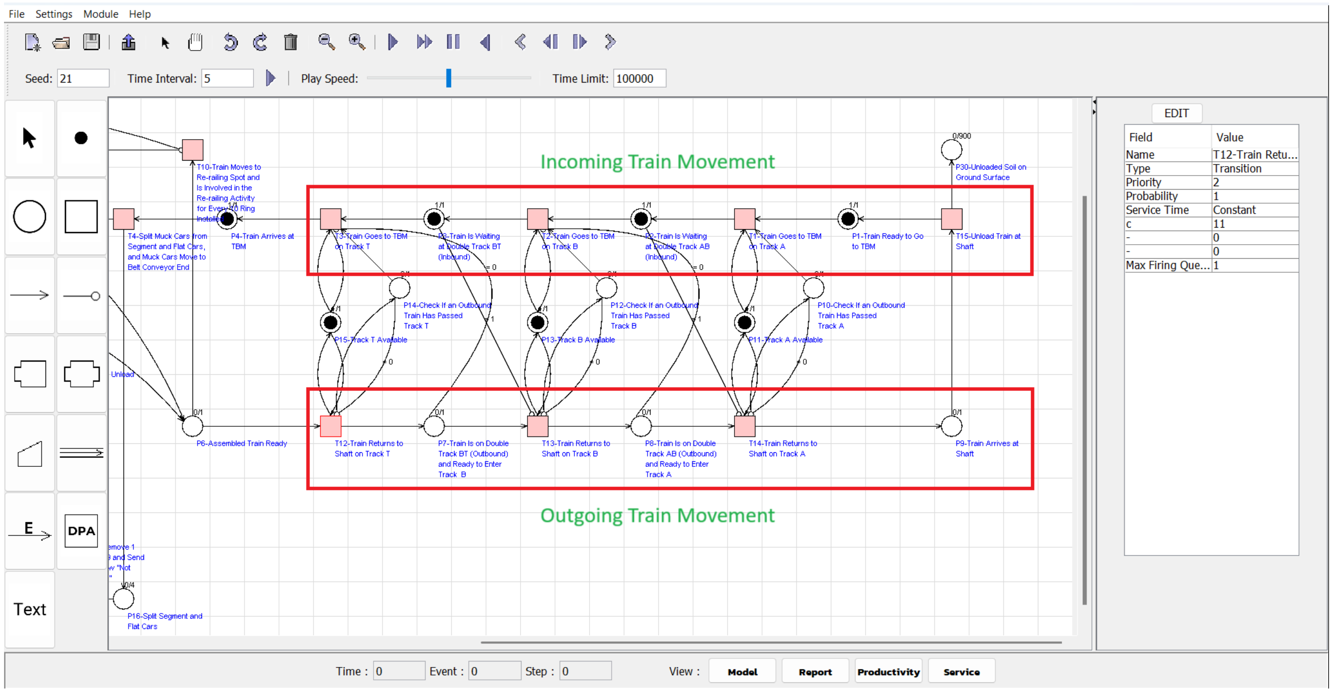
C. *Resource Management for Concrete Placing Operation*

The COSMOS Simulator was applied in this study to analyse and optimise the resource management of a concrete-placing operation in a gas separation plant construction project in Rayong Province, Thailand [7]. The research focused on reducing the duration of concrete placement by improving the coordination of ready-mixed concrete trucks, crane operations, and other logistical factors.


File Settings Module Help
Seed: 21
Time Interval: 5
Play Speed:
Time Limit: 100000
Text
DPA
Activities at TBM
P29-TBM
P27-Installed Rings
T11-Unload Flat Car for Re-railing
P25-Number of Soil Rings to be Excavated
P26-Flat Car Ready to be Unloaded for Re-railing
T10-Train Moves to Re-railing Spot and is Involved in the Re-railing Activity for Every 10 Ring Installation
P28-Total Installed Rings
T6-Install 1 Ring (6 Segments)
P18-Finish Excavation for 1 Ring
T5-Excavate Soil for 1 Ring
P5-Muck Car Ready to be Loaded
T4-Split Muck Cars from Segment and Flat Cars, and Muck Cars Move to Belt Conveyor End
P4-Train Arrives at TBM
T3-Train Goes to TBM on Track T
P3-Train Is Waiting at Double Track BT (Inbound)
P6-Loaded Muck Cars with 40m3 Soil Ready
T9-Assembling Muck and Segment+Flat Cars
P14-Check If an Outbound Train Has Passed Track T
P24-Primary Hoist Ready
P15-Track T Available
P23-Empty Segment and Flat Cars Ready
T17-Not Unload Flat Car, D=0
P21-Allow "Not Unload Flat Car"
T8-Unload Flat Car
P20-Allowing Not to Unload Flat Car for 3 Cycles for Each Set
P8-Assembled Train Ready
T12-Train Returns to Shaft on Track T
P7-Train Is on Double Track BT (Outbound) and Ready to Enter Track B
P22-Counting If 3 Cycles Have Passed After the Last Flat Car Unloaded
P19-Check If It Is the 4th Round of Train Arrival
T16-Dummy: Remove 1 Token from P19 and Send It to P21 to Allow "Not Unload Flat Car"
P17-Six Segments Ready to be Installed
T7-Unload Segment Cars
P16-Split Segment and Flat Cars
Time : 0
Event : 0
Step : 0
View :
Model
Report
Productivity
Service
EDIT

Field	Value
Name	T12-Train Retu...
Type	Transition
Priority	2
Probability	1
Service Time	Constant
c	11
-	0
-	0
Max Firing Que...	1

Figure 11. Partial COSMOS model represents activities at TBM in a tunnel construction operation [6].


File Settings Module Help
Seed: 21
Time Interval: 5
Play Speed:
Time Limit: 100000
Text
DPA
Incoming Train Movement
Outgoing Train Movement
T10-Train Moves to Re-railing Spot and is Involved in the Re-railing Activity for Every 10 Ring Installation
P30-Unloaded Soil on Ground Surface
T4-Split Muck Cars from Segment and Flat Cars, and Muck Cars Move to Belt Conveyor End
P4-Train Arrives at TBM
T3-Train Goes to TBM on Track T
P3-Train Is Waiting at Double Track BT (Inbound)
T2-Train Goes to TBM on Track B
P2-Train Is Waiting at Double Track AB (Inbound)
T1-Train Goes to TBM on Track A
P1-Train Ready to Go to TBM
T15-Unload Train at Shaft
P14-Check If an Outbound Train Has Passed Track T
P12-Check If an Outbound Train Has Passed Track B
P10-Check If an Outbound Train Has Passed Track A
P15-Track T Available
P13-Track B Available
P11-Track A Available
P8-Assembled Train Ready
T12-Train Returns to Shaft on Track T
P7-Train Is on Double Track BT (Outbound) and Ready to Enter Track B
T13-Train Returns to Shaft on Track B
P8-Train Is on Double Track AB (Outbound) and Ready to Enter Track A
T14-Train Returns to Shaft on Track A
P9-Train Arrives at Shaft
P16-Split Segment and Flat Cars
Time : 0
Event : 0
Step : 0
View :
Model
Report
Productivity
Service
EDIT

Field	Value
Name	T12-Train Retu...
Type	Transition
Priority	2
Probability	1
Service Time	Constant
c	11
-	0
-	0
Max Firing Que...	1

Figure 12. Partial COSMOS model illustrates the movement of supply trains in a tunnel construction operation [6].

A COSMOS-based simulation model was developed to represent the real-world construction process. The study collected empirical data over five months to identify patterns in truck arrival times, concrete placement durations, and crane availability. Various scenarios were simulated, adjusting the number of trucks and daily concrete placement volumes. The results indicated that using three concrete trucks and placing 50 cubic meters of concrete per day provided the optimal balance between efficiency and resource utilisation. Implementing this strategy in the second phase of the project resulted in a noticeable improvement in construction progress compared to the first phase, where no simulation-based planning was used. The optimised approach reduced project delays and eliminated unnecessary truck waiting times, which had previously led to inefficiencies.

The study demonstrates the effectiveness of the COSMOS Simulator in optimising construction scheduling and resource allocation through systematic modelling and simulation. However, limitations include the exclusion of unpredictable external factors such as weather conditions and equipment breakdowns. Future studies could incorporate stochastic modelling to account for these uncertainties and enhance the robustness of the simulation approach.

D. Comparison between COSMOS Simulator and Arena

The COSMOS Simulator has been verified and applied in various construction process simulations, with a comparative analysis conducted against the widely used Arena simulation software [8]. Two key case studies were employed to validate COSMOS's accuracy and applicability: (1) concrete placement using a concrete pump and (2) earthmoving operations in tunnel excavation.

The first case study examined the concrete placement process for an eight-story building, where ready-mix concrete was transported by trucks and pumped through pipes to the designated floors. The model accounted for potential operational disruptions, such as pipe blockages and relocations. Simulation results from COSMOS and Arena showed strong consistency in key performance metrics, including the concrete pouring rate and waiting times at the mixing plant and pump. While minor variations were observed due to stochastic process durations, statistical hypothesis testing confirmed no significant differences between the two software outputs.

The second case study focused on tunnel excavation, specifically the earthmoving operations involving a tunnel boring machine (TBM) and supply trains operating on a single-track system with designated passing stations. The COSMOS model successfully replicated the process flow, including train scheduling, material transport, and resource allocation. The total process duration from both COSMOS and Arena simulations matched precisely at 46,690 minutes, further validating COSMOS's reliability.

These verifications demonstrate that the COSMOS Simulator can produce accurate results comparable to Arena, reinforcing its credibility as a tool for construction process simulation. Moreover, the study highlights the suitability of COSMOS for modeling construction workflows due to its Petri Net-based methodology, which aligns well with the logic of construction scheduling techniques such as the Critical Path Method (CPM). This feature enhances model interpretability for construction professionals, distinguishing COSMOS from manufacturing-based simulation tools like Arena

Overall, the findings confirm COSMOS's robustness in simulating complex construction operations, making it a viable alternative to established simulation software for process analysis and optimisation in the construction industry.

E. Auger Horizontal Earth-Boring Process: Comparison with MicroCYCLONE

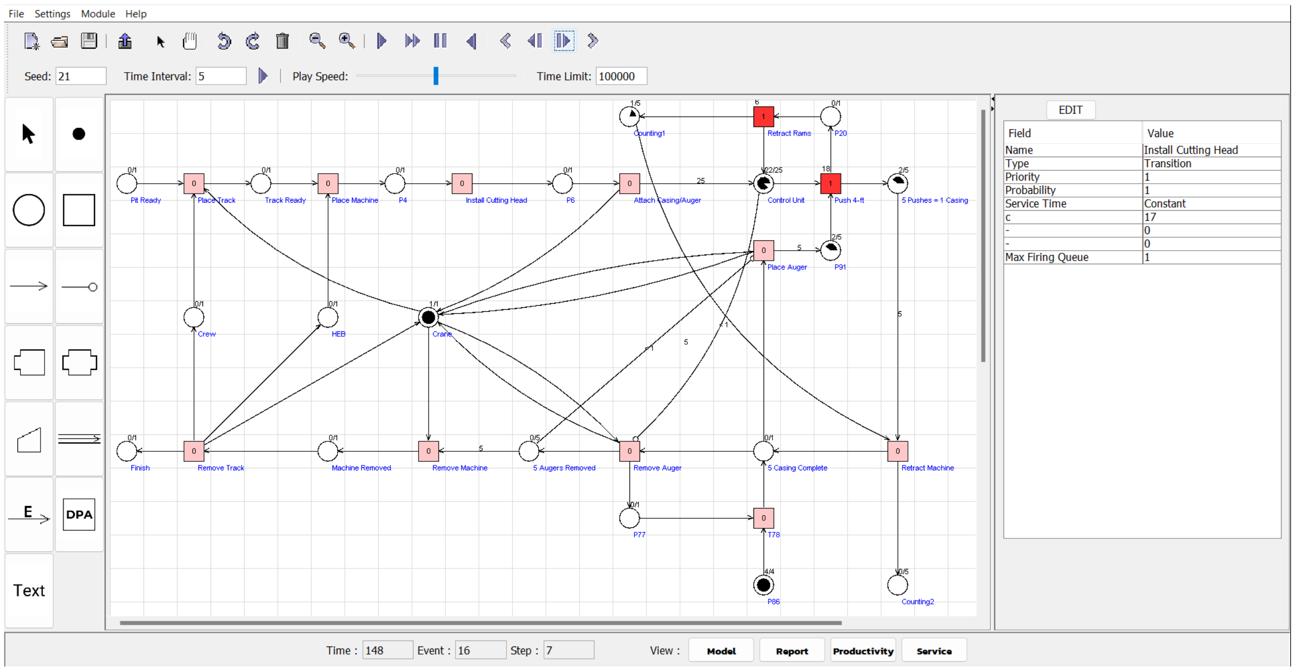
The COSMOS Simulator was evaluated against MicroCYCLONE in modelling an Auger Horizontal Earth-Boring (HEB) process [11]. The case study involved the installation of 100 linear feet of casing using an auger boring machine, with activities including track placement, auger installation, casing attachment, and soil removal. Both deterministic and stochastic simulations were performed to compare the total operation duration.

The deterministic results showed an identical process completion time of 1,107 minutes for both simulators, confirming COSMOS's ability to replicate MicroCYCLONE's output. In stochastic simulations, 40 independent runs were conducted, yielding mean process durations of 1,092 minutes for MicroCYCLONE and 1,096 minutes for COSMOS. A statistical hypothesis test confirmed that the differences were not significant, verifying that COSMOS produces statistically equivalent results to MicroCYCLONE. This validation demonstrates COSMOS's capability to model cyclic construction processes accurately while leveraging Petri Net-based representations that facilitate process visualisation. The COSMOS model of the operation is given in Figure 13.

F. Concreting and Waste-Handling Operation: Comparison with PROMODEL

A second verification in [11] compared the COSMOS Simulator with PROMODEL and SDESA in modeling a concreting and waste-handling operation. The case study involved a tower crane-based concrete pouring process, incorporating skip cycles, slump testing, and waste handling. The COSMOS model shown in Figure 14 effectively captured the flow of materials, truck arrivals, and resource allocations using Petri Net constructs.

The results were analysed in terms of resource utilisation rates. COSMOS exhibited only a 0.6% deviation from PROMODEL, demonstrating a high degree of consistency. When compared to SDESA, COSMOS showed a slightly larger deviation of 3.4%, but this was within acceptable margins given the stochastic nature of the process. An alternative COSMOS model incorporating a more advanced resource allocation technique (using headers, buffers, and end arcs) (see Figure 9) further reduced the deviation from PROMODEL to just 0.1%. This suggests that COSMOS not only provides accurate results but also offers enhanced flexibility for modeling complex resource interactions.

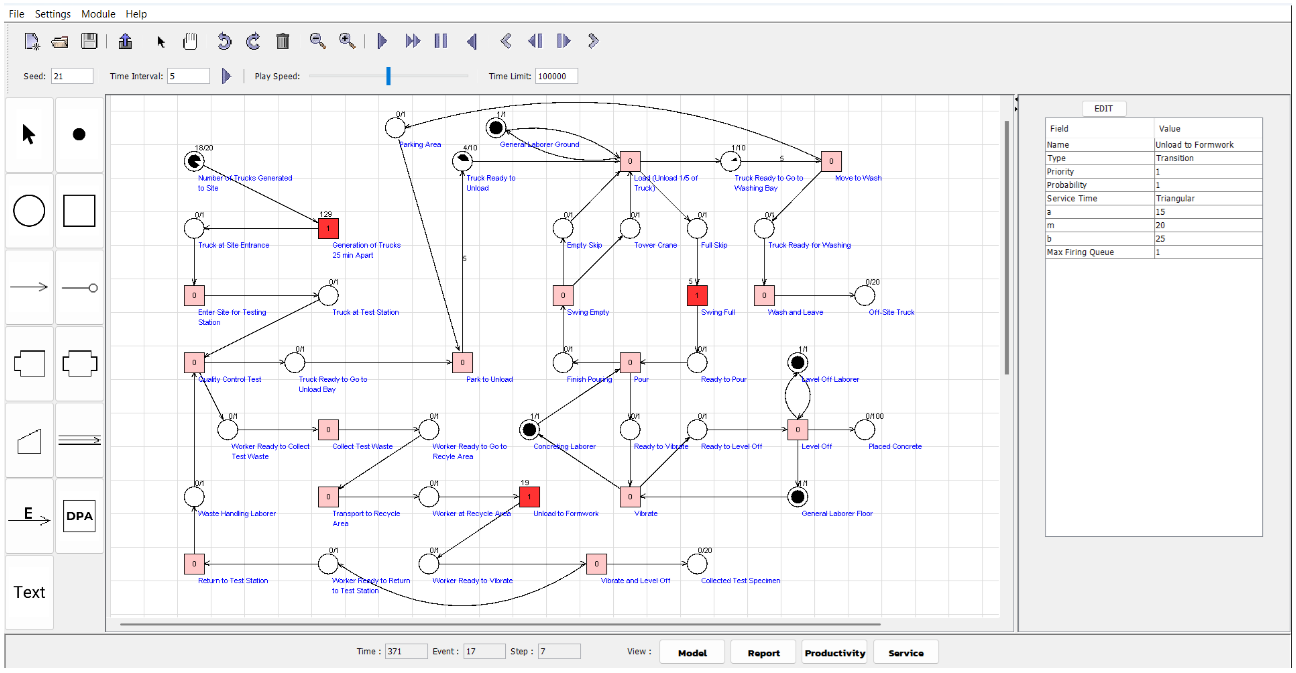


Seed: 21 Time Interval: 5 Time Limit: 100000

Field	Value
Name	Install Cutting Head
Type	Transition
Priority	1
Probability	1
Service Time	Constant
c	17
-	0
-	0
Max Firing Queue	1

Time : 148 Event : 16 Step : 7

Figure 13. COSMOS Model for an auger horizontal earth-boring process [11].



Seed: 21 Time Interval: 5 Time Limit: 100000

Field	Value
Name	Unload to Formwork
Type	Transition
Priority	1
Probability	1
Service Time	Triangular
a	15
m	20
b	25
Max Firing Queue	1

Time : 371 Event : 17 Step : 7

Figure 14. COSMOS model (Petri Nets-based model) of a concreting and waste-handling operation [11].

These verifications confirm that COSMOS can effectively replicate results from both domain-specific (MicroCYCLONE) and general-purpose (PROMODEL) simulation tools while providing construction engineers with an intuitive and domain-relevant modelling approach. The findings reinforce COSMOS's viability as a robust construction-process simulation tool, ensuring accuracy while enhancing process transparency and interpretability.

IV. Discussion

Unlike some existing simulators, which often require users to understand abstract Petri Net constructs or mathematical formalisms, the COSMOS Simulator was developed with a strong emphasis on usability and domain alignment. Its visual modelling components—such as headers, followers, buffer, pipe and dynamically progressive activities—reflect how construction professionals naturally conceptualise their operations. This domain-specific design lowers the learning curve and enables practitioners with a limited background in simulation theory to develop models that still leverage the expressive power of Petri Net-based logic.

In terms of modelling capability, COSMOS enables the representation of complex construction behaviours that are often difficult to express using traditional tools. These include overlapping activities, dynamically progressive operations, flexible precedence logic, and resource allocation constraints. While COSMOS delivers results comparable in accuracy to established simulators, its practical value lies in making such capabilities accessible and directly applicable to construction workflows. This combination of analytical strength and usability differentiates COSMOS as a simulation tool purpose-built for real-world construction process analysis and design.

While COSMOS represents a significant step forward, it is essential to acknowledge its limitations and potential areas for future development. The simulator's primary focus on discrete-event simulation may limit its applicability to continuous processes or systems with complex interactions. Additionally, although COSMOS can simulate dynamic processes, its current version may have limitations in incorporating real-time data from construction sites, which is crucial for achieving a true digital representation of construction processes. Future research and development efforts can focus on expanding COSMOS's capabilities in these areas, further enhancing its value and impact in the construction industry.

The current version of the COSMOS Simulator is available online [18].

V. Conclusion and Future Work

This paper provided a comprehensive description and illustration of the distinctive features of the COSMOS Simulator, a computer program designed to simulate construction processes effectively. COSMOS accounts for real-world construction behaviours such as:

- Concurrent execution of similar activities through "max firing queue" settings.
- Overlapping or interleaved activities facilitated by headers, followers, buffers, pipes, and end arcs.
- Simulation of Dynamically Progressive Activities (DPAs), where duration varies based on workload, commonly seen in tasks like asphalt paving.

Notably, these modelling elements and features—headers, followers, buffers, pipes, end arcs, and DPAs—are unique to COSMOS, enabling the simulation of specific behaviours found in construction, and they are not available in other simulation tools.

Apart from normal arcs, COSMOS also has condition arcs similar to inhibitor arcs in modified Petri Nets but allow more flexibility. They can have any integer weight, enabling the expression of equality or inequality conditions.

The COSMOS Simulator has been validated through various practical applications, demonstrating its accuracy and reliability in simulating construction processes. Its ability to model complex workflows, optimise resource utilisation, and produce results comparable to established simulation tools highlights its potential for broader adoption in the construction industry. However, further research is needed to explore the needs and gather feedback from diverse users. This information will be crucial in enhancing the COSMOS system to make it even more effective in simulating construction processes.

This paper offered a resource for researchers and practitioners interested in leveraging COSMOS for their construction modelling and simulation needs.

Despite its emphasis on functionality for construction practitioners, the COSMOS Simulator and its associated methodology can be used to model and simulate any discrete event process.

Future research and development efforts will focus on expanding COSMOS's capabilities to incorporate real-time data from construction sites. Additionally, it would be beneficial to broaden its functionality to simulate construction processes with even more complex interactions. Enhancing the modelling logic with appropriate control statements will enable users to have finer control over the behaviours of the COSMOS models and support an even wider range of use cases relevant to complex construction scenarios.

References

[1] J. Damrianant and S. Meklersuewong, “COSMOS Simulator: A Software Tool for Construction-Process Modelling and Simulation,” The Sixteenth International Conference on Advances in System Simulation (SIMUL 2024) IARIA, Sept. 2024, pp. 19–27, ISBN: 978-1-68558-197-8.

[2] B. Visartsakul and J. Damrianant, “A review of Building Information Modelling and simulation as virtual representations under the digital twin concept,” Eng. J., vol. 27(1), pp. 11-27, Jan. 2023, doi:10.4186/ej.2023.27.1.11.

[3] J. Damrianant, “COSMOS: A discrete-event modelling methodology for construction processes,” Int. J. Internet Enterp. Manag., vol. 1, pp. 128-152, Apr. 2003, doi:10.1504/IJIEM.2003.003209.

[4] J. Damrianant and R. R. Wakefield, “A Petri Net-Based Methodology for Modelling of Overlapping Activity Processes,” The Second International Conference on Construction Process Re-engineering (CPR-99), July 1999, pp. 375-386.

[5] S. Meklersuewong and J. Damrianant, “Energy Reduction in Road Construction,” The Third International Symposium on Engineering, Energy and Environments (ISEEE 3), Nov. 2013, pp. 523-532, ISBN: 978-974-466-715-1.

[6] J. Damrianant, “Optimisation of Supply Trains in Tunnel Boring Operation Using Tunnel Boring Machines,” The Sixth International Conference on Advances in Civil, Structural and Mechanical Engineering (CSM 2018), Apr. 2018, pp. 8-12, ISBN: 978-1-63248-150-4.

[7] J. Damrianant and T. Panrangsri, “Resource management using COSMOS modelling and simulation system to lessen

concrete-placing duration," (in Thai) Thai J. Sci. Tech., vol. 7(5), pp. 553-566, Aug. 2018, doi:10.14456/tjst.2018.50.

[8] N. Suri and J. Damrianant, "Comparing construction process simulation between the Arena and COSMOS programs," (in Thai). Eng. J. Res. Dev., vol. 30(4), pp. 89-104, Oct. 2019, ISSN: 2730-2733.

[9] B. Visartsakul and J. Damrianant, "Determining costs and time required for building construction by using 3D structural models, unit costs, productivity rates, and project simulations," (in Thai) Eng. J. Res. Dev., vol. 31(4), pp. 63-76, Oct. 2020, ISSN: 2730-2733.

[10] J. Damrianant, "Track management approaches for underground tunnel construction," Thai J. Sci. Tech., vol. 10(5), pp. 621-634, Sep. 2021, ISSN: 2286-7333.

[11] S. Meklersuewong and J. Damrianant, "Evaluating the COSMOS software ecosystem for domain-specific construction process simulation," Int. Rev. Model. Simul., vol. 15(3), pp. 179-188, Jun. 2022, doi:10.15866/iremos.v15i3.20268.

[12] J. Damrianant, "Comparison of process and project duration assessment approaches for an industrial water-system installation project using estimation method and COSMOS program," (in Thai) J. KMUTNB, vol. 33(1), pp. 127-139, Jan. 2023, doi:10.14416/j.kmutnb.2022.06.004.

[13] M. A. Drighiciu and D. C. Cismaru, "Modelling a water bottling line using Petri Nets," Annals of the University of Craiova. Electr. Eng. Ser., vol. 37, pp. 110–115, 2013, ISSN: 1842-4805.

[14] M. Herajy, F. Liu, C. Rohr, and M. Heiner, "Snoopy's hybrid simulator: A tool to construct and simulate hybrid biological models," BMC Syst. Biol., vol. 11(71), pp. 1-16, Jul. 2017, doi:10.1186/s12918-017-0449-6.

[15] R. Davidrajuh, D. Krenczyk, and B. Skolud, "Finding clusters in Petri Nets. An approach based on GPenSIM," Model. Identif. Control, vol 40(1), pp. 1-10, Jan. 2019, doi:10.4173/mic.2019.1.1.

[16] E. Kučera et al., "New software tool for modelling and control of discrete-event and hybrid systems using Timed Interpreted Petri Nets," Appl. Sci., vol. 10(15), pp. 5027, Jul. 2020, doi:10.3390/app10155027.

[17] V. B. Kumbhar1 and M. S. Chavan, "A Review of Petri Net Tools and Recommendations," The International Conference on Applications of Machine Intelligence and Data Analytics (ICAMIDA 2022), May 2023, pp. 710–721, doi:10.2991/978-94-6463-136-4_61.

[18] COSMOS Simulator. [Online]. Available from: https://drive.google.com/file/d/1qpxEvIq9TDNfynrqt_Wj5VEHGE3uBW53/view?usp=drive_link.

Designing at 1:1 Scale on Wall-Sized Displays Using Existing UI Design Tools

1st Lou Schwartz
Luxembourg Institute of Science and Technology (LIST)
Esch-sur-Alzette, Luxembourg
lou.schwartz@list.lu

2nd Mohammad Ghoniem
Luxembourg Institute of Science and Technology (LIST)
Esch-sur-Alzette, Luxembourg
mohammad.ghoniem@list.lu

3rd Valérie Maquil
Luxembourg Institute of Science and Technology (LIST)
Esch-sur-Alzette, Luxembourg
valerie.maquil@list.lu

4th Adrien Coppens
Luxembourg Institute of Science and Technology (LIST)
Esch-sur-Alzette, Luxembourg
adrien.coppens@list.lu

5th Johannes Hermen
Luxembourg Institute of Science and Technology (LIST)
Esch-sur-Alzette, Luxembourg
johannes.hermen@list.lu

***Abstract*—Wall-Sized Displays have spatial characteristics that are difficult to address during user interface design. The design at scale 1:1 could be part of the solution. In this paper, we present the results of two user studies and one technology review, exploring the usability of popular, desktop-optimized prototyping tools, for designing at scale on Wall-Sized Displays. We considered two wall-sized display setups, and three different interaction methods: touch, a keyboard equipped with a touchpad, and a tablet. We observed that designing at scale 1:1 was appreciated. Tablet-based interaction proved to be the most comfortable interaction method, and a mix of interaction modalities is promising. In addition, care must be given to the surrounding environment, such as furniture. We propose twelve design guidelines for a design tool dedicated to this specific context. Overall, existing user interface design tools do not yet fully support design on and for wall-sized displays and require further considerations in terms of placement of user interface elements and the provision of additional features.**

Keywords-wall-sized display; UI design; design at 1:1 scale; user study; large scale display.

I. Introduction

This paper extends a previous study on the usability of Figma, a popular user interface (UI) design tool, for Wall-Sized Displays (WSDs) at the CENTRIC 2024 conference [1]. WSDs are increasingly used in public spaces to provide general or contextual information, provide entertainment, or for artistic purposes [2] [3]. WSDs are also applied in many research areas. Traffic management [4] and automotive design [5] benefit from their large display area. They also support data browsing and manipulation [6] [7], and are crucial for visualizing and interacting with vast amounts of data in natural sciences like physics, astronomy, chemistry, and biology [8] [9]. In the medical field, WSDs aid interdepartmental communication [10], optimize surgery room scheduling, and improve team transitions [11] [12]. They are also used for scheduling activities, such as conference organization [13], and support collaborative design tasks [14], including architectural design reviews [15].

WSDs are also referred to as vertical Large Interactive Displays (LIDs) or Large High-Resolution Displays (LHRDs). However, the notion of 'large' is typically not precisely defined and can be subjective [16]. Belkacem et al. defined an LHRD as a display that "*creates a coherent physical view space that is at least of the size of the human body and exhibits a significantly higher resolution than a conventional display*" [17]. According to Chen et al., WSDs improve user performance and satisfaction for tasks, such as model design, analysis, and visual data mining [18]. But, these new ways of viewing, collaborating and interacting differ from desktop and smartphone applications [16], because of their size, their resolution, the collaboration they foster, and the so-called *natural* interactions they enable, mainly through touch and gestures [19]. WSDs vary in terms of visualization technology, display setup (size, orientation), interaction modality, application objectives (productivity, entertainment, social interaction, games, and advertising) and location (city, office, education, conference, third place and cultural site) [19].

Based on previous work, we enumerate nine challenges raised by WSDs, each needing further research [16] [17] [20]: 1) *More interactions*: "natural" interactions with a mix of interaction methods [16] [20]. 2) *More users*: and improvements regarding how they collaborate [16] [17] [20]. 3) *More space (around)*: users' movements and interactions in the space and the management of the inherent fatigue [17] [20]. 4) *Different usage durations*: for example, collaborative decision-making tasks require long sessions spread over time [20], but for public displays, the duration of use is often very short [16]. 5) *More complex content*: complex data representations, large quantities of complex data needed by experts [16] [17] [20]. 6) *More devices*: WSDs are often composed of several displays, but are also often supplemented by other devices (e.g., supplementary displays or sensors) [16] [17] [20]. 7) *Variable screen space and pixel count*: displaying more data simultaneously requires content to be designed differently to distinguish between and manage the visible, accessible and useful parts of the display [16] [17] [20]. 8) *Support for designers*: tools or methods for designing and testing are needed, as well as guidelines and best practices [16] [20], and 9) *Other concerns* arise like accessibility, compatibility, and portability from one WSD to another, and workflow management [20].

Supporting designers with the right tools and methods to design applications for WSDs is hence an open challenge. Similar questions have also been raised regarding the design of data visualization interfaces for WSDs, specifically [17] [21]. Overall, the challenge is three-fold: the difficulty of scaling visual elements, e.g., text, the limited availability of design software, and the lack of widely adopted design guidelines.

In this paper, we address the *designer support* challenge, i.e., the need for design and testing tools and methods [16] [20]. We look into the design of a UI prototype in WSD environments at 1:1 scale and seek to assess the suitability of existing design tools for this purpose. By 'UI prototyping', we mean the prototyping of the interface, functionalities, screen layouts and behaviors at the mid-fidelity level. We focus on using popular UI design tools, such as Figma [22] and Miro [23], to prototype UIs in WSD environments at 1:1 scale. We have no conflict of interest with any of them.

In the rest of this paper, Section II presents related work on methods and tools for prototyping for WSDs. Our research approach is detailed in Section III. Section IV describes a first user study using Figma to design for WSD at 1:1 scale. This allowed us to identify the required features to support design at 1:1 scale for WSDs. Section V compares existing design tools based on these features. Section VI presents a second user study, diving deeper into the use of another design tool (Miro) with different WSD setups and interaction means. Our observations are discussed in Section VII and used to draw implications for design, and for tooling improvements. The section also discusses the limitations of both user studies. Finally, Section VIII holds a general conclusion.

II. Related work

Many tools and methods have been proposed to support design for WSDs. Below we discuss paper prototyping, prototype development and mixed mockup techniques. We also cover briefly interaction techniques used in WSD environments.

A. Paper prototyping

Paper prototyping is a popular, validated and simple method of mocking up systems before programming [24]. It allows designers to explore, communicate and evaluate early interface designs with end-users or within the design team. A designer typically plays the role of the computer to simulate the behavior of the system by changing the pieces of paper shown to the participants. Numerous studies have used paper prototyping to design applications on a WSD (e.g., [13] [25] [26] [27]).

Bailey et al. used this method to prototype a multi-device environments (MDE) involving personal devices (e.g., computers or tablet), and a wide screen to share data and views [25]. They observed that A4 paper lacks accuracy for large displays due to scaling issues. Indeed, the size hampers reading text at a distance. WSDs require sheets of paper larger than A4, and text size should be adapted to the screen size. However, paper prototyping does not allow checking whether the user can quickly and easily detect where information is displayed and whether changes in the displayed content would be noticed, as the user can follow the facilitator's gaze direction and movements and see where she places the pieces of paper.

Paper prototyping can also be used to define the screen arrangements to compose the WSD [28].

However, paper prototyping is mainly used for UI prototyping on WSDs.

B. Functional prototyping

Mid-fidelity prototype development is also a common practice [29]. Indeed, in the following papers, for example, there are no indication on how the applications were designed, but a developed prototype is used for the studies within them [4] [6] [7] [14] [30] [31].

Some systems have been developed to support prototyping, among others, for WSDs. For instance, *DEBORAh*, is a front-end web-based software layer that supports the orchestration of interactive spaces combining multiple devices and screens, including WSDs [32]. Additionally, *jBricks*, a Java-based toolkit, enables the exploratory prototyping of interaction techniques and rapid development of post-WIMP applications for WSDs, particularly for tiled-displays [33].

C. Mixed mockup techniques

Finally, mixed mockup techniques, e.g., *Mini-studio* [34], consist in using a physical paper model of the system augmented with projected content. They are mainly used to prototype ubiquitous computing systems, but can also be used to mock up WSDs. *SketchStudio* is another example, which combines 2D devices with 3D characters, resulting in a 2.5D animated scenario design tool for rapid prototyping of systems involving multiple users and multiple components or devices [35]. Such methods and tools offer the advantage of allowing the interactions around the WSDs to be tested. However, they are not accurate enough for a prototype of the screen layout and content, especially in contexts where large amounts of data and high resolution are required [17].

Overall, the prototyping method is frequently used for designing WSD applications, but the way the design was conceived is usually not described. Among the few occurrences where the design process is documented, paper prototyping is the most commonly used method. We did not find any studies covering the design of a UI prototype on a WSD at 1:1 scale.

D. Interaction techniques for WSD environments

Various interaction modalities have been used with WSDs, ranging from classic mouse and keyboard to more advanced types of interaction leveraging touch, gaze, mid-air gestures, proxemics, handheld and wearable devices, and tangibles [17].

Mouse and keyboard interaction is readily available in virtually all interface design tools used in common desktop environments. Users may work from a distance while being seated. The physical keyboard is very convenient for text entry, but text may lack legibility at a distance. Virtual keyboards have also been studied in the context of mobile devices [36], horizontal tabletops [37] and virtual environments [38]. They can be spawned on vertical display surfaces too, but hardly any work investigates their usability in this context. While mouse interaction is faster and more accurate than touch [39], the mouse cursor is easily lost and clutching may be tedious when navigating across very large scenes.

The cursor can also be attached to other pointing devices, be it a touchpad or a handheld device such as remote controllers, or flysticks, or head-mounted displays through eye tracking. Handheld display devices, such as smartphones or tablets, can also be turned into pointing devices by attaching sensors to them, or by using them as touchpads to interact with the WSD.

Direct touch can also be used on touch-enabled WSDs. On the one hand, touch is an intuitive interaction modality that exhibits high user performance and acceptance scores [40]. On the other hand, users often have to move across the WSD, or step up close or away from the WSD, to look at details, or get the big picture, respectively [41]. They may also have a hard time reaching the upper or the lower part of the WSD [42].

Overall, many advanced interaction techniques may generate fatigue or muscle strain, e.g., gorilla arm [41], when used for a long time. They also differ significantly from widespread mouse and keyboard interaction, in terms of accuracy, speed, and their hedonic value [40], and may be less suited for certain tasks such as text entry. A more detailed discussion of these interaction techniques for WSDs can be found in the literature [17].

In this work, we evaluate the suitability of popular interface design tools, originally meant for the desktop, to design UIs for WSD environments at the 1:1 scale. Most design tools were not developed for WSD, nor optimized for advanced interaction modalities available in WSD environments.

III. Research approach

As noted by Lischke et al., when it comes to WSDs, it "*is often not possible to prototype in the original size*" [16]. Unlike using a desktop computer to design UIs for WSDs, prototyping in real size directly onto the targeted display could reduce complexity, give a sense of scale, and ensure that the target resolution is correctly achieved and exploited. It could also allow designers to check that the UI is visible at all distances and from all angles [17] [20]. Hence, our interest in prototyping on a WSD is based on supporting the design in real size.

A. Research questions

As part of understanding how to support the design in 1:1 scale for WSDs, we focus on three main research questions:

RQ1: Can a desktop optimized tool be used in a WSD environment to design at 1:1 scale?

RQ2: What would be the best interaction modality during the design at 1:1 scale on a WSD ? And what are the main issues raised by each interaction modality?

RQ3: What are the main features needed to support the design at 1:1 scale on a WSD ?

This research aims to extract initial guidelines for designing at 1:1 scale *on* and *for* WSDs. To answer these questions, we have set up two exploratory user studies and performed a comparative analysis of existing design tools. The first user study involved one participant who consecutively interacted through three distinct interaction methods (a keyboard and its embedded touchpad, direct touch on the WSD, or a connected tablet) and with two WSDs to understand whether and how a desktop optimized tool, Figma, could be used in a WSD environment to design at 1:1 scale. The outcomes of the first user study pointed us to important features, as a basis for doing a technology review and comparative analysis, and selecting a better tool, i.e., Miro, for the next user study. In the second user study, we observed the use of Miro by two participants designing at 1:1 scale on two WSDs with the same three interaction modalities.

B. Task

In both user studies, the task consisted in using a desktop optimized software (Figma or Miro) to reproduce a previously developed UI [43] and adapted for both WSDs as shown in Figure 1, more details can be found online [44] [45]. Before the test session, the participants discovered the design tool on a desktop computer for two hours, and the mock-up they had to reproduce.

This UI was chosen because it comprises a variety of UI elements (text, sliders, graph, a social media feed). Reproducing an existing UI ensures that it is feasible, well adapted to the WSD environment, and allows the observation to be focused on the design software manipulation rather than the process of creating a new design.

C. System

The system consisted of a touch-enabled WSD displaying the design tool (either Figma or Miro) in a web browser window (Google Chrome [46]) in full-screen mode.

Two WSDs were used. First, WSD-IA (curved, diameter: 3.64m, height: 2m, composed of twelve 4K screens in portrait mode, among which eight are touch-enabled using infrared frames. The setup also included a height-adjustable table and a keyboard/touchpad, as shown in Figure 2.

Then, WSD-VW (flat, width: 7m, height: 2m, total resolution $13,152 \times 3,872$ pixels, composed of 24 HD screens with infrared frames enabling touch (see Figure 3). This WSD is located in a room containing three high tables (fixed-height) placed at each end of the WSD, with two mobile extended-height chairs each. The room also contains a large standard height table facing the middle of the WSD (about 3 meters away), and the display itself includes a virtual touch keyboard that appears at the bottom center of the WSD.

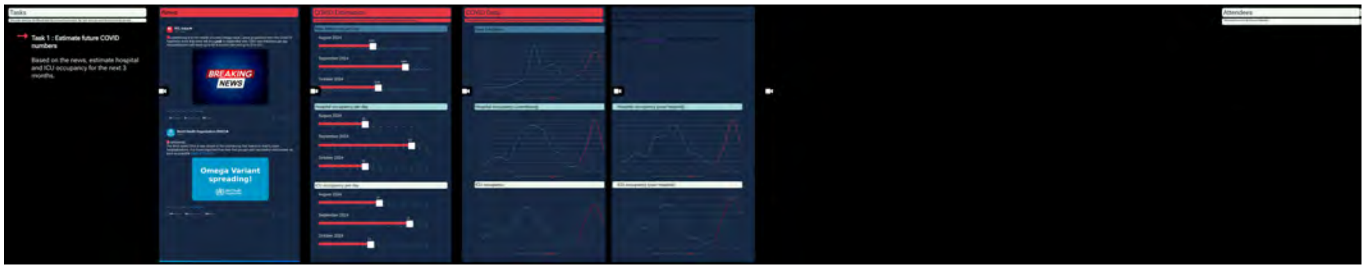


Figure 1. Screenshot of the prototype to replicate, for more information see [43] and [44] [45].

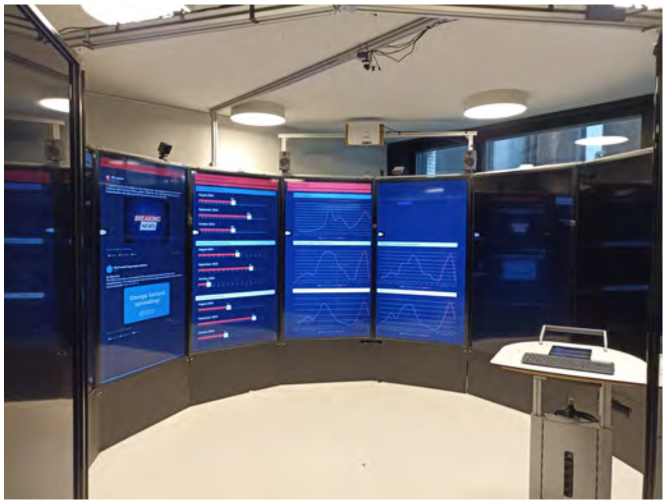


Figure 2. WSD-IA displaying the interface used for the task.

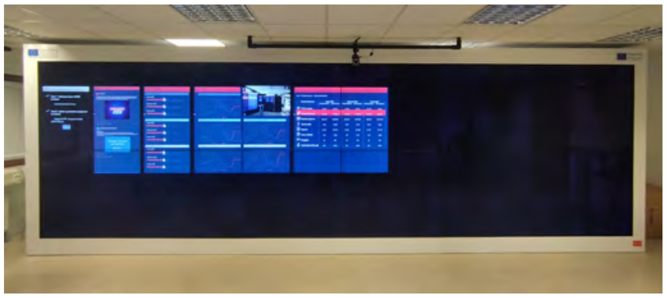


Figure 3. WSD-VW displaying a version of the interface used for the task.

D. Observation and analysis methods

Video cameras and microphones were used to record the sessions. For WSD-IA, three video cameras were used, one top camera placed in the middle of the room and attached to the roof, one in the center and attached to the top of the WSD (in front of the entrance), and one on the back, attached at the top of the entrance screens (the two screens, which can be rotated); see Figure 4 and Figure 5 for clarifications. For WSD-VW, two cameras were placed in the back, in front of both ends of the display for the first user study. They were completed by a third one at the middle back for the second user study. At the end of each session, the participant was invited to discuss and debrief with the facilitator. We analyzed thematically the comments made by the participants, both based on the debriefings and during the sessions themselves, to identify encountered issues. We additionally observed the moves and strategies the participants relied on.

IV. Prototyping at 1:1 scale on WSD with Figma

During the first exploratory user study, we used Figma, a mid-fidelity web-based prototyping tool for designing, collaborating, prototyping and transmitting content [22]. We chose it for its popularity [47] [48] and its availability as a ready-to-use web-based solution. Figma was tested on a WSD by one designer under several experimental conditions: two WSD settings (WSD-IA, and WSD-VW) with different arrangements and surroundings, and three different interaction methods to reproduce the interface: a wireless keyboard with a touchpad, direct touch on the WSD, and a synchronized tablet.

The main research question addressed by this first user study is: **can Figma**, as a desktop optimized tool, **be used in a WSD environment to design at 1:1 scale?**

In other words, we want to identify the challenges and opportunities raised by using an existing design tool to prototype the UI of an application built for a WSD, directly on the WSD.

A. Protocol

The **participant** is an expert in UI design and has participated in the design of several UIs for WSDs, but had never used Figma before. She was free to stop the session whenever she wanted (e.g., when it became too difficult) or after having finished reproducing the design. Since this first user study aimed to verify the feasibility of using Figma under these conditions before carrying out more in-depth studies, a single user was deemed sufficient. Conversation guide and detailed protocol are available in supplementary material online [44].

Figure 4 illustrates the interaction methods tested to interact with Figma to complete the **task** as described in Section III: respectively, a wireless keyboard with a touchpad, direct touch on the WSD, and a tablet synchronized with the WSD.

B. Results

In general, the participant appreciated the ability to design at a 1:1 scale, regardless of the interaction method and the WSD used, with the main advantage of being able to see in real time the final rendering on the target screen. Several issues are related to the lack of familiarity with Figma, as the use of widgets, components, and plugins seemed complicated, and were not used effectively by the participant. Also, some

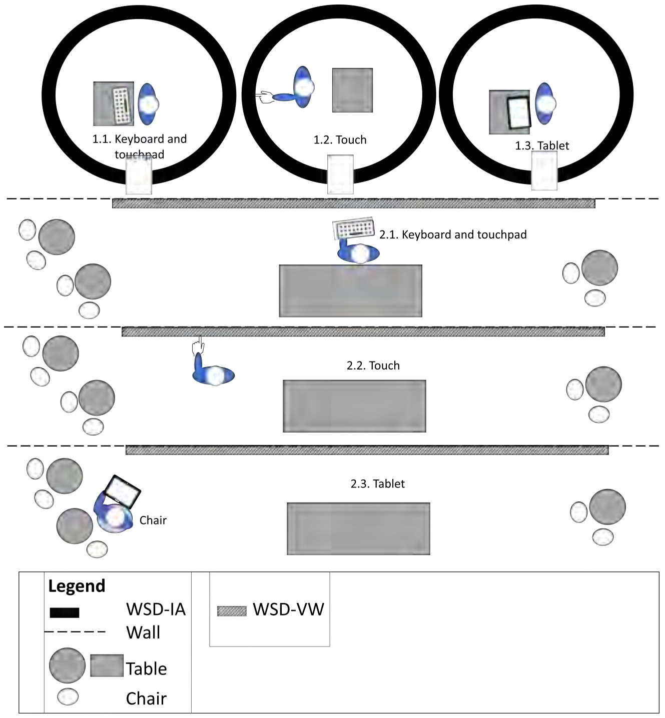


Figure 4. Experimental settings: at the top, the curved WSD-IA, at the bottom, the flat WSD-VW. The spatial relationships depicted here are illustrative and may not correspond to real-world dimensions.

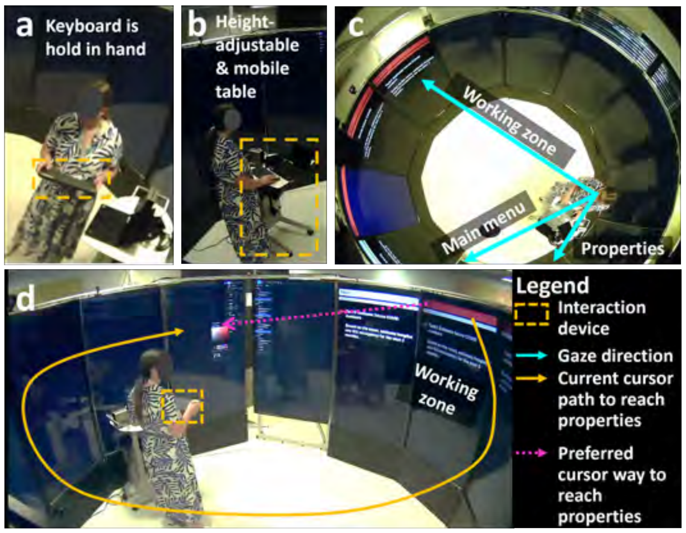

a Keyboard is hold in hand
b Height-adjustable & mobile table
c
Working zone
Main menu
Properties
d
Working zone
Legend
Interaction device
Gaze direction
Current cursor path to reach properties
Preferred cursor way to reach properties

Figure 5. Observations made when interacting with Figma and keyboard+touchpad. a) At first, the participant held the keyboard. b) Use of a table to put down the keyboard. c) Lots of head rotations to see all the important areas. d) WSD-IA does not allow you to cross directly from the left screen to the right screen.

problems are due to or emphasized by the circular nature of WSD-IA.

1) General observation about Figma: The configuration of the **Figma environment** was not always suited for WSDs. For instance, the properties of a selected object are placed on the right-hand side of the display (see Figure 6.a), the main menu is placed at the very top (see Figure 6.b) and dialogue boxes open in the middle of the display. The user must also move the cursor across the entire display or physically walk to the desired location to modify, e.g., element properties (see Figure 5.d), which is tiring on the long run. Below, we discuss in more detail the issues related to each interaction method.

2) Interacting with a wireless keyboard with a touchpad: The session lasted one hour for WSD-IA and ten minutes for WSD-VW. On both WSDs, the participant would sometimes have a hard time finding the cursor on the large display.

Concerning the *WSD-IA*, the menu and items list were displayed on the leftmost screen (one of the two screens also used as a door), and the selected item's properties on the rightmost screen (the other screen forming the side by side door). To avoid turning her head from the extreme left to the extreme right too often, the participant closed the doors to look at them both at the same time. She also placed herself to create an angle that allowed her to see the menu, the properties of the selected object, and the work area at a glance (see Figure 5.c). As the session was short, and all UI elements were tightly grouped on the left, the position was acceptable. But the user could not maintain this position while working in the middle. In this configuration (menu on the left and properties on the right), the participant turned her head and body a lot, which could possibly be painful and exhausting. At first, the participant carried the interaction device, see Figure 5.a. After 15 minutes, she felt tired and placed it on a table, see Figure 5.b. Another problem was the impossibility to move the cursor directly from the WSD's leftmost side to its rightmost side. The participant had to move the cursor all the way around the WSD, which is tiring, despite the physical proximity of both sides of the WSD due to its circular arrangement (see Figure 5.d). To avoid turning her head too much, the participant did not follow the cursor with her eyes when it was behind her back.

In the *WSD-VW* condition, the font size of the Figma interface was problematic. The flat WSD-VW was too wide to read text labels when standing at the opposite side of the display. Hence, to modify a property's value, the user had to walk frequently across the WSD-VW to the properties area, where she rested on a table next to it. Then she leaned on the middle table for comfort and stayed at a certain distance from WSD-VW to get an overview. The fatigue due to walking around, eye strain due to the text size, and carrying the keyboard led the user to stop the test after ten minutes. Because the WSD-IA is twice as large as the WSD-VW in terms of horizontal resolution, the virtual navigation felt more painful in WSD-IA. This is also the case because the cursor and its progress were always visible in WSD-VW, and the session was shorter.

3) Interacting using direct touch on the WSD: The session with WSD-IA lasted twenty minutes, and the session with WSD-VW was interrupted after ten minutes.

In the *WSD-IA* condition, to manage physical fatigue (neck strain and gorilla arm syndrome [49]), the user tried to work

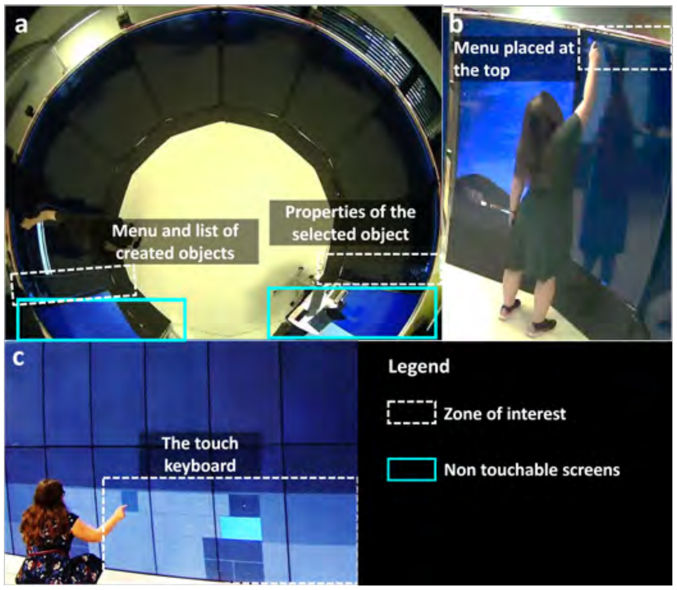


Figure 6. Observations done when interacting with Figma and touch. a) The touch space is occupied by the Figma interface on the left (list of created objects) and the right (properties). b) The menu is too high. c) WSD-VW's touch keyboard is not comfortable; the user had to crouch.

at a lower scale by zooming in on the work area without minimizing the Figma window. Although the menu remained too high and objects' properties too far away, objects could be moved with smaller movements and were better placed relative to the user's field of view, generating less neck pain. Hence, the advantage of working at 1:1 scale was temporarily lost. Only the middle eight screens of WSD-IA support touch, so we resized the browser window used for Figma to fit within these screens. As Figma's interface elements (list of objects and properties) take up space themselves at both ends of the window, the design space was further reduced, by about one extra screen for interface elements from both ends combined. Hence, the total design space available to the participant was reduced compared to the actual interface they had to reproduce. Consequently, the alignment with the display tiles could not be maintained, as shown in Figure 6.a. When using touch, the participant had trouble moving an object across tiles. The wireless keyboard was used to input text or values and was either held by the participant or placed on the table.

On *WSD-VW*, the properties panel was too far away from the work area, but unlike WSD-IA, when a property was changed on WSD-VW, the result was not visible from the user's position. So, she stepped back to check, e.g., whether the font size is large enough. The top menu was out of reach, and WSD-VW's virtual keyboard was not suitable for entering more than one word due to its design (position at the bottom and large size, see Figure 6.c). After ten minutes of use, the participant complained from the gorilla arm syndrome.

4) Interacting on a synchronized tablet: The same Figma project was loaded onto the tablet and onto the WSD. The UI elements were created, moved and adjusted on the tablet.

We observed that the participant mainly looked at the tablet

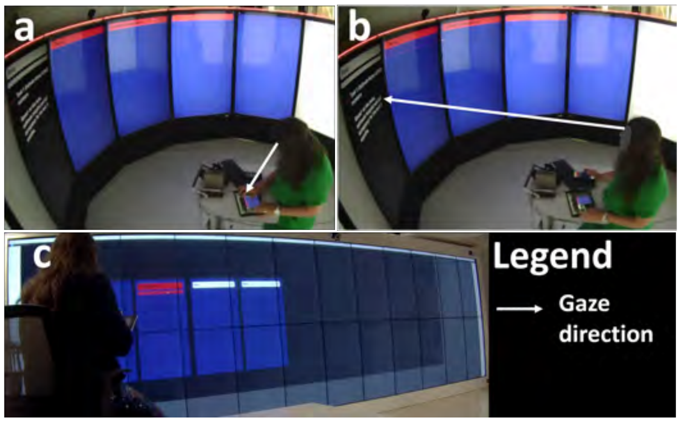


Figure 7. Observations done when interacting with Figma and a tablet. a) The participant modified the prototype on the tablet. b) Then, the participant checked the result on the WSD. c) The participant sat on a chair.

to add UI elements, move them around and set parameters, see Figure 7.a. Then, the participant looked at the WSD to check, e.g., the position and size of the UI elements, the readability of text, and colors, see Figure 7.b. A main issue was the impossibility to select several UI elements at the same time on the tablet, as they are superimposed. The session with WSD-IA lasted ninety minutes, whereas the session with WSD-VW was interrupted after twenty minutes.

On *WSD-IA*, the user had difficulties to position the UI prototype on the WSD correctly. Although the additions and changes to UI elements were synchronized between the tablet and the WSD, the viewports were independent, so positioning the UI had to be done from the WSD directly. This led to the use of an extra wireless keyboard equipped with a touchpad. This happened when the participant closed the project and reopened it. The designed UI was centered horizontally and vertically on the WSD. So, the participant wanted to properly reposition the designed UI to continue the design. The participant also used the touchpad to select a group of UI elements to save them as a new reusable UI element. She placed the tablet on the height-adjustable and mobile table. She felt that WSD-IA and tablet configuration was the most comfortable.

On *WSD-VW*, the participant sat down and placed the tablet on the table, see Figure 7.c. But, as the table was not well positioned and too heavy to be moved, she preferred to hold the tablet in her hand, which was tiresome.

C. Preliminary observations

The duration of the test sessions varied widely, from ten to 90 minutes. The most comfortable condition seems to be WSD-IA with a tablet and a height-adjustable and mobile table. But the problem of multiple selection and correct positioning of the prototype on the WSD needs to be solved. Overall, the main issues were: (I1) physical fatigue, (I2) reachability of Figma elements, (I3) readability of the Figma interface, (I4) the vast interaction surface, (I5) when a project is reopened, objects are moved to the middle, and

(I6) the partial occlusion of the WSD by Figma's UI elements, which is not a perfect 1:1 scale.

To tackle these issues, we outline several design solutions. (I1) could be reduced by managing the physical environment and providing a height-adjustable and movable table to place the interacting device, chairs, and by supporting interaction at a distance. (I2) could be improved by offering floating context menus and value input fields, by opening dialogue boxes close to the work area or by using a smaller interaction device as a tablet or a laptop. For (I3) and (I4), the size of the design tool interface elements should be adapted. For (I4), a bigger cursor should be used as well as pointing facilitation techniques [50] to reach the opposite end of the WSD. (I5) could be solved by fixing the created objects in their positions and reloading them in exactly the same position. To achieve 1:1 scale (I6), the design tool interface should be hidable or movable.

Overall, issues (I2), (I3), (I5) and (I6) show that Figma may not be adapted to prototype at 1:1 scale on WSDs, and that another tool incorporating the outlined design solutions, might be better suited.

V. Comparative analysis of existing design tools

This first user study enabled us to identify the problems with using Figma as a 1:1 scale design tool for WSDs. Based on these results, we then set up a list of required features for such a tool. Below, we first list these features and then analyze 19 different design tools, meant for UI design for computers and smartphones, taking into account the listed features.

Based on our observations, we identified the following main features of tools suited for designing at 1:1 scale on WSD:

- The main menu must be reachable from the work area (not at the top or bottom) [related to I1, I2, I4].
- The properties of the selected object must be reachable from the work area [related to I1, I2, I4].
- Pop-ups must open near the work area [related to I1, I2, I4].
- The font size of the design tool UI elements must be large enough (or adjustable) [related to I3].
- It must be possible to conceal or move the design tool interface elements, i.e., menu, list of created objects, and properties of the selected object [related to I6].
- It must be possible to interact from a synchronized tablet [needed for our user study].
- When reopening a previous project, it must preserve the positions of the created interface elements, as they were previously. [related to I5]
- The workspace size must be large or adaptable to the typically high resolution of WSDs.

Based on this features list, we analyzed 19 design tools. We performed a search using the engine Google with the keywords "design tool", "mockup tool" and "UI design tool" between July and December 2024. We collected the design tools from direct results or from online articles about such tools [48] [51]. We then eliminated the tools that did not support the creation of mock-ups, and the ones that could not be tested and used for free. In the end, we selected 19 tools, which were analyzed against the listed features. Many tools set fixed positions for their menus, and for object properties panels. None of the tools in the selection enabled users to conceal or move the design tool's interface elements.

As our WSDs run on a Linux operating system, we also had to narrow down our list to multiplatform tools, which indirectly led us to mainly consider web-based solutions.

Table II lists the design tools and our analysis of their characteristics against the desirable features. Concerning the maximum surface size of the workspace, this information was rather complex and at times impossible to find for some design tools, so some figures listed in the table might not be accurate. As most of the examined tools provide a web-based version, thus giving access to the zoom functionality of the web browser, we did not check the font size of the design tool's interface.

Based on the features they offer, we pre-selected three multi-platform design tools (Bubble, inVision, and Miro) to test them more in-depth on one WSD (WSD-IA), with the three interaction methods used in the user study, to ensure their usability for the second user study. With inVision, we ran into issues when using touch on our WSD, so we had to reject it. Additionally, we noticed that the solution was discontinued shortly after our tests. As for Bubble, we experienced some synchronization latency when using the tablet, which is why we discarded it as well. We therefore chose Miro as design tool, because it satisfied most of the requirements, including running in a web browser, showing object properties in a context menu, reopening a project at the same position, retractable interface by using the preview mode, and offering the option to open the same project on a tablet as on the WSD.

VI. Prototyping at 1:1 scale on WSD with Miro

Based on the findings from the first user study and the feature matrix, we ran a second user study with some modifications in the protocol compared to the first study, as explained below.

The research questions are adapted from the first study. RQ1: **Can an existing design tool**, that was conceived for desktops, **be used in a WSD environment to design at 1:1 scale?** And RQ2: **What are the main guidelines to support design at 1:1 scale in a WSD environment?**

A. Protocol

Miro was tested as a mockup tool for WSDs by two designers under several conditions: two WSD settings (WSD-IA, and WSD-VW), and three different interaction methods: (1) a wireless keyboard with a touchpad, (2) touch on the WSD completed by a wireless keyboard for entering textual and numeric values, and (3) a synchronized tablet.

Sessions were limited to one hour. But, participants were free to stop the session earlier, if they felt too tired, or uncomfortable, or if they completed the task.

The two **participants** were respectively an expert in UI design (p2) and in visualization design (p1). They both participated in the design of several UIs for WSDs, although (p2) is

TABLE I. Main issues and possible solutions.

	Issues	Keyboard/touchpad		Touch		Tablet		Possible solutions
		WSD-IA	WSD-VW	WSD-IA	WSD-VW	WSD-IA	WSD-VW	
I1	Physical fatigue	Yes frequent head and torso rotations	Yes hand-held, much walking	Yes Menu too high, much walking	Yes Menu too high, much walking, touch keyboard too low	No placed on a table	Yes hand-held	A height-adjustable movable table on which to place the interaction device, a chair for sessions longer than 90 minutes. Interacting at a distance.
I2	Figma's elements reachability	Yes Properties too far from the working zone	Yes Properties too far from the working zone	Yes Properties too far from the working zone	Yes Properties too far from the working zone	No	No	Floating context menus and value input fields near the working zone, allow the participant to move across the display without scrolling, open dialogue boxes near the working zone, and use a smaller interaction device.
I3	Readability	No	Yes UI element not visible when modifying its parameters. Figma interface not readable.	No	Yes UI element not visible when modifying its parameters	No	No	Adapt the size of the Figma elements, use a curved WSD instead.
I4	Huge interaction surface	Yes It takes a long time to reach the extremities.	Yes It takes a long time to reach the extremities.	No	No	No	No	Adapt the size of the Figma elements, use a curved WSD instead. Bigger cursor, pointer acceleration to reach the opposite side of the display.
I5	Not fixed position of UI elements	Yes	Yes	Yes When reopening the project.	Yes	Yes	Yes	Fix the UI elements in their position and restore their position after reloading the project. The Figma design space should match the targeted WSD.
I6	1:1 scale not perfect	No	No	Yes Part of the necessary design surface masked by Figma elements.	No	No	No	Possibility to conceal or move the Figma interface elements such as the menus.

TABLE II. Features matrix of the UI design tools. Cells in gray highlight instances where features or requirements we identified are supported by the corresponding design tool. NA: not applicable. * No source for inVision as the design tool was stopped after our review. Adobe XD is now in maintenance mode. ** E.g., delete.

Name	**Max surface size**	**Supported platforms**	**Main menu position**	**Object properties position**	**Dialog boxes position**	**Reopens at the same position**	**Retractable interface**	**Synchronization with a tablet**	**Selected**
Adobe XD* [52]	50,000x50,000 px	macOS, Windows, Android, iOS, browsers (Chrome, Firefox, Edge, Safari)	fixed on the top and left	fixed on the right	middle	unknown	unknown	unknown	no
Axure RP [53]	20,000 px	Windows, macOS, Android, iOS	fixed on the top	fixed on the right	middle	unknown	unknown	unknown	no
Balsamiq [54]	20,000x20,000 px	Windows, macOS, browsers	fixed on the top	fixed on the right	middle	yes	partially + preview mode	possible with the cloud version	no
Bubble [55]	max width 5,000 px	browsers	fixed on the left	floating and movable	unknown	no	partially + preview mode	Possible with the opening of the same project, but lag	yes
Canva [56]	8000 x 3125 px	browsers, Windows, macOS, iOS, Android	fixed on the left	fixed on the top some actions attached to the selected object**)	middle	no	partially + preview mode	Possible with the cloud version	no
Excalidraw [57]	unknown	browsers	fixed on the top	fixed on the left	unknown	yes	partially	using collaboration feature	no
Figma [22]	no limit	browsers	fixed on the top	fixed on the right	middle	no	partially + preview mode	Possible with the opening of the same project or using collaboration feature	no
Framer [58]	max width 1280px	browsers, macOS, Windows	fixed on the top and left	fixed on the right	middle	no	preview mode	Possible with the cloud version,	no
InVision*	unknown	browsers	fixed on the bottom middle	attached to the selected object	top left	no	unknown	Possible with the opening of the same project,	yes
JustInMind [59]	unknown	Windows, macOS	fixed on the top	fixed on the right	middle	unknown	unknown	using collaboration feature	no
Miro [23]	unknown	browsers, iOS, Android, macOS, Windows	fixed on the left	attached to the selected object	middle	yes	preview mode	Possible with the opening of the same project, or using collaboration feature	yes
MockFlow [60]	unknown	browser, macOS, Windows	fixed on the left	fixed on the left	unknown	no	no	unknown	no
Mockplus [61]	no limit	browsers, Android, iOS, Windows, macOS	fixed on the top and left	fixed on the right	middle	yes	preview mode	using collaboration feature (only visualization, no modification)	no
Penpot [62]	no limit	browsers	fixed on the top	fixed on the right	middle	yes	yes + preview mode	using collaboration feature	no
Proto.io [63]	unknown	browsers (Chrome, Safari, Firefox)	fixed on the right	fixed on the right some actions attached to the selected object	middle	no	preview mode	unknown	no
ProtoPie [64]	unknown	Windows	fixed on the top	fixed on the right	middle	unknown	unknown	unknown	no
Sketch [65]	unknown	macOS	fixed on the top	fixed on the right	unknown	unknown	unknown	unknown	no
UXPin [66]	25000x25000 px	browsers, macOS, Windows	fixed on the left	fixed on the right	middle	yes	preview mode	Possible with the opening of the same project, but lag	no
Webflow [67]	max width 10,000 px	browsers	fixed on the left	fixed on the right	middle	no	preview mode	unknown	no

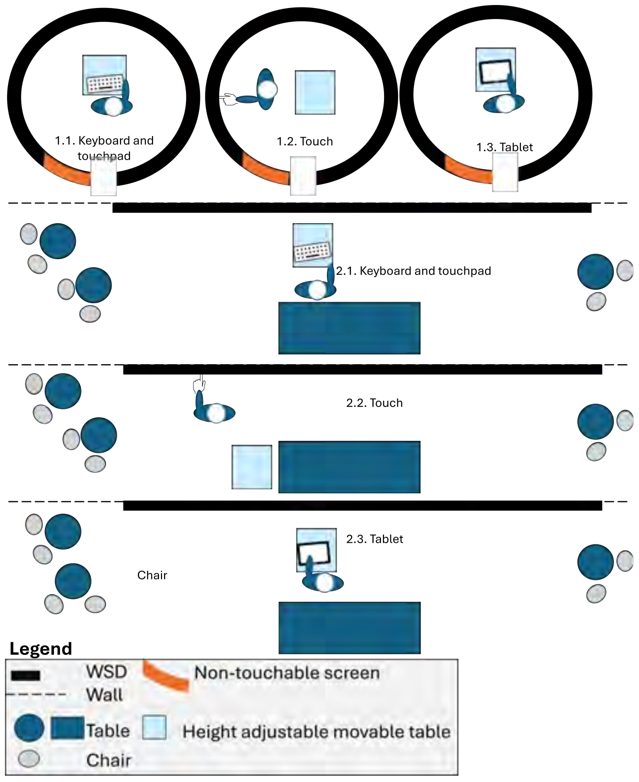


Figure 8. Experimental settings for the second study: on the top the curved WSD-IA, on the bottom the flat WSD-VW. The spatial relationships depicted in this figure are illustrative and may not correspond to real-world dimensions.

more experienced in UI design per se. They both had already used Miro beforehand as a collaborative mind mapping tool, but not for UI design, with p2 who was competent using Miro and p1 self-declaring as more novice.

Like in the first study, the **system** used for this second user study consists of a touch-enabled WSD displaying the design tool, Miro in this case, through the Chrome web browser. The zoom level in the browser was set to 200%, for the main menu items (located on the left and vertically centered) to be wide enough to be usable. We used the same two WSDs as previously: WSD-IA (curved) and WSD-VW (flat).

Learning from the first user study, some adaptations were made on both WSDs. As for WSD-IA, additional infrared touch frames were put in place, increasing the number of touch enabled screens to eleven. As text input with the virtual keyboard on WSD-VW was quite cumbersome, and to ensure a fair comparison between the conditions between both WSDs, participants were also provided with a physical keyboard for the touch condition, but its use was limited to text input. We have also provided a height-adjustable movable table with WSD-VW, to allow users to put down the interaction device anywhere in the room (see Figure 8).

The **task** consisted in using Miro to replicate the same UI as in the first study (see Figure 1). The participants first discovered Miro on a desktop computer for two hours, with the goal of reproducing another small UI similar to the main one in terms of components to be drawn. Detailed task, protocol, and discussion guide are available in supplementary material [45].

TABLE III. Order of test conditions for the second user study using Miro.

Condition ID	WSD	Interaction method
Participant 1 (p1)		
IA-Keyboard-p1	WSD-IA	Keyboard
IA-Tablet-p1	WSD-IA	Tablet
IA-Touch-p1	WSD-IA	Touch
VW-Touch-p1	WSD-VW	Touch
VW-Keyboard-p1	WSD-VW	Keyboard
VW-Tablet-p1	WSD-VW	Tablet
Participant 2 (p2)		
IA-Tablet-p2	WSD-IA	Tablet
IA-Touch-p2	WSD-IA	Touch
IA-Keyboard-p2	WSD-IA	Keyboard
VW-Keyboard-p2	WSD-VW	Keyboard
VW-Touch-p2	WSD-VW	Touch
VW-Tablet-p2	WSD-VW	Tablet

We used similar interaction methods as before, i.e., a wireless keyboard with a touchpad, touch on the WSD with a keyboard, and a tablet synchronized with the WSD (see Figure 8). We changed the order in which participants used the interaction methods, as shown in Table III. For logistic reasons, the tests were run on WSD-IA first, and then on WSD-VW.

At the end of each session, the facilitator debriefed with the participants to discuss their feedback and suggestions for improvement. The debriefing collected spontaneous open-ended feedback first, and then leveraged sentence completion [68] regarding first impression, the utility, usability, and user experience of the test condition, as well as its capacity to meet the needs and desired improvements. See the conversion guide and user study guide provided in supplementary material [45].

B. Results

Designing in real size was appreciated by participants, it enhances the understanding of object dimensions, placement, and the use of space ("*the model on a large scale gives more precise impression of the objects' size, of their position and the use of space.*" p1, "It is great to see the result directly in real size on the targeted display." p2).

1) General observations:

a) General observation about Miro: In Miro, the main menu that allows to create objects has a fixed position on the left side of the workspace and is centered vertically. The context menu for a selected object is displayed in the vicinity of the object, as shown in Figure 9. Nobody complained about the position of the two menus. The context menu allowed to modify properties and perform actions on the selected object, such as deleting or duplicating it, while the main menu on the middle-left side allowed the creation of objects ("*It is convenient that a part of the functionalities [object properties] are placed in the working zone.*" p2). The only exception to the above is for p2 in condition touch on WSD-IA ("*It's not working well.*" p2, "*It's not comfortable.*" p2). We however note that the mock-up to reproduce mainly used the leftmost parts of the WSD. Had UI elements from the mock-up been placed in a more central position (or even towards the right side of the WSD), it is likely that the position of the main menu

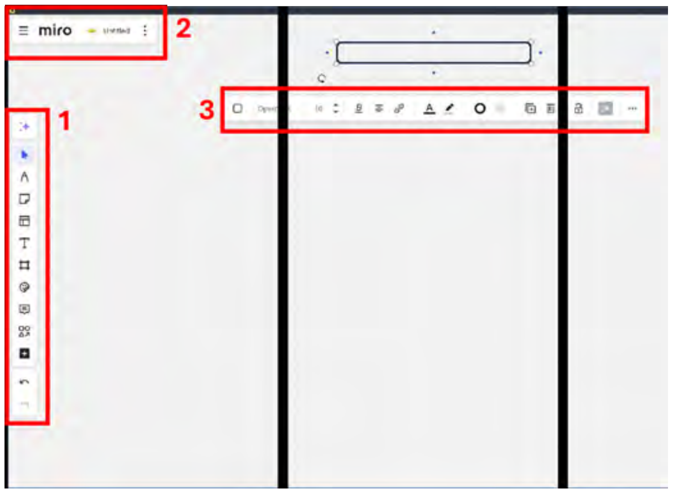


Figure 9. 1) shows the left menu used to create objects, 2) is the menu used to set some parameters for the workspace or go back to the list of projects in Miro, and 3) is the context menu of the selected object.

would have been more of an issue. The lack of complaints on the menu placement could also be due to the fact that the mock-up only contained a moderate number of objects that the participants must create. Another explanation could come from a strategy used by both participants to mainly rely on duplicating an existing object then completely modifying its properties, rather than creating a new object using the menu. This behavior could also be a reaction to the problematic placement of the menu. The context menu was used a lot, even though it sometimes stopped working for no apparent reason. The top menu, which allows users to set some parameters of the workspace, was only used to change the background color of the workspace.

The most critical aspect of Miro noted by both participants is that it does not provide a rich objects library. There are for example no chart elements ("*The available objects are very basic, so everything must be laboriously drawn by oneself.*" p1). Both participants complained that the minimum font size is blocked at 10. It was observed that, when an object was created (or duplicated) partially outside the current viewport (the portion of the workspace that is visible on the WSD), then Miro adapted the view to include that and all the objects ("*The zoom level changes when I copy/paste something out of the view, to put everything in view.*" p2). This disturbed the participants. During the session on WSD-VW with p1 and the touch condition, a modal popup window opened in the middle of the WSD, blocking all interactions. It took p1 a whole minute to notice it, which was frustrating. Both users liked the ease of changing the background color of the workspace.

b) Participants' strategy: The participants often used a strategy consisting in subdividing the mock-up in panels or zones, using a lot of copy and paste actions, and working on main structural elements before tackling the details, for all conditions. With both the touch and the keyboard conditions, p1 started by creating objects at the center level, to interact at comfortable height. He then panned the mock-up to move the objects up. P2 also used this strategy but only when interacting with touch ("*It is practical to be able to pan the mock-up, because I was able to work in the middle of the screen at the beginning, and then move everything up.*" p2).

c) Task duration: One-hour sessions were judged acceptable, but chairs were deemed mandatory for longer sessions.

2) Interacting with a wireless keyboard equipped with a touchpad: In WSD-IA, p1 left the keyboard on the table and turned his head and torso to see what he was doing, particularly when he created the last panel of the mock-up, which was behind his back, see Figure 10.a. Once, he turned around the table without moving it. In WSD-VW, p1 always left the keyboard on the table including when he moved (five times) to the extreme right to change the zoom level from the zoom menu (at the opposite end of the WSD, in the bottom right corner) or to create the last panel of the mock-up. In this case, p1 rolled the table until he was facing the zoom menu or the new work area (see Figure 10.b). P2 noted that she moved more using the keyboard with WSD-VW than with WSD-IA. In WSD-IA, p2 remained at the table and turned it to face the work area (see Figure 10.g and h). P2 mainly leaned over the table (see Figure 10.d), but she often held the keyboard on her arm during the last half of the session. P2 kept the keyboard in her hands most of the time when doing modifications like creating an object, moving or resizing it, changing its properties or zooming and panning. To enter text, p2 mostly put the keyboard on the table. She showed some signs of physical fatigue halfway through the session (movements with the neck, massages of the neck, see Figure 10.e. In WSD-VW, p2 moved the table with the keyboard on it to face the panel she was working on, and halfway through the session, sat down for 15 minutes (see Figure 10.f). But p2 also often stood up while holding the keyboard in her hands. Occasionally, p2 stepped back to get an overview.

In WSD-IA, p1 started by creating the left panel (Tasks), then he created the title and subtitle of each panel. He looked for some chart objects or chart icons. As he did not find any, he instead went on to focus on creating the sliders. In WSD-VW, p1 adopted another strategy. He first created one big blue rectangle to serve as background for all panels. Then, he created the other objects on top of it. This choice generated difficulties later on, when trying to select objects without selecting the background shape. He created the titles in the middle of the WSD, then panned the workspace to place them at the correct height. He built the interface panel by panel from the left to the right. He sometimes zoomed in to work on details (e.g., icons). He had some trouble grouping and ungrouping objects. In WSD-IA, p2 started to design objects directly where they were supposed to be placed. After creating the titles, p2 designed panel by panel from left to right, before coming back to the charts and sliders to add some details. At the end, she added blue boxes to create the background of each panel. In WSD-VW, p2 created the objects at a lower position than their final placement, then panned the workspace to adjust the height. After first creating the titles, she then designed the remaining objects panel by panel from left to right.

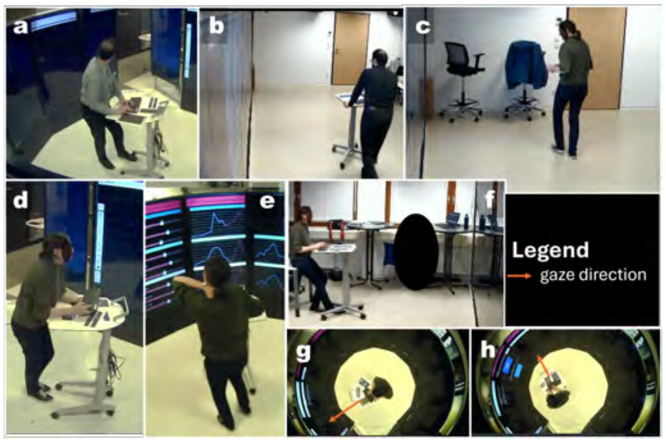


Figure 10. Using Miro with keyboard+touchpad. a) P1 turns his torso to look behind in WSD-IA. b) P1 moves the table with the keyboard on in WSD-VW. c) P2 carries the keyboard in her hands in WSD-VW. d) P2 leans on the table and tilts the keyboard with her hands in WSD-IA. e) P2 massages her neck in WSD-IA. f) P2 sits down in WSD-VW. g) and h) p2 turns the table to face the work area.

P2 used keyboard shortcuts a lot, to delete, copy, paste, but also to create new shapes, text boxes, or zoom in and out. P2 and p1 used keyboard arrow keys to place objects. This facilitated a precise placement of objects ("*I tried to be accurate with the position of the objects from the paper mock-up, taking into account the position relative to the various screens of the wall.*" p1). It also allowed participants to be more precise in the layout of the mock-up itself relative to the workspace. ("*With the keyboard, positioning the scene correctly is easier.*" p1). The keyboard was also appreciated for entering text. P2 also found it easier to draw graphs curves using the keyboard touchpad than with the tablet or in the touch condition.

The participants disliked the touchpad because it lacked precision ("*It was OK with the touchpad, but for smaller things it would be better with a mouse.*" p2). P1 complained about triggering actions inadvertently, such as an unintended pinch leading to a change of zoom, but unwanted panning, and selecting/deselecting actions also occurred ("*Of all the modalities tested, the keyboard was the most frustrating one, due to the touchpad's extreme sensitivity. As a result, this modality is difficult to use alone, it would be better to complement it with other modalities.*" p1). That's why participants asked for an option to change the zoom level independently of the panning. P1 and p2 in general prefer to use a mouse instead of the touchpad. It however remains unclear how a mouse would be dealt with when a participant walks around with the interactive device in hand, since a mouse might be difficult to use without being placed on the table. P2 also used some shortcuts to avoid using the touchpad, such as those enabling the creation of a new shape or a new text box at the cursor's position ("*But it's not nice with the touchpad, so I tried to avoid using it, and used shortcuts instead.*" p2). The participants were uncomfortable when inputting text ("*The table height was too low, which hurt my wrist. I tried to raise it, but it was already at the maximum height. In the end, I carried the keyboard to be more mobile, more flexible and to avoid wrist pain.*" p2) The workspace moved a lot inadvertently for both participants, who wished they could lock the zooming and panning.

3) Interacting using direct touch on the WSD: P1 preferred not to move the table. He mainly left the paper mock-up and the keyboard on the table ("*I didn't move the table, I preferred to have the papers in my hand in front of the display when I really need them. I didn't want to be cluttered with the table.*" p1). P2 preferred to move the table closer to the WSD to let the paper mock-up on it and to have quick access to the keyboard when needed, while maintaining access to the left menu on the WSD. She often moved the table to follow her process of working from left to right, but also sometimes pushed the table out of her way (see Figure 11.b). We also observed that both participants stepped back from time to time to get an overview (both participants did so for WSD-VW and only p2 for WSD-IA; "*We can't check the exact position of the created objects; to do this we need to move back.*" p2). They both knelt down a few times in WSD-VW to interact with the zoom menu, and p1 also did so to quickly modify some objects ("*I had to kneel down, that was more physically demanding than with other modalities.*" p1, see Figure 11 g and h). Touch was the modality requiring the most movement. Some physical fatigue arose, particularly in WSD-VW for p2 where she worked closer to the WSD ("*It hurts my neck when I am so close to the display and I look up. And it is warmer when we are close to the screens.*" p2).

We also observed that p1 and p2 first created the top objects of the mock-up at arm's height (vertically centered on the WSD), then moved the workspace upwards to correctly place the different elements "*I moved [panned] the designed interface so that it was at my height.*" p1, see Figure 11 a, b and c). On WSD-VW, both participants made use of the zoom in and out capabilities several times to design the smallest objects and then return to the initial view (see Figure 11 e then f). P2 noted that it was tiring to work at the top or the bottom of the WSD, especially when using the zoom widget at the bottom right (see Figure 11.g). They both created the titles first and then worked panel by panel. This eased the interactions and decreased the difficulty of working on such a huge surface by staying in a limited area and decreasing the amount of physical movements ("*I designed panel by panel, so I had the feeling of working only locally.*" p1).

A notable difference between WSD-IA and WSD-VW is that the participants moved back more often and further back, to get a bigger picture with WSD-VW than with WSD-IA ("*I needed to move back to have a global view.*" p1, when interacting on WSD-VW). It was also noted by p2 that touch on WSD-VW was more tiring than on WSD-IA. This could be explained by the fact that in WSD-VW, the zoom widget was far from the users when they stood in the main working area. To use it, they were forced to kneel down and press buttons, which is uncomfortable. However, they used the zoom widget more often on WSD-VW than on WSD-IA (only one time on WSD-IA, by p2), which might be due to the issues encountered

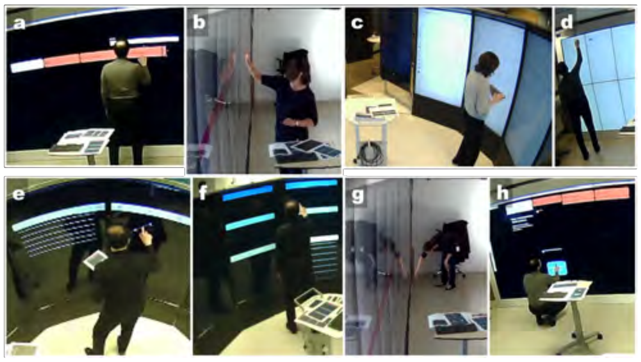


Figure 11. Using Miro with touch. In a) b) and c) participants create the objects at the WSD center. e) shows p1 zooming in and panning to design details, and then in f) repositioning the workspace correctly. In a) and f) p1 leaves the table in the middle. In b) p2 puts the table closer. d), g) and h) show moments when the participants interact too low or too high.

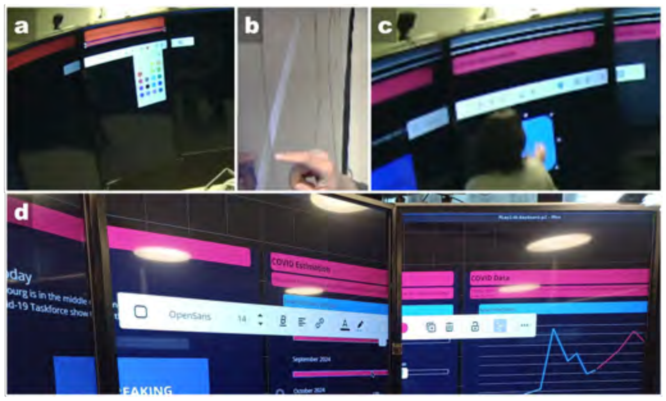


Figure 12. Bezel issue on WSD-IA (a, c and d), and on WSD WSD-VW (b).

with touch-based zooming and panning actions, which were sometimes too fast and sometimes laggy ("*It is more tiring than with WSD-IA because of the zoom button at the bottom right.*" p2). Overall, both participants had the feeling that touch worked better on WSD-VW than on WSD-IA.

The touch modality helps the most with the sense of space. It is also more instinctive and rapidly allows to place the created objects near their target location, even if the positioning is rarely immediately perfect ("*For some actions it is nice to do them directly on the display, e.g., when I add a text box, I can place it directly where I want.*" p2). The feeling of working directly on the interface was liked by participants ("*We can change directly on the display.*" p2).

Many issues appeared with touch on both WSDs. The sensitivity of the touch-based gestures was not adapted; sometimes it was too sensitive, sometimes not responsive enough. The technology used to recognize touch events was questioned ("*The touch needs to be improved.*" p1). Indeed, participants wondered whether the unexpected zooms, pans, and deselections/selections were triggered by pre-touch when a finger, sheet of paper or sleeve inadvertently entered the infrared detection zone of the touch frames (e.g., pointing a finger while thinking "*I wonder if it is due to a finger entering the detection zone of the infrared frame.*" p1, "*being very close to the display is sometimes enough to be detected as a touch.*" p1, "*It was disturbing when my pointing finger was close to the surface, because I was thinking before acting and then a touch was detected.*" p1). Both participants had trouble placing and resizing objects, particularly for the first object with this modality ("*I still have some issues with the touch behavior, notably the resizing of the objects and text fields.*" p1). This was particularly the case when objects were tiny ("*When the manipulated objects are too small, it is impossible or very hard to modify them, like with the sliders or the curves.*" p2). Furthermore, grasping tiny objects such as the handles used for resizing objects is complex and sometimes even impossible (p1 & p2). Modifying tiny objects therefore requires zooming in, and it then becomes difficult to readjust to the right zooming level and to position the mock-up in the workspace afterwards ("*When objects are very small, like the one composing the video camera icon, it was complicated to position them. So, I zoomed in. Then, I used the zoom menu to go back to a 100% zoom level.*" p2). Hence, the participants asked for an option to restore to the initial view, or at least to be able to go back to a given fixed view of the workspace.

Positioning the objects on different layers (z-index) was also problematic. Ensuring that each object was in the right layer was sometimes tricky. But, the main issue was that, when participants wanted to select an object in the foreground, a background object was often selected instead ("*When we are drawing objects composed of simple forms, it is complicated to specify what should be in the front/background.*" p1).

The amount of unexpected zooming, panning, and deselecting/selecting actions hampered work progress. In reaction, both participants had to frequently zoom and pan. They asked for the possibility to perfectly re-center the mock-up and resize the workspace based on the WSD dimensions ("*I need a button to rescale the scene.*" p1).

Screen bezels between the tiles composing the WSDs made the touch interaction harder. It was difficult to drag an element across screens because the bezel in between would interrupt the gesture ("*Objects need to be large enough to be moved across screens. Smaller bezels between two screens would improve the touch reliability.*" p1). Sometimes, the bezel occluded partially the options in the context menu (see Figure 12). Participants worked around it by first moving the selected object to ensure that the whole context menu was visible, then repositioned the object in the right position. Bezel-related issues were more raised by the participants with WSD-IA than with WSD-VW, likely because they are wider for that WSD.

Drawing straight lines was not possible using touch (p1 & p2 "*Drawing a straight line is complicated with touch because of finger tremor.*" p1).

While typing with the keyboard was appreciated, using an additional device poses a classic parking problem or requires the user to carry it around ("*The physical keyboard is better than the virtual keyboard on the tablet, and I was able to input* *more text. But the fact that the keyboard is placed on the table is not comfortable. Why not searching for a solution to have the keyboard closer without having to hold it in the hands, like attaching it to the left arm?*" p2).

4) Interacting on a synchronized tablet: Both participants moved very little during their sessions with the tablet. In WSD-IA, p1 and p2 stayed at the table and mostly did not move it. They both held the tablet during about half of the session, but p1 did so in the first part (see Figure 13.b), while p2 rather did that in the second half. They both changed their holding strategy because of the pain they felt in their arm (p1, see Figure 13.e) or neck (p2). They both showed signs of physical fatigue in the second half of the session (see Figure 13.d). P1 started the WSD-IA session with the tablet in his hand, then put the tablet on the table, complaining about the weight of the tablet. After a while, since the position at the table was not comfortable either, p1 took the tablet back in his hand. For WSD-VW, p1 left the tablet on the table throughout the session. None of the participants moved the table extensively, and they did always face the area they were working on. For instance, p1 created the rightmost panel on WSD-VW while staying at the table that was placed at the left of the WSD, as shown on Figure 13.b. P1 also sometimes held the tablet to move objects (e.g., title and subtitle of the rightmost panel) from a screen in his back to a target position he was looking at, without gazing at the objects during their movement. When the tablet was placed on the table, they both put their other hand (the one they were not using for interaction) on the table (as shown in Figure 13.a and g), except for p2 when she was entering text and therefore needed both hands (see Figure 13.f). P2 changed the tablet orientation from landscape to portrait in both WSD environments, for the whole second half of the session for WSD-IA and during 15 minutes in the beginning of the second half of the session for WSD-VW, before switching back to landscape in that case. P2 often tilted the tablet.During the WSD-VW session, p2 sat down on a table for the first fifteen minutes of the session (see Figure 13.c), then on a chair until the end of the session (see Figure 13.h). With WSD-VW, they both moved a little to check results on the WSD, leaving the tablet on the table. They both looked at the tablet for the creation and modification of objects, and then regularly checked the results on the WSD. As for the placement of objects, they partially looked at the tablet, mainly when the titles had already been created (as they then had a reference), but also looked at the WSD, although the lag between updates on the tablet and their effect on the WSD impeded these checks. They both opted to directly position the objects precisely on the WSD. They both also worked with the tablet by zooming in on part of the working area.

The most interesting aspect of this modality is that the viewports of the tablet and that of the WSD are dissociated. This allows the user to zoom in considerably on the tablet while keeping the view on the WSD unchanged. This was noted as the main advantage of the tablet modality by both participants ("*Dissociating the viewport of the tablet and that of the wall, especially for the zoom and pan*" was pointed out

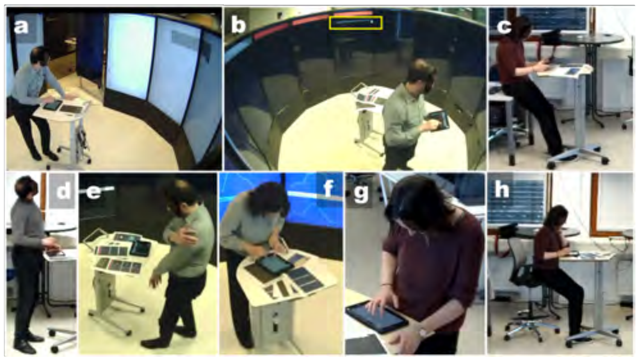


Figure 13. Using Miro with a tablet. a) and g) participants placed the tablet on the table and their non-interactive hand, b) p1 holds the tablet in hands, c) and h) shows p2 sitting down on a table and on a chair, d) and e) shows that p1 is uncomfortable and stretches f) shows p2 entering some text with two hands with the tablet placed on the table, in b) the yellow rectangle shows the object manipulated by p1 that is in his back.

as the reason that makes the system easy to use by p1, "*The decoupling between zoom and pan on the tablet and on the wall is interesting. As the tablet is tiny, I needed to zoom a lot.*" p1 "*The workspace on WSD-VW never moved, this is really comfortable.*" p2).

This zoom in also helped to place elements more precisely (as touch is more reliable on the tablet than on the WSD) ("*With zoom [on the tablet], it becomes possible to position smaller elements.*" p2). The ability to zoom in considerably cancels out the fat finger effect observed with the touch modality (p1) [69]. Creating small objects was experienced differently by each participant. For p1, the ability to zoom on the tablet without affecting the view on the WSD enabled him to work on the smaller objects and their details. However, p2 still found the tablet inadequate to design tiny objects ("*I avoided working on the very small details with the tablet, it was not adapted.*" p2)

P2 appreciated a lot the tablet condition, particularly on WSD-VW ("*It was with the tablet that I was able to do the most. It works best.*" p2).

The most negative aspect regarding the use of the tablet was the delay between actions performed on the tablet and their effect becoming visible on the WSD (p1 & p2, "*The tablet seemed slow, it didn't respond right away." p2, "There is a delay between moving something on the tablet and seeing it move on the WSD.*" p2). P1 additionally observed disconnections on the tablet at times.

Both participants found the tablet rather heavy, which made it difficult to carry it around. They therefore put it on the table, but the ergonomics of the table were not good enough, which led to pain in their wrist, neck, back and shoulders. The pain levels reported by the participants were more intense with WSD-VW. We note that the table was not exactly the same as that of WSD-IA, which could explain the increase in complaints ("*The tablet is heavy to carry. So, carrying it in my hands didn't suit me. So, I put it on the table. But in this position, my shoulder and neck muscles were hurting. I should have raised the table up, but I didn't think about it.*"

p1, "*I had a bit of pain in my wrist due to the position while interacting, especially while entering text.*" p2).

Positioning the objects precisely with the tablet was complicated ("*You can't place [the objects] as precisely as you would like.*" p1). Grouping objects was not easy either, using touch input on the small surface of the tablet. The guiding and snapping lines to help with aligning objects were visible only when interacting on the tablet. P1 would have preferred to see them on the WSD as well.

The working surface on the tablet is small, especially when the virtual keyboard is opened ("*But it is very small: the visible surface is very small, and the virtual keyboard takes a lot of the available space.*" p2). For this reason, p2 changed the tablet's orientation from landscape to portrait, to see the workspace better when the virtual keyboard was opened.

Participants deemed drag and drop difficult on the tablet, so was multiple selection. The drag and drop of an object was in fact accelerated depending on the distance on the tablet. However, as the surface is small, the acceleration quickly becomes too pronounced, which disturbed p1 ("*When I moved an object, it seemed like the speed of moving was changing depending on the distance.*" p1).

It was also noted by p2 that the available colors on the WSD and the tablet seemed slightly different, making the selection of the right color harder ("*It was harder for me to find the right colors. Maybe it's because they are not exactly the same on the tablet and on WSD-VW, or perhaps they are not perceived the same way.*" p2).

Last but not least, both participants did not like to type text with the tablet; they preferred the physical keyboard. The predictive text input of the tablet's virtual keyboard was not working well. Indeed, after p1 copied a text box, deleted the text in the new text box, and started to input new text, the predictive text input proposed a word starting with the first letter of the previous text ("*It's a weird behavior, and it poses a problem with the strategy I used for copying/pasting and then modifying.*" p1, "*Typing is more painful than with a keyboard. But with a more efficient automatic text completion system, it would be faster. But, here, it is not efficient enough. When I duplicated a text box and then deleted its text to enter new text, it seemed that the system was keeping the first letter in mind, although it was deleted, as it was still considered by the text completion system.*" p1).

To improve the use of the tablet, both participants suggested using a stylus for more precision, and a keyboard, mainly to facilitate text entry ("*A stylus to better move the small elements and a physical keyboard.*" p2). They also suggested using a wider tablet and would have liked a more ergonomic table.

5) Suggested improvements: Participants proposed different features and options to improve the 1:1 scale design on a WSD.

- an option to lock and unlock zoom on WSD (p1 & p2);
- an option to lock and unlock panning on WSD (p1 & p2);
- a button to center & zoom the design within the WSD viewport (p1 & p2);
- an enriched objects' library (p1 & p2);
- an improved touch detection system (p1 & p2);
- more flexibility regarding the font size (p1 & p2);
- the ability to move the context menu to work around occlusions by the bezel (p1 & p2);
- freehand drawing with an AI-based conversion of hand drawings into standard UI elements ("*It would be interesting to be able to draw by hand, to make sketches.*" p1, "*Hand drawing with an AI that recognizes drawings and normalizes drawn shapes.*" p1, "*Some AI assistance, which gets what I'm doing and enables me to go faster.*" p1);
- speech recognition to dictate text and for simple commands ("*As well as using some vocal commands, for instance to duplicate an object or change the color*" p1);
- limiting the number of sub-menus, particularly in the context menu, to 2 or 3 at most (p1);
- limiting pop-ups as much as possible; those that cannot be avoided should not appear in the middle-center part of the screen, but instead appear in the work area. (p1);
- a smart feature that recognizes series of actions that are repeated, to ask the user whether and how many times to repeat them ("*a smart function that, for example, if we repeat the same gesture [action] several times, then the system asks if we should repeat it again and how many times, e.g., for the labeled ticks along sliders.*" p1);
- a wearable keyboard (p2);
- supporting mid-air gestures ("*And maybe find a solution to move less [than with touch], like using mid-air pointing instead of touch, to support working from a distance.*" p2)
- adding specific actions for each tile composing the WSD, such as changing that screen's background ("*It would be interesting to be able to change the background for each screen*" p2), that would require to declare the tiling configuration of the WSD.

VII. Discussion

In comparison to the first user study, the sessions of the second one lasted longer (the full hour in all cases) and participants managed to advance much further in the creation of the mock-up. Hence, we can say that with Miro, participants could work more efficiently, and this tool can be considered as better adapted for the design in 1:1 scale on WSDs than Figma. Our observations have shown that this improvement was mainly due to the main menu being placed in the middle part on the left side, and the context menu directly next to the selected object that allows to select properties. User feedback further revealed that the context menus were appreciated for offering quick access to common actions (e.g., duplicate, delete, change the z-index, modify object properties). The most liked functionalities were the ability to duplicate objects, to make multiple selections, to position objects in different layers, and to modify the background color. As compared to the first user study conducted with Figma and its right-anchored menu, the physical fatigue was considerably reduced.

Despite this general improvement over Figma, the second user study also showed that some issues still remain with Miro, such as the position of the zoom widget placed in the bottom right, which was disliked by all participants.

In addition, the second user study has shown that the interaction modality and the WSD configuration has impacts onto the experience of the user, which we will discuss further in the following sections.

A. Benefits and issues with each modality

All three interaction modalities used in the user studies were deemed familiar by both p1 and p2, which made them easier to interact with the WSDs ("*It resembles the interaction modalities that we know on the desktop. It embraces the great metaphors we are accustomed to.*" p1, "*It uses standard interaction methods.*" p2). However, we found that in each of the conditions, participants faced issues and appreciated other features. Based on our observations, we categorized them into pros and cons, described below and summarized in Table IV.

P1 preferred the touch modality for the sense of acting directly on the WSD, then placed the tablet as second choice, with the keyboard last ("*My favorite modality is the touch, although I experienced some glitches that caused problems.*" p1, "*With the keyboard, positioning the objects is more precise and entering text is easier, but with the tablet I have better control on the zoom level on the display, and no unwanted zoom.*" p1). P2 preferred the tablet, then ranked the keyboard second owing to the ability to use shortcuts, and ranked the touch modality last. Touch was the most tiring interaction modality for both participants, because it required more movement. Concerning the keyboard, participants would have liked to use it with a mouse rather than a touchpad. As for the tablet, participants would have preferred a larger tablet, and would have liked to use a stylus as well as a physical keyboard.

When an interaction device was used in the given condition, p1 generally preferred to leave it on the table. When movement was necessary, he preferred to move the table, e.g., when accessing the zoom menu at the right bottom on WSD-VW ("*I preferred not to carry the keyboard in hand for comfort. I preferred to move the table, that was less painful.*" p1). In such situations, p2 preferred instead to carry the device around for more flexibility and to avoid wrist pain linked to entering text at an uncomfortable angle ("*I carried the keyboard with me in order to be more mobile, more flexible and to avoid wrist pain.*" p2, "*Having the tablet on the table was eventually an uncomfortable position for the neck, so, in the end, I preferred to hold the tablet in my hand.*" p2). Ultimately, both users complained of physical fatigue and that the positioning of the interaction devices was a pain-inducing issue. Interaction devices also divided the participants' attention between the device and the WSD, typically when entering text ("*[I] divided my attention between the display and the keyboard.*" p1).

Our results also show the main issues related to each interaction modality. In the keyboard + touchpad setting, the touchpad was too sensitive, which resulted in erratic movements of the workspace and created frustration. Also, typing text induced wrist pain due to the uncomfortable typing position, but this may be solved with a tilting table. Direct touch on the WSD induces physical fatigue and pain in the arm and neck. It lacks precision, particularly in the manipulation of tiny objects, and requires a lot of zoom and pan interactions. Some issues may be related to the touch hardware we are using, as it lacked robustness according to the participants. Finally, the tablet weight induces pain in the arm and neck, that could be alleviated by a tilting table, the working surface is too small, and a fat finger effect was observed, the long-range drag and drop is difficult, and lastly, there is a synchronization latency between the tablet and the WSD.

Overall, the participants would have preferred to have the ability to choose and switch between several interaction methods throughout the design session.

B. Difference between both WSDs

Our observations also indicate some differences between the two WSD settings. First, bezel issues were raised more frequently with WSD-IA than WSD-VW, likely because they are wider on WSD-IA, even if they are more numerous on WSD-VW.

The position of the zoom widget was disliked by participants on WSD-VW, although they did not comment on it on WSD-IA. This is probably due to the fact that it was displayed in the back of the participants on WSD-IA, and went unnoticed. Only p2 used these options on WSD-IA and only once, with the keyboard modality.

The physical ergonomics play an important role and must be handled with care. The height-adjustable and movable table in WSD-VW was liked and used, even if it was not ergonomic enough. Indeed, the participants were quite tall and the maximum height of the table in WSD-VW was insufficient for them. P2 adapted the table height in WSD-IA for more comfort. Participants wished they could tilt the table, especially with the keyboard and the tablet conditions to avoid wrist, shoulder, neck and back pain. P2 sometimes sat down in WSD-VW ("*I was not too tired, but maybe it was because I sat down.*" p2 when using the keyboard on WSD-VW). But, none of the participants sat in WSD-IA, likely because available chairs were behind the screens, i.e., outside the participants' field of view. It was noted by p2 that being able to sit down in WSD-VW was less tiring when using the keyboard and the tablet, and offered better positioning for entering text.

Finally, the space around WSD-VW was appreciated ("*It was nicer with [WSD-VW] than with [WSD-IA] because there was more space, and I sat down. I was able to arrange my desk better.*" p2) and led to a better user experience, likely because participants realized they could use a chair and sit down.

C. Implications for design

Based on our results from the two user studies, we can propose several **design guidelines** for creating a WSD design tool:

- Main actions should be reachable from the work area (e.g., creation of a new object, selected object properties, deletion, layering management).

TABLE IV. Pros and cons with each interaction modality used to design at scale 1:1 on a WSD.

	Keyboard & Touchpad	Touch	Tablet
Benefits	- Precise positioning (using arrow keys) - Easier text entering - Use of shortcuts - Lighter than a tablet	- Better sense of space - Instinctive, direct, and immediate	- Dissociated viewports of tablet and WSD, allowing two distinct views, - Easier work from a distance, reducing physical fatigue as it is possible to sit down - Easier work on tiny objects
Issues	- Touchpad not appreciated - Not comfortable and wrist pain - Workspace moves a lot - Unintended action triggering	- Physical fatigue - Arm and neck pain - Not so robust - Unintended action triggering - Not so precise (tiny objects cannot be done) - Requires a lot of zoom and pan	- Arm and neck pain - Weight of the tablet - Fat finger effect - Lag between the two views - Working surface is small - Long-range drag and drop is difficult - Virtual keyboard not appreciated

- Dialogue boxes should be avoided, or opened near the work area.
- All design tool elements should be legible from near and far, and actionable with the used modality.
- It should be possible to use the entire WSD surface for design and that the design tool elements do not get in the way, e.g., by making the design tool menus retractable, or movable. All context menus should be movable to avoid the bezels.
- A distinct device with a screen, like a tablet, should provide a second view for zooming in and carrying out design actions.
- It should be possible to lock the viewport of the workspace, with independent settings for zoom and pan actions.
- A possibility for resetting the viewport to a defined setting should be provided, as reopening the design at the same position.
- The system should support very large workspaces (gigapixel resolution).
- It should be possible to change the background color.
- It should be possible to duplicate (multiple) objects.
- Expert interactions, e.g., shortcuts, should be supported.
- There should be an enriched objects' library with advanced widgets, graphs, and icons to choose from.

Overall, regarding the different interaction methods, and based on our results, the best approach seems to be to **provide a mix of modalities**, including touch interaction for direct manipulation of objects on the WSD, a distinct device with a dissociated view (e.g., tablet), and a physical keyboard as efficient means to enter text.

Regarding **the design of the WSD itself**, it is better to reduce as much as possible the bezels between the display tiles. It is also important to ensure the visibility of the cursor at all times, and to facilitate long-range interactions, e.g., through pointing facilitation techniques [50].

Flat and circular displays have their pros and cons. Circular displays reduce the amplitude and the amount of movement required, text is easier to read from everywhere, but there is always a part of the screen at the back that can be invisible or forgotten, and there is less space to add furniture to make the interaction more comfortable. Flat displays, i.e., the most common form of WSDs, require much physical navigation. The user must move a lot to see the content on the other side of the screen and to get an overview. But there is more space around the display, which makes it easier to adapt it for a better user experience, for instance, by adding furniture.

To improve the user experience regarding the **space around the WSD**, we propose to provide: a height-adjustable mobile and tilting table to park interaction devices, and chairs for users to rest.

D. Limitations

Concerning the user studies, our findings come with several limitations. First, the task was limited to the reproduction of an existing UI prototype for a WSD environment, and no creative part was involved. In addition, the user studies involved only three participants who were all unfamiliar with the design tools used. The advantage here, however, was that they had no prior habits, e.g., using specific shortcuts, and were not frustrated by not being able to work as quickly as an expert would on a familiar software.

In the second user study, a clear learning effect was observed between the first (played on WSD-IA) and last sessions (played on WSD-VW). Indeed, participants were not experts at using Miro, despite having used it for other purposes than interface design. They discovered some features and faster ways of performing some actions during the sessions. This makes it difficult to compare the completion levels between them.

Finally, it must be mentioned that we encountered (unexplained) bugs with the display of the context menu in Miro, and that bezels would sometimes occlude menu options. Also, the mock-up covered mainly the left part of the display. We expect more issues to arise when users need to create objects on the right side of the display using touch (particularly for flat WSD such as WSD-VW) or the keyboard interaction methods.

VIII. Conclusion and Future Work

In this paper, we presented two user studies and a comparative analysis investigating the suitability of existing UI design tools to design at 1:1 scale on WSDs using three different interaction methods: touch, a keyboard with a touchpad, and a tablet. Our main takeaways are that (i) prototyping at 1:1 scale and being able to see live the final rendering in real time is appreciated, (ii) tablet-based interaction proved to be the most comfortable, (iii) each interaction method has its benefits and drawbacks; using a mix of modalities is promising, (iv) the spatial positioning of interaction elements on the screen, as well as the design of the physical environment, is of utmost importance to reduce physical fatigue and support efficient work, (v) among the nineteen desktop-optimized design tools analyzed, none satisfies all the criteria, but Miro is the most suitable.

So, to answer our research questions, a desktop optimized tool cannot be used as-is to design at 1:1 scale in a WSD environment. To improve the design of such a tool, we propose 12 design recommendations indicating how UI elements should be placed, and which features need to be provided, see Section VII-C. Regarding interaction modalities, the best approach for this type of design currently seems to be the synchronized tablet, but the other modalities also provide some unique benefits. A mix of interaction methods seems promising and is yet to be tested.

However, these studies involved few users and are focussed on the reproduction of a UI design. To validate the proposed guidelines, further experiments are needed.

More specifically, future work needs to further explore how 1:1 scale design can be supported by focussing on creating a new design rather than reproducing an existing one. As design is often a collaborative task, future research also needs to investigate how to manage collaboration during design sessions at 1:1 scale on WSDs. Additional questions are how modalities can be mixed and support the transitioning between multiple interaction methods in the same session. The studies from this paper enabled us to better understand the requirements for technical systems and the related problems and opportunities when using them to design for WSDs. In a longer perspective, this work is done as part of a design science methodology [70], ultimately aiming to create a design tool that is suitable for WSDs. The results presented in this paper are part of the first step, the relevance cycle, and will enable us to identify the user requirements.

IX. Acknowledgments

We thank all user study participants.

References

[1] L. Schwartz, V. Maquil, and M. Ghoniem, “Designing for and on wall-sized displays: a preliminary study with FIGMA,” in *CENTRIC 2024: The Seventeenth International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services*. IARIA, 2024, pp. 41–44.

[2] C. Parker, M. Tomitsch, N. Davies, N. Valkanova, and J. Kay, “Foundations for designing public interactive displays that provide value to users,” in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–12.

[3] M. Finke, A. Tang, R. Leung, and M. Blackstock, “Lessons learned: Game design for large public displays,” in *Proceedings of the 3rd International Conference on DIMEA Digital Interactive Media in Entertainment and Arts*, ser. DIMEA ’08. New York, NY, USA: Association for Computing Machinery, 2008, p. 26–33. [Online]. Available: https://doi.org/10.1145/1413634.1413644

[4] A. Prouzeau, A. Bezerianos, and O. Chapuis, “Towards road traffic management with forecasting on wall displays,” in *Proceedings of the ACM ISS International Conference on Interactive Surfaces and Spaces*. ACM, 2016, pp. 119–128.

[5] W. Buxton, G. Fitzmaurice, R. Balakrishnan, and G. Kurtenbach, “Large displays in automotive design,” *IEEE Computer Graphics and Applications*, vol. 20, no. 4, pp. 68–75, 2000.

[6] C. Liu, O. Chapuis, M. Beaudouin-Lafon, and E. Lecolinet, “Shared interaction on a wall-sized display in a data manipulation task,” in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 2075–2086. [Online]. Available: https://doi.org/10.1145/2858036.2858039

[7] D. Wigdor, H. Jiang, C. Forlines, M. Borkin, and C. Shen, “WeSpace: The design development and deployment of a walk-up and share multi-surface visual collaboration system,” in *Proceedings of the 2009 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 1237–1246. [Online]. Available: https://doi.org/10.1145/1518701.1518886

[8] F. Rajabiyazdi, J. Walny, C. Mah, J. Brosz, and S. Carpendale, “Understanding researchers’ use of a large, high-resolution display across disciplines,” in *Proceedings of the 2015 International Conference ITS on Interactive Tabletops &; Surfaces*. New York, NY, USA: Association for Computing Machinery, 2015, p. 107–116. [Online]. Available: https://doi.org/10.1145/2817721.2817735

[9] E. Pietriga, F. Del Campo, A. Ibsen, R. Primet, C. Appert *et al.*, “Exploratory visualization of astronomical data on ultra-high-resolution wall displays,” in *Software and Cyberinfrastructure for Astronomy IV*, vol. 9913. SPIE, 2016, pp. 344–358.

[10] J. Simonsen, H. Karasti, and M. Hertzum, “Infrastructuring and participatory design: Exploring infrastructural inversion as analytic, empirical and generative,” *Computer Supported Cooperative Work (CSCW)*, vol. 29, no. 1, pp. 115–151, 2020.

[11] J. Rambourg, H. Gaspard-Boulinc, S. Conversy, and M. Garbey, “Welcome onboard: An interactive large surface designed for teamwork and flexibility in surgical flow management,” in *Proceedings of the 2018 ACM International Conference ISS on Interactive Surfaces and Spaces*, ser. ISS ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 5–17. [Online]. Available: https://doi.org/10.1145/3279778.3279804

[12] M. M. Thomas, T. Kannampallil, J. Abraham, and G. E. Marai, “Echo: A large display interactive visualization of ICU data for effective care handoffs,” in *2017 IEEE Workshop VAHC on Visual Analytics in Healthcare*, Phoenix, USA, 2017, pp. 47–54.

[13] V. Doshi, S. Tuteja, K. Bharadwaj, D. Tantillo, T. Marrinan *et al.*, “Stickyschedule: An interactive multi-user application for conference scheduling on large-scale shared displays,” in *Proceedings of the 6th ACM PerDis International Symposium on Pervasive Displays*, ser. PerDis ’17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3078810.3078817

[14] J. Wall and M. Bertoni, “The decision arena: A model-centric interactive workspace for product-service system design,” in *Proceedings of NordDesign 2020*. Design Society, 2020, pp. 1–10.

[15] S. Kubicki, A. Guerriero, L. Schwartz, E. Daher, and B. Idris, “Assessment of synchronous interactive devices for BIM project coordination: Prospective ergonomics approach,” *Automation in Construction*, vol. 101, pp. 160–178, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580517304284

[16] L. Lischke, L. Janietz, A. Beham, H. Bohnacker, U. Schendzielorz *et al.*, “Challenges in designing interfaces for large displays: The practitioners’ point of view,” in *Proceedings of the 11th NordiCHI Nordic Conference on Human-Computer Interaction*. ACM, 2020, pp. 1–6.

[17] I. Belkacem, C. Tominski, N. Médoc, S. Knudsen, R. Dachselt *et al.*, “Interactive visualization on large high-resolution displays: A survey,” in *Computer Graphics Forum*. Wiley Online Library, 2022, p. e15001.

[18] L. Chen, H.-N. Liang, J. Wang, Y. Qu, and Y. Yue, “On the use of large interactive displays to support collaborative engagement and visual exploratory tasks,” *Sensors*, vol. 21, no. 24, p. 8403, 2021.

[19] C. Ardito, P. Buono, M. F. Costabile, and G. Desolda, “Interaction with large displays: A survey,” *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, pp. 1–38, 2015.

[20] L. Schwartz, V. Maquil, and M. Ghoniem, “The challenges of designing for large interactive displays,” in *IHM23 Adjunct*. ACM, 2023, pp. 1–6.

[21] M. Sinaei Hamed, P. Kwan, M. Klich, J. Aurisano, and F. Rajabiyazdi, “The elephant in the room: Expert experiences designing, developing and evaluating data visualizations on large displays,” *Proceedings of the*

ACM on Human-Computer Interaction, vol. 8, no. ISS, pp. 301–329, 2024.

[22] Figma. FIGMA design. Accessed: 2025.05.06. [Online]. Available: https://www.figma.com/design/

[23] Miro. Welcome to the innovation workspace. Accessed: 2025.05.06. [Online]. Available: https://miro.com/

[24] C. Snyder, *Paper prototyping: The fast and easy way to design and refine user interfaces*. Morgan Kaufmann, 2003.

[25] B. Bailey, J. Biehl, D. Cook, and H. Metcalf, "Adapting paper prototyping for designing user interfaces for multiple display environments," *Personal and Ubiquitous Computing*, vol. 12, pp. 269–277, 2008.

[26] I. Avellino, C. Fleury, W. E. Mackay, and M. Beaudouin-Lafon, "Camray: Camera arrays support remote collaboration on wall-sized displays," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 2017, pp. 6718–6729.

[27] D. Fallman, M. Kruzeniski, and M. Andersson, "Designing for a collaborative industrial environment: the case of the ABB powerwall," in *Proceedings of the 2005 conference on DUX Designing for User eXperience*, 2005, pp. 41–es.

[28] L. Lischke, S. Mayer, K. Wolf, N. Henze, H. Reiterer *et al.*, "Screen arrangements and interaction areas for large display work places," in *Proceedings of the 5th ACM PerDis International Symposium on Pervasive Displays*, 2016, pp. 228–234.

[29] M. Chegini, S. Lin, D. J. Lehmann, K. Andrews, and T. Schreck, "Interaction concepts for collaborative visual analysis of scatterplots on large vertically-mounted high-resolution multi-touch displays," in *Forum Media Technology*, 2017, pp. 90–96.

[30] R. Langner, U. Kister, and R. Dachselt, "Multiple coordinated views at large displays for multiple users: Empirical findings on user behavior, movements, and distances," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 608–618, 2018.

[31] A. Bezerianos, P. Dragicevic, and R. Balakrishnan, "Mnemonic rendering: an image-based approach for exposing hidden changes in dynamic displays," in *Proceedings of the 19th ACM symposium UIST on User Interface Software and Technology*, 2006, pp. 159–168.

[32] L. Vandenabeele, H. Afkari, J. Hermen, L. Deladiennée, C. Moll *et al.*, "DeBORAh: A web-based cross-device orchestration layer," in *Proceedings of the 2022 International Conference AVI on Advanced Visual Interfaces*, 2022, pp. 1–3.

[33] E. Pietriga, S. Huot, M. Nancel, and R. Primet, "Rapid development of user interfaces on cluster-driven wall displays with jbricks," in *Proceedings of the ACM EICS Symposium on Engineering Interactive Computing Systems*, 2011, pp. 185–190.

[34] H.-J. Kim, J.-W. Kim, and T.-J. Nam, "Ministudio: Designers' tool for prototyping ubicomp space with interactive miniature," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 213–224.

[35] H.-J. Kim, C. M. Kim, and T.-J. Nam, "Sketchstudio: Experience prototyping with 2.5-dimensional animated design scenarios," in *Proceedings of the 2018 DIS Designing Interactive Systems Conference*. ACM, 2018, pp. 831–843.

[36] I. S. MacKenzie and R. W. Soukoreff, "Text entry for mobile computing: Models and methods,theory and practice," *Human–Computer Interaction*, vol. 17, no. 2-3, pp. 147–198, 2002. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/07370024.2002.9667313

[37] U. Hinrichs, M. Hancock, S. Carpendale, and C. Collins, "Examination of text-entry methods for tabletop displays," in *Second Annual IEEE International Workshop TABLETOP on Horizontal Interactive Human-Computer Systems*, 2007, pp. 105–112.

[38] T. J. Dube and A. S. Arif, "Text entry in virtual reality: A comprehensive review of the literature," in *Conference HCII on Human-Computer Interaction. Recognition and Interaction Technologies*, M. Kurosu, Ed. Cham: Springer International Publishing, 2019, pp. 419–437.

[39] M. R. Jakobsen and K. Hornbæk, "Negotiating for space? collaborative work using a wall display with mouse and touch input," in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, ser. CHI '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 2050–2061. [Online]. Available: https://doi.org/10.1145/2858036.2858158

[40] F. Heidrich, M. Ziefle, C. Röcker, and J. Borchers, "Interacting with smart walls: a multi-dimensional analysis of input technologies for augmented environments," in *Proceedings of the 2nd AH Augmented Human International Conference*, ser. AH '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/1959826.1959827

[41] M. R. Jakobsen, Y. Jansen, S. Boring, and K. Hornbæk, "Should I stay or should I go? selecting between touch and mid-air gestures for large-display interaction," in *Conference INTERACT on Human-Computer Interaction*, J. Abascal, S. Barbosa, M. Fetter, T. Gross, P. Palanque, and M. Winckler, Eds. Cham: Springer International Publishing, 2015, pp. 455–473.

[42] A. Prouzeau, A. Bezerianos, and O. Chapuis, "Evaluating multi-user selection for exploring graph topology on wall-displays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 8, pp. 1936–1951, 2017.

[43] V. Maquil, D. Anastasiou, H. Afkari, A. Coppens, J. Hermen *et al.*, "Establishing awareness through pointing gestures during collaborative decision-making in a wall-display environment," in *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM, 2023, pp. 1–7.

[44] L. Schwartz, M. Ghoniem, V. Maquil, A. Coppens, and J. Hermen, "Designing at 1:1 scale on wall-sized displays using existing ui design tools - annex 1," 2025.

[45] ——, "Designing at 1:1 scale on wall-sized displays using existing ui design tools - annex 2," 2025.

[46] Google. The browser built to be yours. Accessed: 2025.05.06. [Online]. Available: https://www.google.com/intl/en/chrome/

[47] A. C. Figliolia, F. E. Sandnes, and F. O. Medola, "Experiences using three app prototyping tools with different levels of fidelity from a product design student's perspective," in *International Conference ICITL on Innovative Technologies and Learning*. Springer, 2020, pp. 557–566.

[48] Uxtools.co. UI design. Accessed: 2025.05.06. [Online]. Available: https://uxtools.co/survey/2023/ui-design/

[49] J. T. Hansberger, C. Peng, S. L. Mathis, V. Areyur Shanthakumar, S. C. Meacham *et al.*, "Dispelling the gorilla arm syndrome: the viability of prolonged gesture interactions," in *9th International Conference VAMR on Virtual, Augmented and Mixed Reality*. Springer, 2017, pp. 505–520.

[50] R. Balakrishnan, ""Beating" fitts' law: virtual enhancements for pointing facilitation," *International Journal of Human-Computer Studies*, vol. 61, no. 6, pp. 857–874, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S107158190400103X

[51] S. Phillora. (2024) 23 UX/UI design tools for better product design and UX. Accessed: 2025.05.06. [Online]. Available: https://maze.co/collections/ux-ui-design/tools/

[52] Adobe. (2024) Get started with ADOBE XD. Accessed: 2025.05.06. [Online]. Available: https://helpx.adobe.com/xd/get-started.html

[53] Axure. AXURE RP 11. Accessed: 2025.05.06. [Online]. Available: https://www.axure.com/

[54] Balsamiq. The effortless wireframing tool built for your big ideas. Accessed: 2025.05.06. [Online]. Available: https://balsamiq.com/

[55] Bubble. The full-stack, no-code app builder for everyone. Accessed: 2025.05.06. [Online]. Available: https://bubble.io/

[56] Canva. What will you design today? Accessed: 2025.01.30. [Online]. Available: https://www.canva.com

[57] Excalidraw. Excalidra. Accessed: 2025.05.06. [Online]. Available: https://excalidraw.com/

[58] Framer. The web builder for stunning sites. Accessed: 2025.05.06. [Online]. Available: https://www.framer.com

[59] Justinmind. Interaction design and prototyping tool for web and mobile apps. Accessed: 2025.05.06. [Online]. Available: https://www.justinmind.com

[60] MockFlow. Think. wireframe. brainstorm. with mockflow. Accessed: 2025.05.06. [Online]. Available: https://mockflow.com/

[61] MockPlus. Sign in to mockplus. Accessed: 2025.05.06. [Online]. Available: https://rp.mockplus.com/

[62] Penpot. Design and code beautiful products. together. Accessed: 2025.05.06. [Online]. Available: https://penpot.app/

[63] Proto.io. Prototyping for all. Accessed: 2025.05.06. [Online]. Available: https://proto.io/

[64] Protopie. #1 advanced prototyping tool for dynamic & multimodal interactions. Accessed: 2025.05.06. [Online]. Available: https://www.protopie.io/

[65] Sketch. Designers, welcome home. Accessed: 2025.05.06. [Online]. Available: https://www.sketch.com

[66] UXpin. Faster prototyping with AI component creation. Accessed: 2025.05.06. [Online]. Available: https://app.uxpin.com

[67] WebFlow. Your site should do more than look good. Accessed: 2025.05.06. [Online]. Available: https://webflow.com/

[68] C. Lallemand and G. Gronier, *Méthodes de design UX: 30 méthodes fondamentales pour concevoir et évaluer les systèmes interactifs*. Editions Eyrolles, 2015, original document in French, English title ”UX Design Methods: 30 Fundamental Methods for Designing and Evaluating Interactive Systems”.

[69] S. Voida, M. Tobiasz, J. Stromer, P. Isenberg, and S. Carpendale, “Getting practical with interactive tabletop displays: designing for dense data,” fat fingers,” diverse interactions, and face-to-face collaboration,” in *Proceedings of the ACM International ITS Conference on Interactive Tabletops and Surfaces*, 2009, pp. 109–116.

[70] A. R. Hevner, “A three cycle view of design science research,” *Scandinavian Journal of Information Systems*, vol. 19, no. 2, p. 4, 2007.

Comparing Closed-Source and Open-Source Code Static Measures

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
luigi.lavazza@uninsubria.it

***Abstract*—Most software engineering empirical studies are based on the analysis of open-source code. The reason is that open-source code is readily available, while usually software development organizations do not give access to their code, not even when the purpose is research and the code itself will not be disclosed. As a consequence, the corpus of empirical knowledge is related almost exclusively to open-source software. This poses a quite important question: do the conclusions we draw from the analysis of open-source code apply to closed-source code as well? In this paper, a comparison of open-source and closed-source code is performed, to provide some preliminary answers to the question. Specifically, the goal of the paper is to evaluate whether static code measures from open-source code are similar to those obtained from closed-source code. To this end, an empirical study was performed, involving closed-source code from two organizations and open-source code from a few different projects. The most popular static code measures were collected using a commercial tool, and compared. The study shows that open-source code measures appear similar to the measures obtained from industrial closed-source code. However, we must note that the study reported here involved just a few industrial projects' measures. Therefore, replications of the work presented here would be very useful.**

Keywords-software code measures; static code measures; open-source code; closed-source code.

I. INTRODUCTION

Software development organizations make their code available to researchers very rarely. This is due to their need for preserving the competitive advantage deriving from code ownership. As a consequence, the great majority of the empirical studies involving source code analyze open-source code, which is freely available. The conclusions reached by these studies are expected to apply to all code, including industrial closed-source code. However, the generalizability of studies based on open-source software relies on the assumption that closed-source software is "similar" to open-source software. Specifically, it is expected that the measures of open-source code are representative of closed-source software as well.

This paper describes an empirical study that aims at verifying if and to what extent code measures of open- and closed-source projects are similar. To this end, we measured a set of industrial closed-source projects and a set of open-source projects and compared the resulting measures. While in a previous paper [1] we considered only method-level measures of Java code, in this paper we extend the analysis to class-level measures.

Based on our results, there are no major differences among the measures collected from industrial and open-source projects. The study reported here has the merit to provide some objective evidence that studying open-source projects as representative of closed-source projects is sound.

In this study, the investigation is limited to static code measures for Java projects. Specifically, we consider the code metrics that are most frequently used in the research literature and the software industry, which can be easily obtained via any state-of-the-art tool.

The paper is structured as follows. Section II describes the static code measures investigated in this study. Section III describes the empirical study; results are given in Sections IV and V for method and class measures, respectively Section VI discusses the results obtained by the study. Section VII discusses the threats to the validity of the study. Section VIII accounts for related work. Finally, in Section IX some conclusions are drawn, and future work is outlined.

II. CODE MEASURES

Since the first high-level programming languages were introduced, several measures were proposed, to represent the possibly relevant characteristics of code [2]. For instance, the size of a software module is usually measured in terms of Lines Of Code (LOC), while McCabe Complexity (also known as Cyclomatic Complexity) [3] was proposed to represent the "complexity" of code, with the idea that high levels of complexity characterize code that is difficult to test and maintain. The object-oriented measures by Chidamber and Kemerer [4] were proposed to recognize poor software design. For instance, modules with high levels of coupling are supposed to be associated with difficult maintenance.

We have considered some of the most popular method-level measures (listed in Table I) and class-level measures (listed in Table II). All the measures mentioned in Tables I and II were collected via the SourceMeter tool [5]. Interested readers can find additional information concerning the definition and meaning of the selected metrics in the documentation of SourceMeter.

Halstead proposed several code metrics [6], based on the total number of occurrences of operators N_1, the total number of occurrences of operands N_2, the number of distinct operators η_1 and the number of distinct operands η_2. SourceMeter does not provide the individual measures of N_1, N_2, η_1 and η_2; instead, it provides Halstead Program Length (HPL), which is defined as $HPL = N_1 + N_2$, and Halstead Program Vocabulary (HPV), which is defined as $HPV = \eta_1 + \eta_2$. Halstead Volume (HVOL) is defined as $HVOL = (N_1 + N_2) * log_2(\eta_1 + \eta_2)$; Halstead Calculated Program Length (HCPL) is defined as

TABLE I
METHOD MEASURES COLLECTED VIA SOURCEMETER.

Metric name	Abbreviation
Lines of Code	LOC
Logical Lines of Code	LLOC
Halstead Program Length	HPL
Halstead Program Vocabulary	HPV
Halstead Calculated Program Length	HCPL
Halstead Volume	HVOL
Maintainability Index (Original version)	MI
McCabe's Cyclomatic Complexity	McCC

$HCPL = \eta_1 * log_2(\eta_1) + \eta_2 * log_2(\eta_2)$. McCabe's complexity (McCC) is used to indicate the complexity of a program, being the number of linearly independent paths through a program's source code [3]. The Maintainability Index (MI) [7] is defined as $MI = 171 - 5.2 * ln(HVOL) - 0.23 * (McCC) - 16.2 * ln(LLOC)$, where LLOC is the number of Logical LOC, i.e., the number of non-empty and non-comment code lines.

At the class level, we considered size metrics and the metrics proposed by Chidamber and Kemerer [4].

TABLE II
CLASS MEASURES COLLECTED VIA SOURCEMETER.

Metric name	Abbreviation
Lines of Code	LOC
Logical Lines of Code	LLOC
Coupling between Objects	CBO
Response for a Class	RFC
Weighted Methods per Class	WMC
Lack of Cohesion in Methods	LCOM5
Depth of Inheritance	DIT
Number of Children	NOC

Besides LOC and LLOC, which have the same meaning and definition as the corresponding metrics used for methods, the other class-level metrics are briefly described below [8].

WMC (Weighted methods per class) was defined as the sum of the complexities of its methods. As a measure of complexity SourceMeter assigns 1 to each method, hence WMC is equal to the number of methods in the class.

CBO (Coupling between object classes) represents the number of classes coupled to a given class (efferent couplings and afferent couplings) through method calls, field accesses, inheritance, arguments, return types, and exceptions.

RFC (Response for a Class) measures the number of different methods that can be executed when a method is invoked for that object.

LCOM (Lack of cohesion in methods) measures the lack of cohesion and computes into how many coherent classes the class could be split. The original definition of LCOM [4] was criticized because it depended on the number of methods in the considered class. A few alternative definitions were given. We use the one supported by SourceMeter, which is computed by taking a non-directed graph, where the nodes are the implemented local methods of the class and there is an edge between the two nodes if and only if a common (local or inherited) attribute or abstract method is used or a method invokes another. The value of the metric is the number of connected components in the graph not counting those, which contain only constructors, destructors, getters, or setters.

DIT (Depth of Inheritance Tree) provides for each class a measure of the inheritance levels from the hierarchy top class. In Java, where all classes inherit Object, the minimum value of DIT is 1.

NOC (Number of Children) measures the number of immediate descendants of the class.

III. THE EMPIRICAL STUDY

The empirical study involved closed-source and open-source Java programs. This code was measured, and the collected data were analyzed via well established statistical methods. The dataset is described in Section III-A, while the measurement and analysis methods are described in Section III-B. The results we obtained are reported in Sections IV and V.

A. The Dataset

As already mentioned, obtaining source code from software industries is not easy. Therefore, the closed-source code analyzed within the study is a convenience sample: it is the code that we were able to obtain from industrial developers. The open-source code analyzed within the study is the open-source code used within or together with the analyzed industrial projects. This guarantees a sort of "homogeneity" of code with respect to the required quality.

The open-source projects that supplied the code for the study are: Log4J [9], JCaptcha [10], Pdfbox [11], JasperReports (abbreviated JReports where necessary) [12], Hibernate [13].

Because of confidentiality reasons, the names of the industrial projects that supplied the code to be measured are not given: these projects are named Industrial1, Industrial2, Industrial3 (abbreviated Ind1, Ind2 and Ind3 where necessary). Ind1 and Ind2 are client and contract management systems from a large service company, Ind3 is the back-end of a web application. All of the industrial projects aimed to develop software supporting the main business of the owner companies, i.e., none of the considered projects delivered a product to be sold on the market. Also, all projects were developed by external software houses on behalf of the owner companies. Because of confidentiality reasons, the code and the raw measures are not available.

Table III gives some descriptive statistics of the considered projects.

TABLE III
DESCRIPTIVE STATISTICS OF THE DATASETS.

	Number of files	LOC total	LOC per file mean	sd	median	range
Ind1	1507	202299	134	268	91	[1–6851]
Ind2	280	56419	201	286	93	[3–2336]
Ind3	1323	250193	189	307	100	[6–3644]
Log4J	1067	126354	118	121	80	[20–1357]
JCaptcha	248	25292	102	99	75	[16–691]
Pdfbox	1215	252158	208	251	125	[21–2966]
JReports	3177	533008	168	285	89	[27–4398]
Hibernate	2392	236527	99	127	63	[9–2146]

Table III provides, for each analyzed project, the number of files, the total number of LOC, and the mean, standard deviation, median and range of the LOC per file.

B. The Method

The first phase of the study consisted in measuring the code. We used SourceMeter [5] to obtain the measures.

When analyzing the measures concerning methods, we decided to exclude from the study all the methods having unitary McCabe complexity, i.e., the methods that contain no decision points, since those methods would bias the results. In fact, these methods are quite numerous (since they include all the setters and getters) and very small (the excluded methods have mean and median LOC in the [3,6] range).

After removing the methods having unitary McCabe complexity, we got the dataset whose descriptive statistics are given in Table IV.

TABLE IV
DESCRIPTIVE STATISTICS OF THE DATASETS' METHODS, AFTER REMOVING METHODS WITH UNITARY MCCABE COMPLEXITY.

	Num. methods	LOC total	LOC per method mean	sd	median	range
Ind1	1342	32654	24	38	15	[3, 626]
Ind2	703	17099	24	25	16	[3, 197]
Ind3	3339	127170	38	61	21	[3, 1272]
Log4J	1729	29948	17	17	12	[3, 176]
JCaptcha	362	6386	18	15	13	[3, 100]
Pdfbox	3738	92679	25	26	16	[3, 380]
JReports	6815	180104	26	31	17	[3, 453]
Hibernate	2746	46505	17	15	12	[3, 221]

Table V gives the descriptive statistics of the datasets with respect to classes.

TABLE V
DESCRIPTIVE STATISTICS OF THE DATASETS' CLASSES.

	Num. classes	LOC total	LOC per class mean	sd	median	range
Ind1	1159	151726	131	293	65	[7, 6735]
Ind2	247	48261	195	273	88	[3, 2268]
Ind3	1389	222711	160	292	75	[2, 3639]
Log4J	1210	85242	70	93	39	[1, 1218]
JCaptcha	240	17941	75	93	50	[2, 648]
Pdfbox	1408	206029	146	225	72	[2, 2894]
JReports	2895	371364	128	262	55	[3, 4088]
Hibernate	2832	166082	59	95	32	[2, 2039]

Finally, we compared the collected measures. To this end, we provide a visual representation of the data via boxplots that describe the distributions, the mean and the median of the measures collected from each project. We also performed statistical analysis:

1) We performed a Kruskal-Wallis test for all the considered metrics, since the conditions for performing ANOVA tests did not hold. As a result, we obtained that, for all metrics, projects are not all equivalent with respect to the considered measure.
2) To explore in detail the differences among projects, we performed Wilcoxon rank sum tests for all project pairs, for all the considered metrics.
3) When a Wilcoxon rank sum test excluded that the measures are equivalent, we evaluated the effect size via Hedge's g.

In all the performed analysis, we considered the results significant at the usual $\alpha = 0.05$ level.

IV. RESULTS OF METHOD-LEVEL MEASUREMENTS

This section reports the data collected from methods, grouped according to the type of property being measured.

A. Size Measures

Boxplots of LOC measures are given in Figure 1: for the sake of readability, Figure 1 provides also a view without outliers. The mean values are represented as blue diamonds.

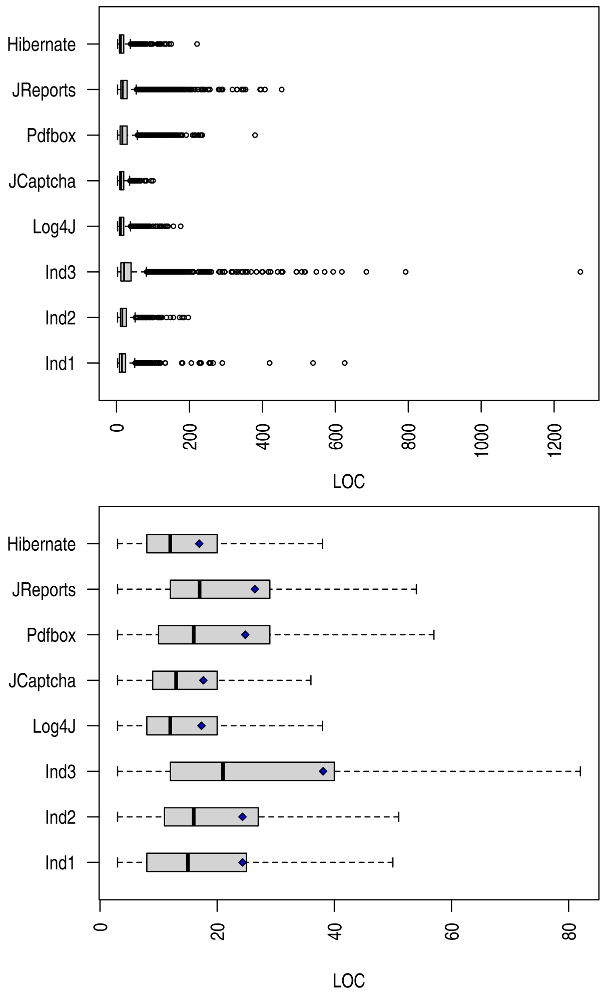


Figure 1. Boxplots of size (measured in LoC) distributions, with (top) and without (bottom) outliers.

The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning LOC are given in Table VI. Specifically, a cell includes symbol '=' if the Wilcoxon rank sum test could not exclude that the considered measures are equivalent; otherwise, a cell includes one of the symbols 'n,' 's,' 'm' for negligible, small and medium effect size, respectively (in no case a large effect size was found).

TABLE VI
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR METHODS' LOC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	s	s	n	n	n	s
Ind2	n	–	s	s	s	=	n	s
Ind3	s	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	s	s	n
JCaptcha	n	s	s	n	–	s	s	n
Pdfbox	n	=	s	s	s	–	n	s
JReports	n	n	s	s	s	n	–	s
Hibernate	s	s	s	n	n	s	s	–

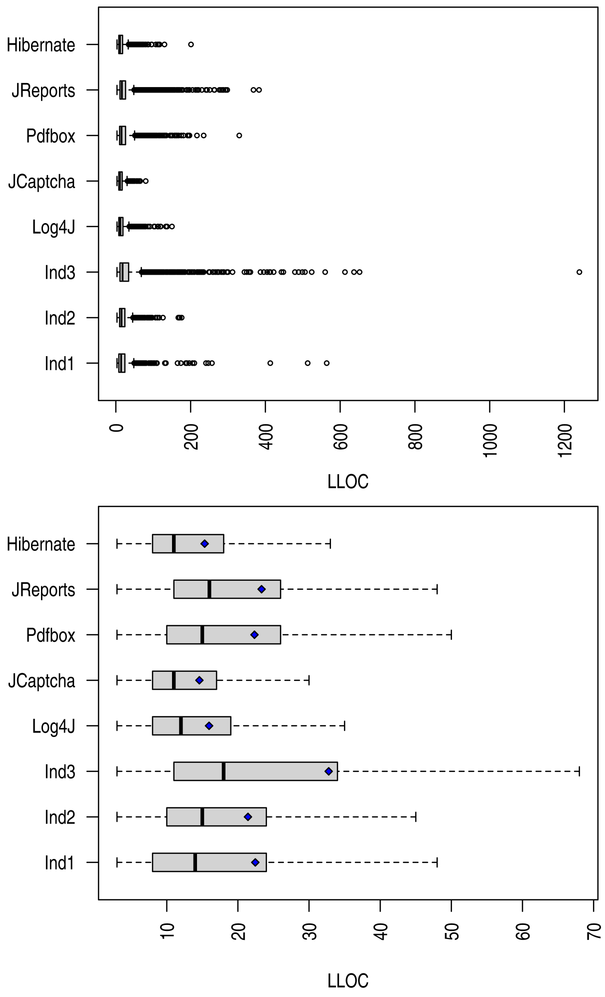


Figure 2. Boxplots of size (measured in LLoC) distributions, with (top) and without (bottom) outliers.

Boxplots of LLOC measures are given in Figure 2.

The results of the Wilcoxon rank sum tests and Hedges's g evaluations for LLOC measures are given in Table VII.

B. Complexity

Boxplots of McCabe cyclomatic complexity measures are given in Figure 3. For the sake of readability, Figure 4 provides the same data, excluding outliers.

The results of the Wilcoxon rank sum tests and Hedges's g evaluations are given in Table VIII.

TABLE VII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR LLOC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	s	s	s	n	n	s
Ind2	n	–	s	s	s	n	n	s
Ind3	s	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	s	s	=
JCaptcha	s	s	s	n	–	s	s	n
Pdfbox	n	n	s	s	s	–	n	s
JReports	n	n	s	s	s	n	–	s
Hibernate	s	s	s	=	n	s	s	–

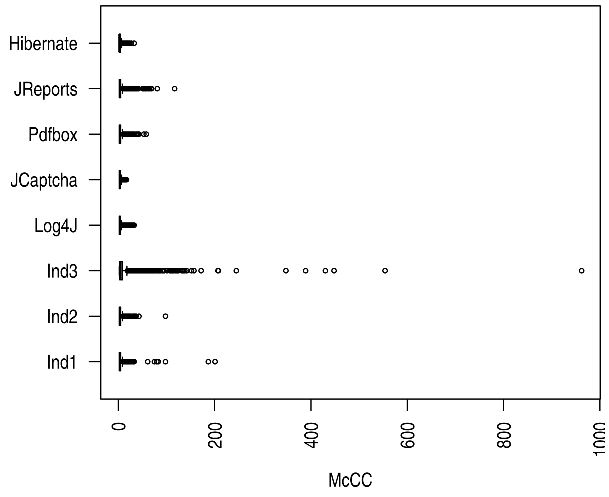


Figure 3. Boxplots of McCabe cyclomatic complexity distributions.

C. Maintainability

Maintainability is measured via the Maintainability Index (MI) [7].

Boxplots of MI measures are given in Figure 5. For the sake of readability, Figure 6 provides the same data, excluding outliers.

The results of the Wilcoxon rank sum tests and Hedges's g evaluations are given in Table IX.

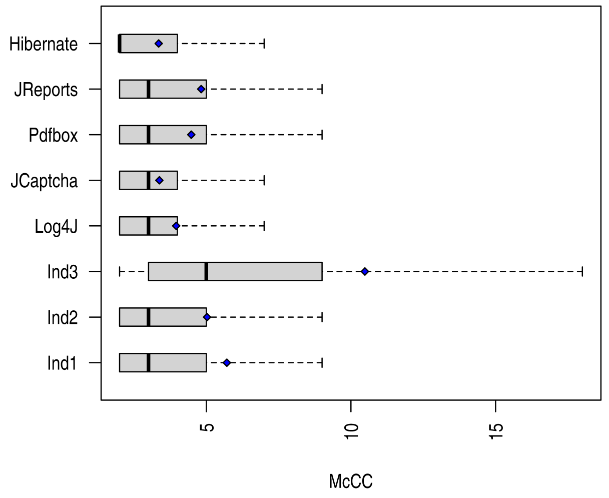


Figure 4. Boxplots of McCabe cyclomatic complexity distributions. Outliers omitted.

TABLE VIII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR MCCABE COMPLEXITY.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	n	s	s	n	=	s
Ind2	n	–	s	s	s	=	n	s
Ind3	n	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	n	n	s
JCaptcha	s	s	s	n	–	s	s	n
Pdfbox	n	=	s	n	s	–	n	s
JReports	=	n	s	n	s	n	–	s
Hibernate	s	s	s	s	n	s	s	–

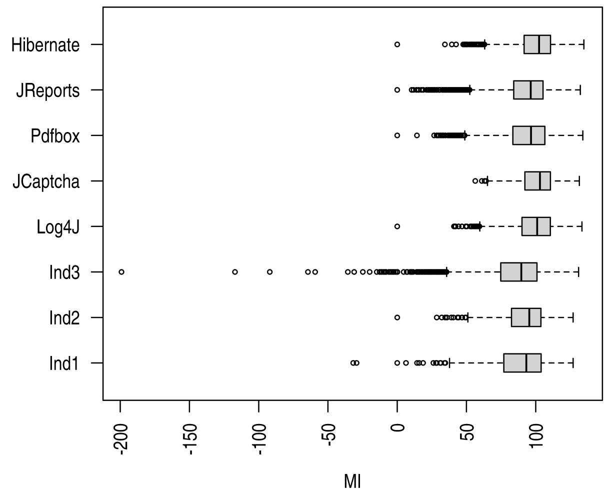


Figure 5. Boxplots of Maintainability Index MI distributions.

D. Halstead Measures

Halstead identified measurable properties of software in analogy with the measurable properties of matter [6]. Among these properties is the volume, measured via the Halstead Volume (HVOL). Boxplots of HVOL measures are given in Figure 7. For the sake of readability, Figure 8 provides the same data, excluding outliers. The results of the Wilcoxon rank sum tests and Hedges's g evaluations are given in Table X.

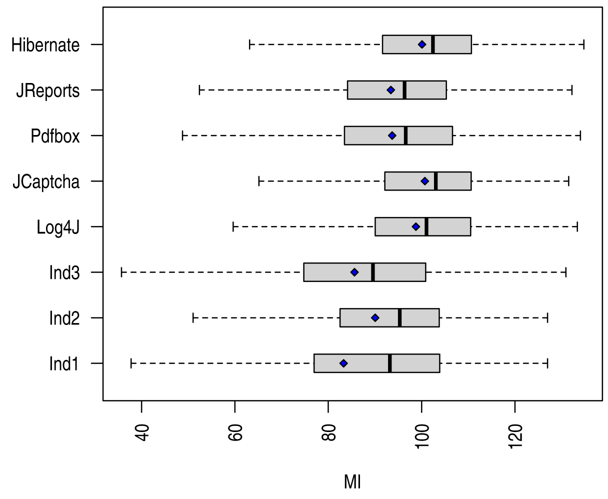


Figure 6. Boxplots of Maintainability Index MI distributions. Outliers omitted.

TABLE IX
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR THE MAINTAINABILITY INDEX (MI).

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	s	n	m	m	s	s	m
Ind2	s	–	n	s	m	n	n	m
Ind3	n	n	–	m	m	s	s	m
Log4J	m	s	m	–	n	s	s	n
JCaptcha	m	m	m	n	–	s	s	=
Pdfbox	s	n	s	s	s	–	n	s
JReports	s	n	s	s	s	n	–	s
Hibernate	m	m	m	n	=	s	s	–

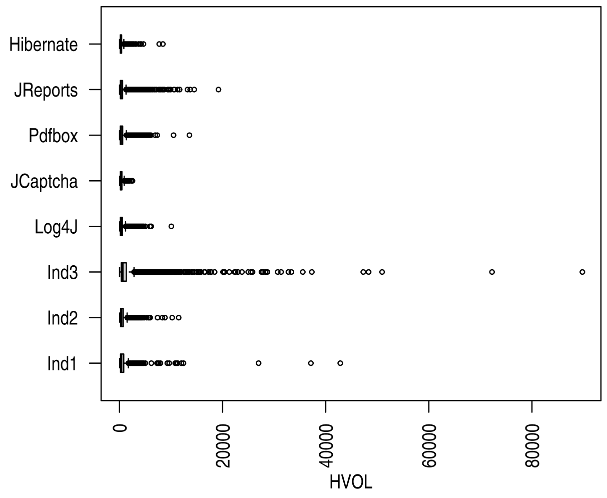


Figure 7. Halstead volume distributions.

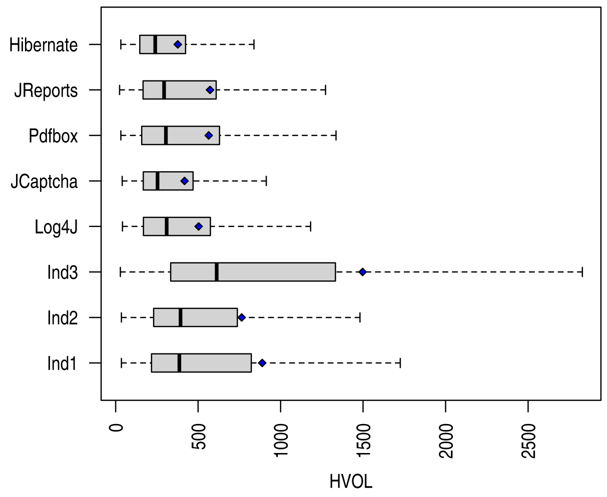


Figure 8. Halstead volume distributions. Outliers omitted.

Boxplots of Halstead Calculated Program Length (HCPL) measures are given in Figure 9. For the sake of readability, Figure 10 provides the same data, excluding outliers. The results of the Wilcoxon rank sum tests and Hedges's g evaluations are given in Table XI.

TABLE X
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR THE HALSTEAD VOLUME.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	=	n	s	s	s	s	s
Ind2	=	–	s	s	s	s	s	m
Ind3	n	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	n	=	s
JCaptcha	s	s	s	n	–	n	n	n
Pdfbox	s	s	s	n	n	–	n	s
JReports	s	s	s	=	n	n	–	s
Hibernate	s	m	s	s	n	s	s	–

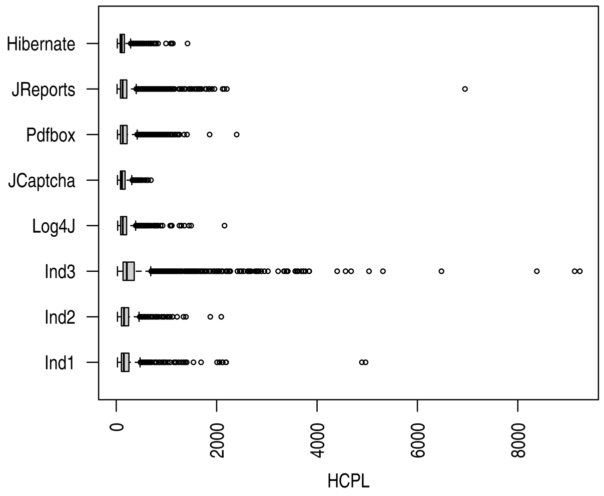


Figure 9. Halstead computed program length distributions.

V. RESULTS OF CLASS-LEVEL MEASUREMENTS

This section reports the data collected from classes, grouped according to the type of property being measured.

A. Size Measures

Boxplots of LOC and LLOC measures are given in Figure 11.

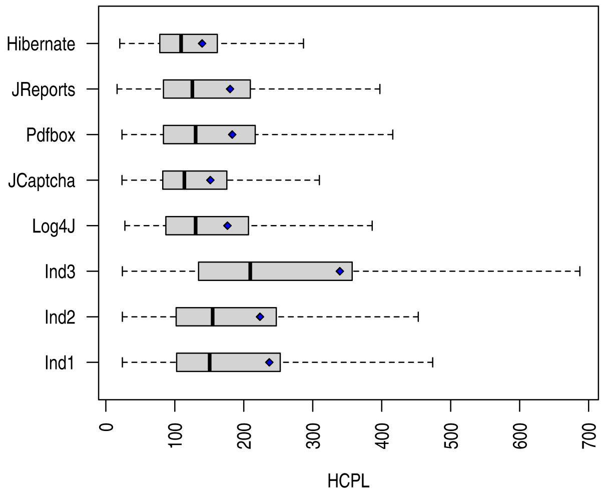


Figure 10. Halstead computed program length distributions. Outliers omitted.

TABLE XI
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR THE HALSTEAD COMPUTED PROGRAM LENGTH.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	=	s	s	s	s	s	m
Ind2	=	–	s	s	s	s	s	m
Ind3	s	s	–	s	s	s	s	m
Log4J	s	s	s	–	n	=	n	s
JCaptcha	s	s	s	n	–	n	n	n
Pdfbox	s	s	s	=	n	–	n	s
JReports	s	s	s	n	n	n	–	s
Hibernate	m	m	m	s	n	s	s	–

The results of the Wilcoxon rank sum tests and Hedges's g evaluations are given in Tables XII and XIII.

TABLE XII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR CLASS LOC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	s	s	n	n	n	s
Ind2	n	–	s	s	s	=	n	s
Ind3	s	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	s	s	n
JCaptcha	n	s	s	n	–	s	s	n
Pdfbox	n	=	s	s	s	–	n	s
JReports	n	n	s	s	s	n	–	s
Hibernate	s	s	s	n	n	s	s	–

TABLE XIII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR CLASS LLOC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	s	s	s	n	n	s
Ind2	n	–	s	s	s	n	n	s
Ind3	s	s	–	s	s	s	s	s
Log4J	s	s	s	–	n	s	s	=
JCaptcha	s	s	s	n	–	s	s	n
Pdfbox	n	n	s	s	s	–	n	s
JReports	n	n	s	s	s	n	–	s
Hibernate	s	s	s	=	n	s	s	–

B. Coupling Measures

Boxplots of CBO measures are given in Figures 12 and 13. The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning CBO measures are given in Table XIV.

TABLE XIV
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR CBO.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	s	n	n	s	s	s
Ind2	n	–	s	n	n	n	s	n
Ind3	s	s	–	s	s	s	n	s
Log4J	n	n	s	–	n	s	s	n
JCaptcha	n	n	s	n	–	s	s	s
Pdfbox	s	n	s	s	s	–	n	n
JReports	s	s	n	s	s	n	–	s
Hibernate	s	n	s	n	s	n	s	–

C. Response for a Class

Boxplots of RFC measures are given in Figures 14 and 15. The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning RFC measures are given in Table XV.

D. Results for WMC

Boxplots of WMC measures are given in Figures 16 and 17. The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning WMC measures are given in Table XVI.

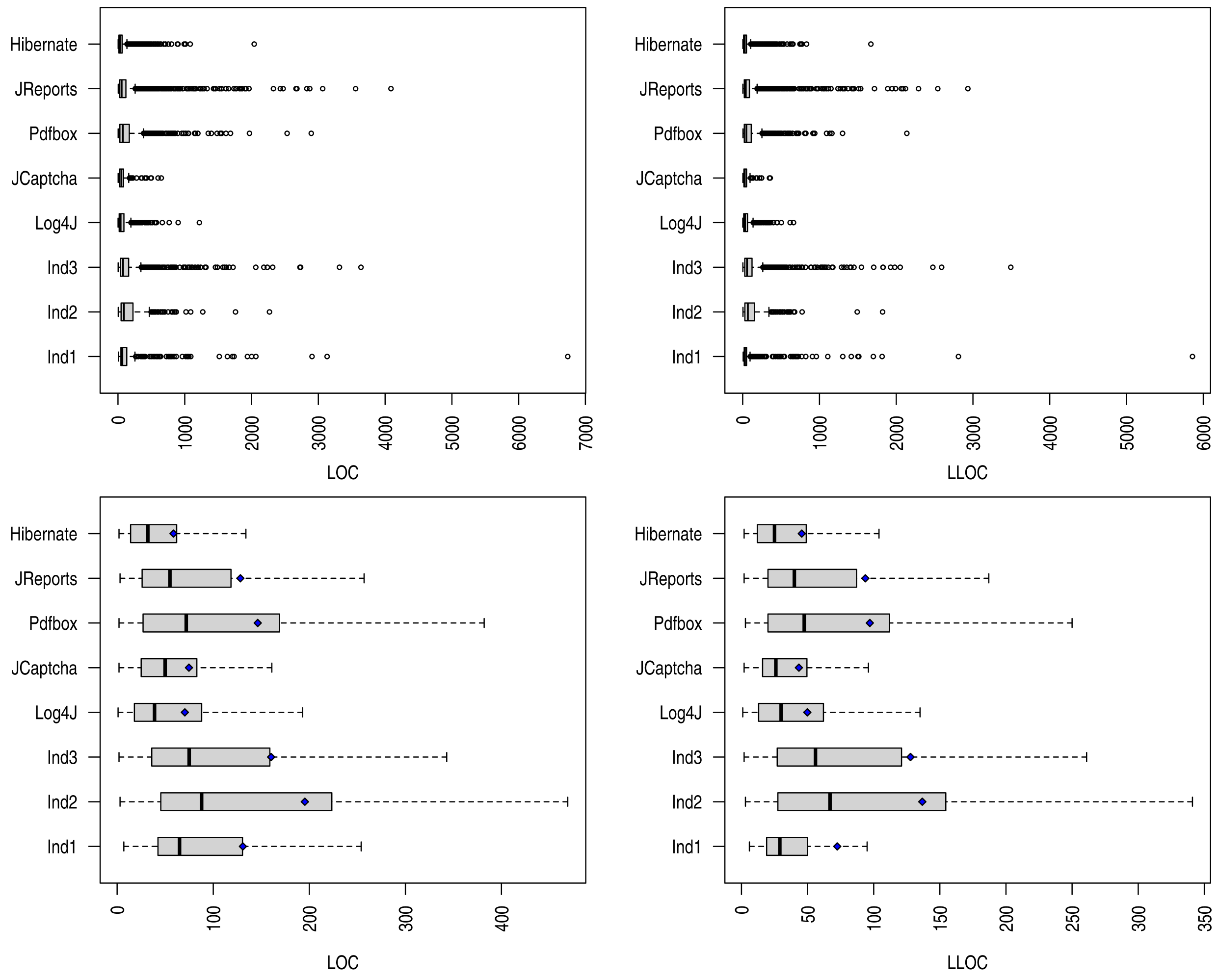


Figure 11. Boxplots of LoC (left) and LLOC (right) distributions, with (top) and without (bottom) outliers.

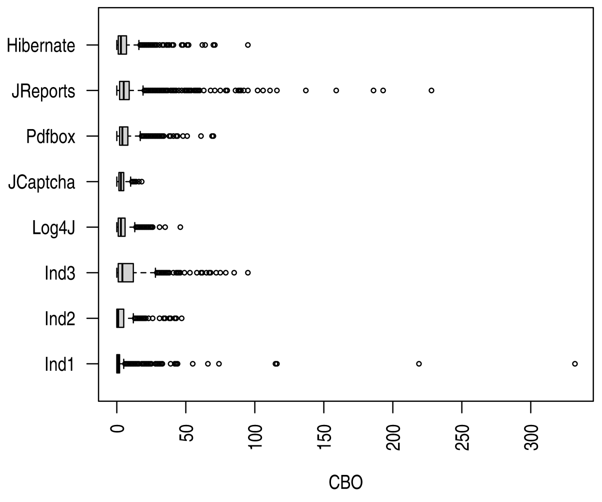


Figure 12. Boxplots of CBO distributions.

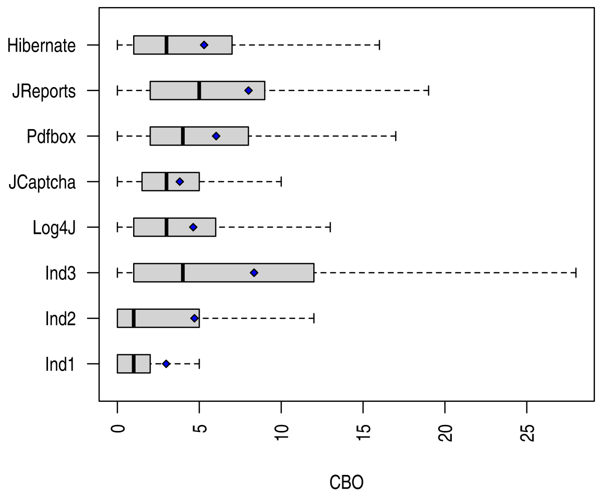


Figure 13. Boxplots of CBO distributions. Outliers omitted.

E. Results for Class Cohesion

Class cohesion was measured via the LCOM5 metric. Boxplots of LCOM5 measures are given in Figures 18 and 19.

The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning LCOM5 measures are given in

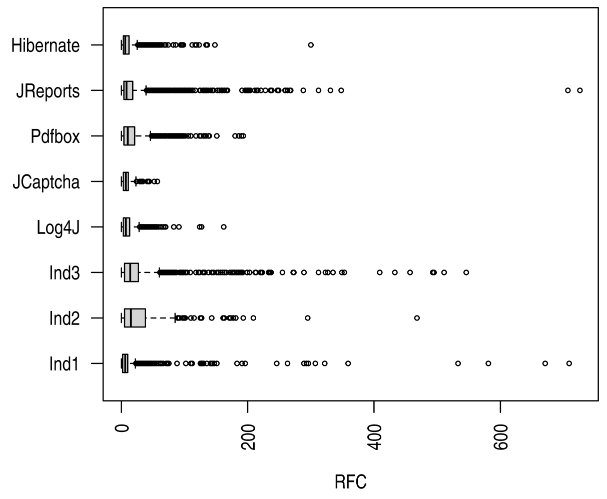


Figure 14. Boxplots of RFC distributions.

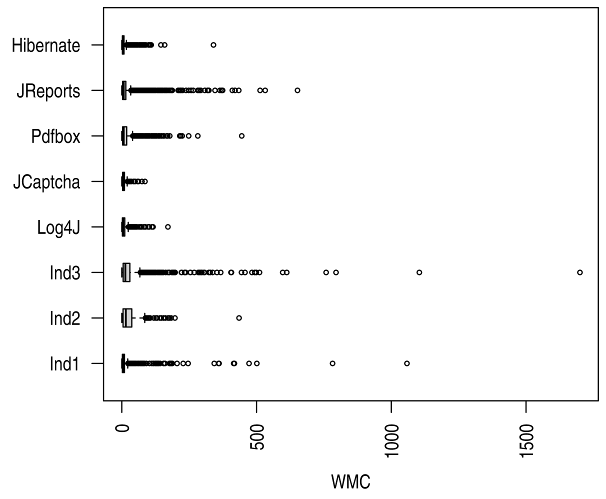


Figure 16. Boxplots of WMC distributions.

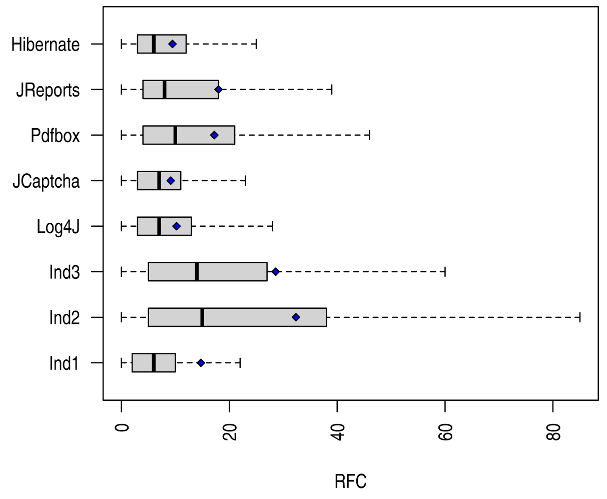


Figure 15. Boxplots of RFC distributions. Outliers omitted.

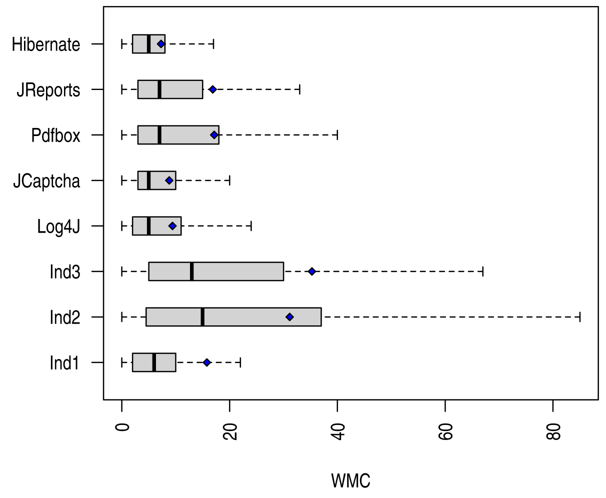


Figure 17. Boxplots of WMC distributions. Outliers omitted.

Table XVII.

F. Results for the Depth of Inheritance

Boxplots of DIT measures are given in Figures 20 and 21. The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning DIT measures are given in Table XVIII.

G. Results for the Number of Children

Boxplots of NOC measures are given in Figures 22 and 23. The results of the Wilcoxon rank sum tests and Hedges's g evaluations concerning NOC measures are given in Table XIX.

VI. Discussion

Figure 1 shows that the set of chosen projects are quite homogeneous with respect to size, all projects having the great majority of methods no longer than 200 LOC. This homogeneity is confirmed by the effect size evaluations given

TABLE XV
Wilcoxon rank sum test and Hedges's g results for RFC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	s	s	n	n	n	n	n
Ind2	s	–	n	l	m	m	s	l
Ind3	s	n	–	s	s	s	s	m
Log4J	n	l	s	–	=	s	s	n
JCaptcha	n	m	s	=	–	s	s	n
Pdfbox	n	m	s	s	s	–	n	s
JReports	n	s	s	s	s	n	–	s
Hibernate	n	l	m	n	n	s	s	–

TABLE XVI
Wilcoxon rank sum test and Hedges's g results for WMC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	s	s	n	n	n	n	s
Ind2	s	–	=	l	m	s	s	l
Ind3	s	=	–	s	s	s	s	m
Log4J	n	l	s	–	=	s	s	n
JCaptcha	n	m	s	=	–	s	s	n
Pdfbox	n	s	s	s	s	–	n	s
JReports	n	s	s	s	s	n	–	s
Hibernate	s	l	m	n	n	s	s	–

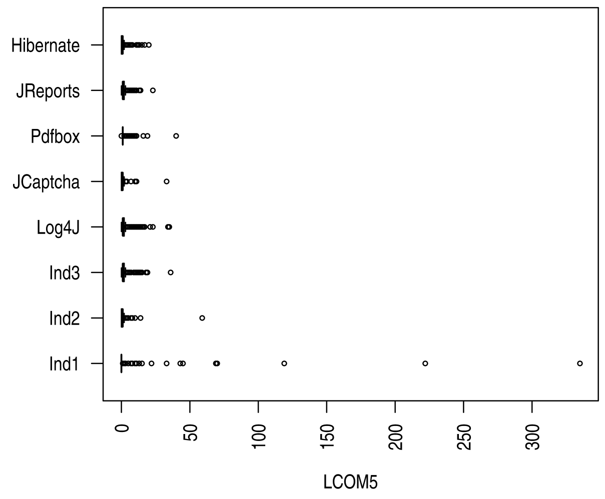


Figure 18. Boxplots of LCOM5 distributions.

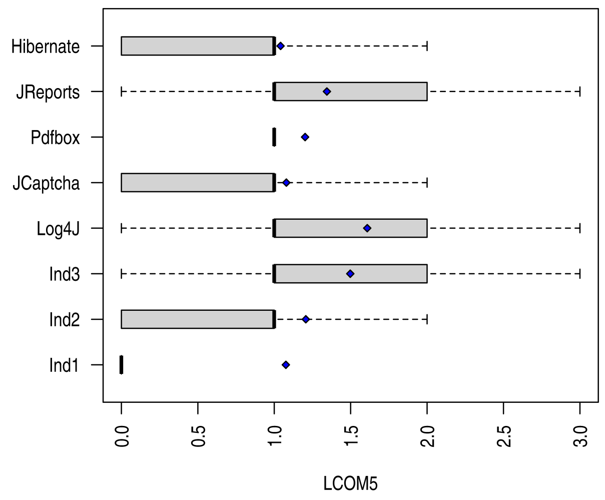


Figure 19. Boxplots of LCOM5 distributions. Outliers omitted.

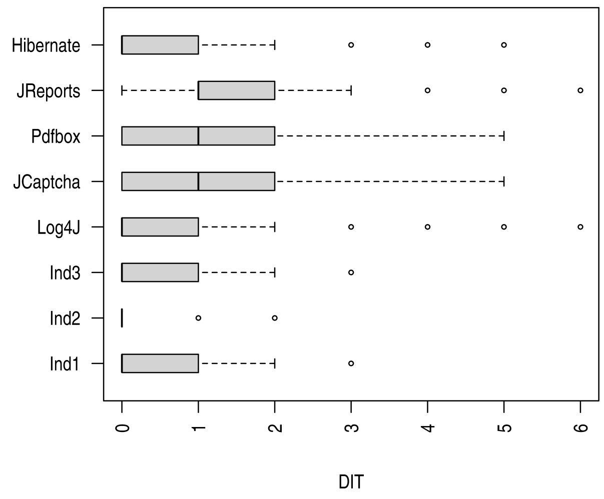


Figure 20. Boxplots of DIT distributions.

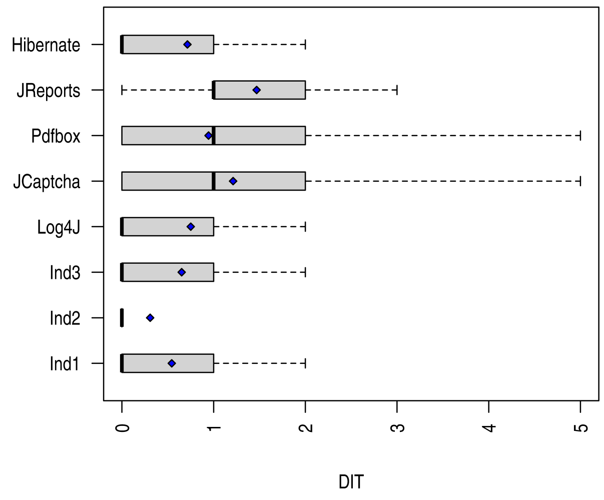


Figure 21. Boxplots of DIT distributions. Outliers omitted.

in Tables VI and XII: only negligible and small effect sizes were found, both at the method and class level.

Similarly, the great majority of methods have McCabe complexity not greater than 5 for all projects, with the only exception of Industrial3 (see Figure 4). However, also in project Industrial3, only outliers have alarmingly high McCabe complexity. As for LOC, the effect size is at most small, indicating substantial equivalence of the projects' complexity measures.

Concerning the Maintainability Index, Figure 5 shows that Industrial1 and Industrial3 are the only projects that include methods with negative MI; specifically, Industrial3 has several methods with negative MI, some with alarmingly low values. So, even though the situation excluding outliers (Figure 6) seems to indicate a rather homogeneous situation, industrial projects appear to be less maintainable then open-source projects in several cases: according to Table IX, in 8 out of 15 comparisons involving a closed-source and an open-source

TABLE XVII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR LCOM5.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	n	n	n	n	n	n
Ind2	n	–	n	n	n	n	n	n
Ind3	n	n	–	n	n	n	=	s
Log4J	n	n	n	–	s	s	n	s
JCaptcha	n	n	n	s	–	n	n	n
Pdfbox	n	n	n	s	n	–	n	n
JReports	n	n	=	n	n	n	–	s
Hibernate	n	n	s	s	n	n	s	–

TABLE XVIII
WILCOXON RANK SUM TEST AND HEDGES'S G RESULTS FOR DIT.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	s	n	n	m	s	m	n
Ind2	s	–	s	s	l	m	l	s
Ind3	n	s	–	n	m	s	m	n
Log4J	n	s	n	–	s	n	m	n
JCaptcha	m	l	m	s	–	s	s	m
Pdfbox	s	m	s	n	s	–	s	s
JReports	m	l	m	m	s	s	–	m
Hibernate	n	s	n	n	m	s	m	–

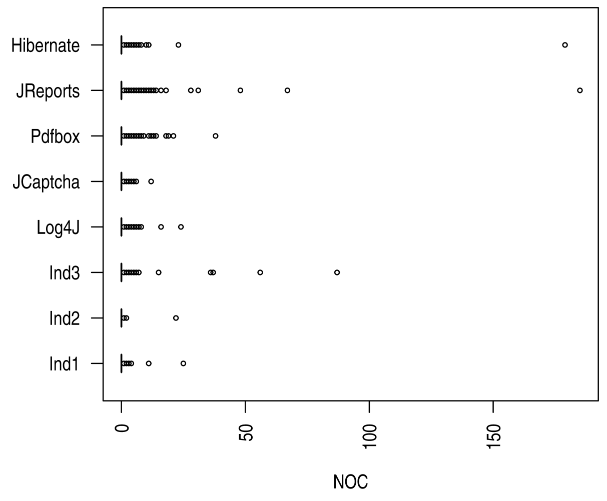


Figure 22. Boxplots of NOC distributions.

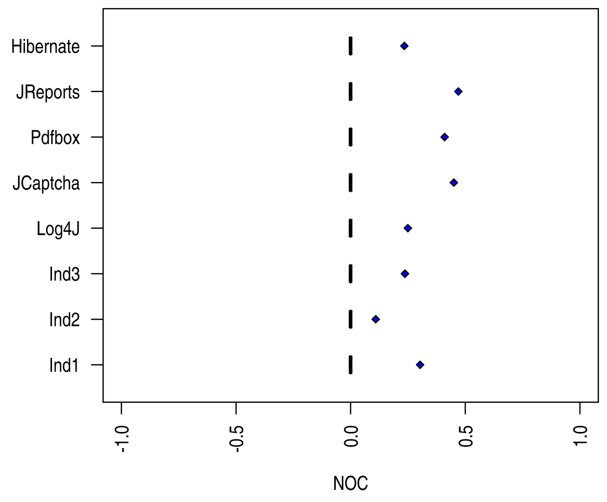


Figure 23. Boxplots of NOC distributions. Outliers omitted.

project, the effect size was medium. Instead, comparisons involving only open-source projects and comparisons involving only closed-source projects revealed at most small effect size.

Finally, we can see that all projects are fairly homogeneous with respect to Halstead volume (Figures 7 and 8 and Table X). Similar considerations apply for Halstead Computed Program Length (HCPL), with medium effect size differentiating industrial projects only with respect to Hibernate (Table XI).

In conclusion, we can observe that the analyzed open-source and closed-source code appear sufficiently similar, as far as method-level metrics are concerned.

Halstead volume and program length appear homogeneous through open-source and closed source programs (Tables X and X), with the exception of Hibernate, which appears "smaller" than the closed-source industrial programs.

Concerning coupling (CBO) and cohesion (LCOM5), no difference could be spot between open-source and closed-source programs.

RFC and WMC appear larger in Ind2 and Ind3, with medium effect size in just a few cases (Tables XV and XVI).

Finally, the situation appear quite homogeneous also when inheritance-related metrics (DIT and NOC) are concerned. There are several medium effect size values in Table XVIII concerning DIT, but we have to consider that all the DIT values are small: as shown in Figure 21, the great majority of classes has DIT not grater than 2. Only Ind2 seems to make very little usage of inheritance (its values of both DIT and NOC are definitely small).

Overall, neither for classes nor for methods there are remarkable differences between open-source and closed-source programs.

VII. Threats to Validity

Concerning the application of traditional measures, we used a state-of-the-art tool (SourceMeter), which is widely used and mature, therefore we do not see any threat on this side.

A risk with the type of work presented here is that the code that companies are willing to provide to researchers might differ from the code they would not provide. This is possibly due to the desire to "hide" low-quality code. In our case, it is not so: the closed-source code being measured is the complete code being used to build production applications and is representative of the companies' software in general.

Concerning the external validity of the study, as with most Software Engineering empirical studies, we cannot claim that the obtained results are generalizable. Specifically, the limited number of considered projects calls for replications of this study, involving more industrial closed-source code projects.

VIII. Related Work

Open-source projects have been compared with closed-source ones multiple times, but usually with respect to external perceivable qualities. In fact, many of the published papers aimed at answering questions like "Should I use this open-source software product or this closed-source one?" These papers considered issues like reliability, speed and effectiveness of defect removal, evolution, security, etc.

Bachmann and Bernstein [14] surveyed five open source projects and one closed source project to evaluate the quality and characteristics of data from bug tracking databases and version control system log files. Among other things, they discovered a poor quality in the link rate between bugs and commits.

The debate on the security of open-source software compared to that of closed-source software have produced several

TABLE XIX
Wilcoxon rank sum test and Hedges's g results for NOC.

	Ind1	Ind2	Ind3	Log4J	JCaptcha	Pdfbox	JReports	Hibernate
Ind1	–	n	n	n	n	n	=	n
Ind2	n	–	n	n	s	n	n	n
Ind3	n	n	–	n	n	n	n	n
Log4J	n	n	n	–	n	n	n	n
JCaptcha	n	s	n	n	–	n	n	n
Pdfbox	n	n	n	n	n	–	n	n
JReports	=	n	n	n	n	n	–	n
Hibernate	n	n	n	n	n	n	n	–

studies. This is due not only to the relevance of the problem, but also to the fact that security issues concerning closed-source software are publicly available, even when the source code is not.

Schryen and Kadura [15] analyzed and compared published vulnerabilities of eight open-source software and nine closed-source software packages. They provided an empirical analysis of vulnerabilities in terms of mean time between vulnerability disclosures, the development of disclosure over time, and the severity of vulnerabilities.

Schryen [16] also investigated empirically the patching behavior of software vendors/communities of widely deployed open-source and closed-source software packages. He found that it is not the particular software development style that determines patching behavior, but rather the policy of the particular software vendor.

Paulson et al. [17] compared open- and closed-source projects to investigate the hypotheses that open-source software grows more quickly, that creativity is more prevalent in open-source software, that open-source projects succeed because of their simplicity, that defects are found and fixed more rapidly in open-source projects.

As opposed to the papers mentioned above, here a fairly systematic comparison of code measures is proposed. Previously, MacCormack et al. compared the structure of an open-source system (Linux) an a closed-source system (Mozilla) [18]. With respect to our work, they evaluated just one code property (modularity) for a single pair of products.

A comparison based on code metrics involving multiple open-source and closed-source projects [19] was performed from a different point of view and using different techniques: the authors modified the Least Absolute Deviations technique where, instead of comparing metrics data to an ideal distribution, metrics from two programs are compared directly to each other via a data binning technique.

IX. Conclusions

Open-source projects provide the code used in many empirical studies. The applicability of the results of these studies to software projects in general, i.e., including closed-source projects, would be questionable, if open-source code were not representative of closed-source code as well.

To address this issue, a comparison of open-source and closed-source code was performed. Specifically, static code measures from five open-source projects were compared to those obtained from three closed-source projects. The study—which addressed only Java code—shows that some of the most well-known static code measures appear similar in open-source and in industrial closed-source products.

However, we recall that the study reported here involved just a few industrial projects' measures, because getting access to industrial code is not easy. Hence, the presented analysis should be regarded as a preliminary results, which needs replications before it can be considered valid in general.

References

[1] L. Lavazza, "A comparison of closed-source and open-source code static measures," in ICSEA 2024-The Nineteenth International Conference on Software Engineering Advances. IARIA, 2024, pp. 1–6.

[2] N. Fenton and J. Bieman, Software metrics: a rigorous and practical approach. CRC press, 2014.

[3] T. J. McCabe, "A complexity measure," IEEE Transactions on software Engineering, no. 4, 1976, pp. 308–320.

[4] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on software engineering, vol. 20, no. 6, 1994, pp. 476–493.

[5] "SourceMeter," https://www.sourcemeter.com/, [retrieved August, 2024].

[6] M. H. Halstead, Elements of software science. Elsevier North-Holland, 1977.

[7] K. D. Welker, P. W. Oman, and G. G. Atkinson, "Development and application of an automated source code maintainability index," Journal of Software Maintenance: Research and Practice, vol. 9, no. 3, 1997, pp. 127–159.

[8] Diomidis Spinellis, "ckjm – A Tool for Calculating Chidamber and Kemerer Java Metrics," https://www.spinellis.gr/sw/ckjm/doc/indexw.html.

[9] "Log4j."

[10] "Jcaptcha," https://jcaptcha.sourceforge.net/.

[11] "Apache pdfbox," https://pdfbox.apache.org/.

[12] "Jasperreports."

[13] "Hibernate," https://hibernate.org/.

[14] A. Bachmann and A. Bernstein, "Software process data quality and characteristics: a historical view on open and closed source projects," in Proceedings of the Joint int. ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops, 2009, pp. 119–128.

[15] G. Schryen, "Security of open source and closed source software: An empirical comparison of published vulnerabilities," AMCIS 2009 Proceedings, 2009, p. 387.

[16] ——, "A comprehensive and comparative analysis of the patching behavior of open source and closed source software vendors," in 2009 Fifth International Conference on IT Security Incident Management and IT Forensics. IEEE, 2009, pp. 153–168.

[17] J. W. Paulson, G. Succi, and A. Eberlein, "An empirical study of open-source and closed-source software products," IEEE transactions on software engineering, vol. 30, no. 4, 2004, pp. 246–256.

[18] A. MacCormack, J. Rusnak, and C. Y. Baldwin, "Exploring the structure of complex software designs: An empirical study of open source and proprietary code," Management Science, vol. 52, no. 7, 2006, pp. 1015–1030.

[19] B. Robinson and P. Francis, "Improving industrial adoption of software engineering research: a comparison of open and closed source software," in Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2010, pp. 1–10.

VR-GitEvo+CI/CD: Visualizing the Evolution of Git Repositories and CI/CD Pipelines in Virtual Reality

Roy Oberhauser[0000-0002-7606-8226]
Computer Science Dept.
Aalen University
Aalen, Germany
e-mail: roy.oberhauser@hs-aalen.de

Abstract – **Source code, together with its dependencies, is constantly under change pressure, and hence the codebase, typically stored and managed in a Git repository, evolves. Similarly, the associated DevOps CI/CD (Continuous Integration and Continuous Delivery) pipelines (automated processes or workflows), which automate the preparation and delivery of software artifacts, must adapt and evolve. However, visualization of the evolution of both the codebase and the associated CI/CD pipelines remains a challenge and hinders comprehension, analysis, and collaboration among DevOps stakeholders. To address this, our solution concept VR-GitEvo+CI/CD contributes an immersive visualization of codebases, dependencies, and CI/CD pipelines in Virtual Reality (VR). Our prototype realization shows its feasibility, while a case-based evaluation provides insights into its capabilities for supporting comprehension and analysis of the state and evolution of codebases and CI/CD pipelines.**

Keywords – Git; DevOps; virtual reality; visualization; software engineering; continuous integration; continuous delivery; CI/CD pipelines; code evolution; automation workflows.

I. Introduction

This paper extends our VR-DevOps paper [1], focusing on visualizing the evolution of both Git source code repositories and CI/CD (Continuous Integration and Continuous Delivery) pipelines [2] (a.k.a. DevOps pipelines); it incorporates VR visualization of Git repository evolution and extends our DevOps integration and capabilities.

"Everything moves on and nothing is at rest" is ascribed by Plato to Heraclitus [3] and reformulated by others in various ways; it essentially expresses that *change* or *dynamicity* is the only *constant* in our world. And yet many if not all of the restraints to change anchored in the physical world are absent in the digital world that software inhabits. As posited by F.P. Brooks Jr., software incurs essential difficulties or challenges that relate to its essence (inherent in its nature): complexity, conformity, changeability, and invisibility [4]. Software is essentially infinitely changeable, highly complex, exacting in conformance, and invisible. For developers, the invisibility of software obscures the underlying dependencies and complexity. This, in turn, hinders its comprehension, analysis, and management during the rapid evolution of both the codebase and the associated CI/CD pipeline, which automates the transformation of code into delivered (invokable) artifacts.

As to the degree of opaque dependencies, a 2024 industry analysis [5] of over 20,000 enterprise applications found they had 180 component dependencies on average (10% had over 400), with modern commercial software consisting of up to 90% Open-Source Software (OSS) components. As to change and evolution frequency, 6.6 trillion downloads were expected for 2024 across 7M OSS projects or components that encompass over 60M releases (on average 16 releases per OSS project annually) [5]. Already back in 2012, Google was said to have an average deployment frequency of 5,500 times a day [6][7], while Amazon was at 23,000 a day on average [6]. While we cannot extrapolate to today's rates for these IT behemoths, a 2021 survey of 1200 professionals indicated elite performers (26%) were deploying on demand multiple times a day [8]. The complex and obscure dependencies and rapid evolution of code and pipelines make comprehension challenging.

DevOps [9][10] is a methodology that combines development (Dev) and operations (Ops) with automation to improve the quality and speed of software deliveries. While there is no universally agreed to definition, key principles include Continuous Integration (CI), Continuous Delivery (CD), shared ownership, workflow automation, and rapid feedback. Both the code and tool integration and automation that DevOps addresses has become indispensable to modern software development. It has been reported that 83% of developers surveyed reported being involved in DevOps-related activities [11]. Lately, Security (Sec) has often been included in DevOps, denoted at the stage where it is primarily considered, e.g., DevSecOps [12]. Despite the popularity of DevOps, no modeling language nor visualization standard currently exists for CI/CD pipelines; each platform and vendor has their own, and it can thus be difficult for non-developers to grasp - and hence collaborate regarding - the current state of pipeline runs, and the processes involved in software development, testing, and delivery.

With the increasing demand for software functionality, large number of source code files are stored and managed in code repositories using Version Control Systems (VCS) like Git. GitHub has over 420M repositories with over 100M users [13]. A repository can become very large and continually evolve. For instance, in 2015 the Google monorepo shared by 25K developers contained 2B Lines of Code (LOC) across 9M source files having a history of 35M commits with 40K commits each workday [14], while the Linux kernel code repository contains over 40M LOC [15] across 60K files. For repositories, especially at such a scale, the dynamism of the changes as the codebase evolves can challenge comprehension and analysis.

Current visualization tools for Git repositories and CI/CD pipelines face inherent visualization limitations. While The potential of Virtual Reality (VR) to address both visual scalability and dynamic evolution has as yet been insufficiently explored. VR offers a mediated visual digital environment created and then experienced as telepresence by the perceiver. In contrast to a 2-dimensional (2D) space, VR enables an unlimited immersive space for visualizing and analyzing models and their interrelationships simultaneously in a 3D spatial structure viewable from different perspectives. In their systematic review of the DevOps field, Khan et al. [16] identified a lack of collaboration and communication among stakeholders as the primary critical challenge. Towards addressing this collaboration challenge, our contribution leverages VR towards enabling more intuitive DevOps visualization and interaction capabilities for comprehending and analyzing CI/CD pipelines, thereby supporting enhanced collaboration and communication among a larger spectrum of stakeholders. A further challenge is the finding by Giamattei et al. [17] that the landscape for DevOps tools is extremely fragmented, meaning stakeholders access various custom webpages or logs. Hence, a further goal of our solution concept is unifying the visualization and information access across heterogeneous Git and CI/CD tools.

In our prior VR work, VR-Git [18] and VR-GitCity [19] focused on Git support in VR with a vertical commit plane or city metaphor respectively, whereas VR-DevOps [1] focused on VR support for depicting pipelines. This paper contributes our solution concept VR-GitEvo+CI/CD, which provides an immersive visualization in VR of the state and evolution of both codebases, dependencies, and cross-platform CI/CD pipelines. Our prototype realization shows its feasibility, and a case-based evaluation provides insights into its potential towards supporting comprehension, analysis, and collaboration among DevOps stakeholders.

This paper is organized as follows: the next section discusses related work. Section III presents our solution concept. Section IV describes our realization, followed by an evaluation in Section V. Finally, a conclusion is drawn.

II. Related Work

Work on VR-based visualization of Git includes our own prior work VR-Git [18] and VR-GitCity [19]: VR-Git uses consecutive vertical commit planes on a hyperplane to represent commits, whereas VR-GitCity uses a city metaphor to convey code sizes across files. Bjørklund [20], who used a directed acyclic graph visualization in VR using the Unreal Engine, with a backend using NodeJS, Mongoose, and ExpressJS, with SQLite used to store data. GitHub Skyline [21] provides a VR Ready 3D contribution graph as an animated skyline that can be annotated. VRGit by Zhang et al. [22] depicts the non-linear version history via a history graph anchored to the user's arm, where each node is a 3D miniature of that version highlighting changed objects.

As to non-VR based Git visualization, Git-Truck [23] hierarchical visualizations (i.e., Treemaps, Circle packing) to represent files nested in folders, with code metrics mapped to size and color; GitTruck@Duck [24] extends this further for filtering polymetric views scoped to time intervals. Gource [25] and CodeFlower [26] use an animated tree with directories as branches and files as leaves. Githru, [27] enables interactive scalable exploration of large Git commit graphs using graph reconstruction, clustering, and context-preserving squash merge. Evo-Clocks [28] represents repository evolution with each node history depicted as a separate disk clock. RepoVis [29] offers a comprehensive visual overview and search facilities using a 2D JavaScript-based web application and Ruby-based backend with a CouchDB. Githru [30] utilizes graph reconstruction, clustering, and context-preserving squash merge to abstract a large-scale commit graph, providing an interactive summary view of the development history. VisGi [31] utilizes tagging to aggregate commits for a coarse group graph, and Sunburst Tree Layout diagrams to visualize group contents. It is interesting to note that the paper states "showing all groups at once overloads the available display space, making any two-dimensional visualization cluttered and uninformative. The use of an interactive model is important for clean and focused visualizations." UrbanIt [32] utilizes an iPad to support mobile Git visualization aspects, such as an evolution view. Besides the web-based visualization interfaces of Git cloud providers, various desktop Git tools, such as Sourcetree and Gitkracken, provide typical 2D branch visualizations.

Regarding VR-based DevOps-related work, VIAProMa [33] provides a visual immersive analytics framework for project management. DevOpsUseXR is mentioned in the paper as an eXtended Reality (XR) approach for incorporating end users to allow them to directly provide feedback in Mixed Reality (MR) regarding their experience when using a specific MR app. In contrast, our solution concept is independent of the software type being built in the pipeline, and is purely virtual, remaining consistent, app-independent, and focusing on visualizing and collaborating with regard to the DevOps pipeline. The systematic review of DevOps tools by Giamattei et al. [17] does not mention any VR, XR, or MR tools.

Non-VR based DevOps work includes DevOpsML [34], a platform modeling language and conceptual framework for modeling and configuring DevOps engineering processes and platforms. DevOpsUse [35] expands DevOps to collaborate more closely with end users. The authors also state that there is in general a research gap in applying information visualization to software engineering data, and that this needs further investigation. This concurs with our view, as we were not able to find much VR or non-VR work related to DevOps visualization. Zampetti et al. [2] analyzed the pipeline evolution of 4,644 projects in 8 programming languages using Travis-CI, creating a taxonomy of 34 CI/CD pipeline restructuring actions and metric extractor of 16 indicators of how a pipeline evolves.

In contrast to the above work, VR-GitEvo+CI/CD focuses on immersively visualizing the dynamic evolution of both the codebase in Git repositories and heterogeneous CI/CD pipelines.

III. Solution Concept

Our solution approach leverages VR for visualizing the evolution of codebases and CI/CD pipelines via models that can be immersively explored and experienced in 3D.

A. *Grounding in VR-Related Research*

Our rationale for integrating VR into our solution concept stems from existing VR research in fields we consider relevant to modeling, analysis, and collaboration, several of which are summarized here. Akpan and Shanker [36] in their systematic meta-analysis demonstrate that VR and 3D provide marked advantages in discrete event modeling, including model development, analysis, and Verification and Validation (V&V), with common findings pointing to the positive impact of 3D/VR model analysis and V&V. Across 23 studies on 3D analysis, 95% found 3D more effective and conducive to enhanced analysis compared to 2D, notably when assessing model behavior or conducting what-if scenarios; there was broad agreement that 3D/VR effectively communicates results to decision-makers with clarity and conviction; 74% of 19 papers concluded that 3D/VR significantly enhances model development tasks, benefiting team support, and sharpening precision and clarity. In exploring VR applicability for analytical tasks within an information architecture, Narasimha et al. [37] conducted a card sorting collaboration study. Their findings indicated VR matched or exceeded in-person card sorting for certain variables, surpassing both traditional and video-based settings. Qualitative insights on awareness suggested that during collaboration, participants maintained awareness of tasks, people, and the environment, mimicking in-person interactions, with positive perceptions of VR. These findings indicate that VR facilitates a sense of presence and collaboration akin to face-to-face settings. Fonnet and Prie's [38] survey of Immersive Analytics (IA) reviewed 177 papers and found that for complex graph and spatial data, IA offers advantages over non-IA methods, though for multi-dimensional data, benefits vary with the task. They highlight that while IA supports the exploration of extensive data environments, the underutilization of context-aware navigation techniques is problematic, despite their importance to users. Müller et al. [39] compared VR vs. 2D for a software analysis task, finding no significant decrease for VR in comprehension and analysis time. While interaction time was less efficient, VR improved the user experience, was more motivating, less demanding, more inventive/innovative, and more clearly structured.

Consequently, we infer that an immersive, context-rich VR experience holds significant promise for comprehensively depicting 3D models while enhancing comprehension, awareness, analysis, and the inclusion of and collaboration with stakeholders.

B. *Prior VR-Related Research*

Our VR-GitEvo+CI/CD solution concept is highlighted in blue relative to our other VR solutions in Figure 1. It is based on our generalized VR Modeling Framework (VR-MF), detailed in [40], which provides a VR-based domain-independent hypermodeling framework addressing four aspects requiring special attention when modeling in VR: visualization, navigation, interaction, and data retrieval/integration.

Our VR-based solutions specific to the SE and Systems Engineering (SysE) areas include: VR-DevOps [1], which this paper extends; VR-Git [18] and VR-GitCity [19] visualize Git repositories; VR-SDLC (Software Development LifeCycle) [41] visualizes lifecycle activities and artefacts in software and systems development; VR-SBOM [42] for Software Bill of Materials and Supply Chain visualization; VR-ISA [43] for visualizing an Informed Software Architecture; VR-V&V (Verification and Validation) [36], for visualizing aspects related to quality assurance; VR-TestCoverage [45] for visualizing in VR which tests cover what test target artefacts; VR-UML [46] supports visualizing UML models; and VR-SysML [47] supports Systems Modeling Language (SysML) models.

In the Enterprise Architecture (EA) and Business Process (BP) space (EA & BP), we developed VR-EA [48] to support mapping EA models to VR, including both ArchiMate as well as BPMN via VR-BPMN [39]; VR-EAT [49] adds enterprise repository integration (Atlas and IT blueprint integration); VR-EA+TCK [50] adds knowledge and content integration; VR-EvoEA+BP [51] adds EA evolution and Business Process animation; while VR-ProcessMine [52] supports process mining in VR. Since DevOps (or DevSecOps or DevOpsUse) can be viewed as inter-disciplinary, for software organizations we view both the EA and BP areas as potentially applicable for VR-GitEvo+CI/CD to support synergies, more holistic insights, and enhanced collaboration across the enterprise and organizational space.

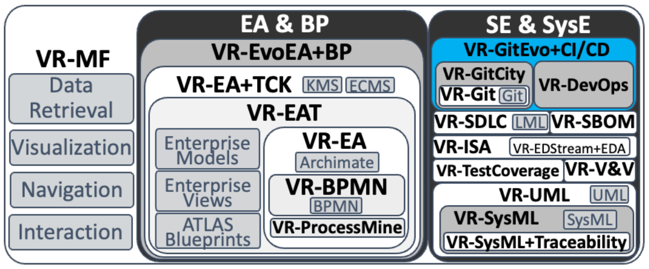


Figure 1. Conceptual map of our various published VR solution concepts with VR-GitEvo+CI/CD highlighted in blue.

C. *Visualization in VR*

A *pipeline* is represented as a horizontal *pipeline hyperplane*, holding vertical semi-transparent colored boxes called *run planes* (see Figure 2), which are ordered chronologically left to right. A *run plane* represents a pipeline *run*, which is colored based on status (green=success, yellow=in progress, red=error, grey= aborted). Hyperplanes also enable inter-project pipeline differentiation for larger portfolio scenarios involving multiple pipelines. The bottom of each run plane encloses a directed graph of sequential stages (cubes) of the pipeline between a start (black sphere) and an end (black sphere), while vertically stacked smaller cubes linked with lines above each stage show the internal *steps* within a stage. A cube with black borders is used to represent the entire run, and is all that is shown when a run is collapsed (e.g., to reduce visual clutter); on its front various details are depicted (ID, run duration, circular percentage of stages with status). The visualization form remains consistent across DevOps tools.

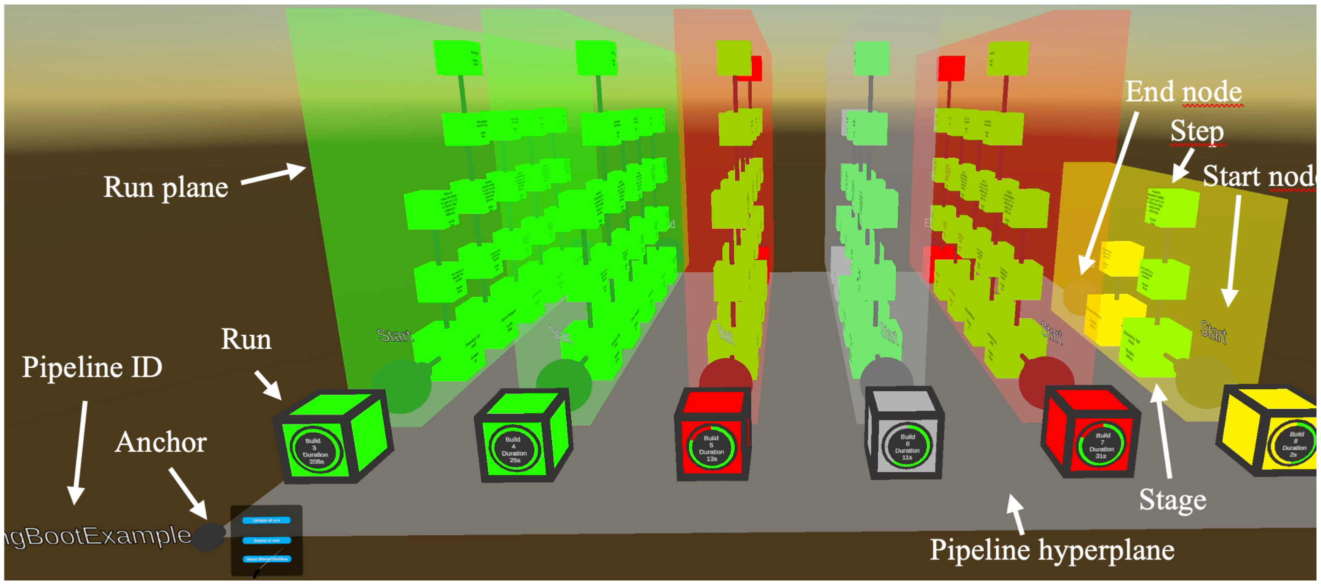


Figure 2. Hyperplane (annotated) of SprintBootExamaple Jenkins pipeline showing vertical colored run planes on a pipeline hyperplane.

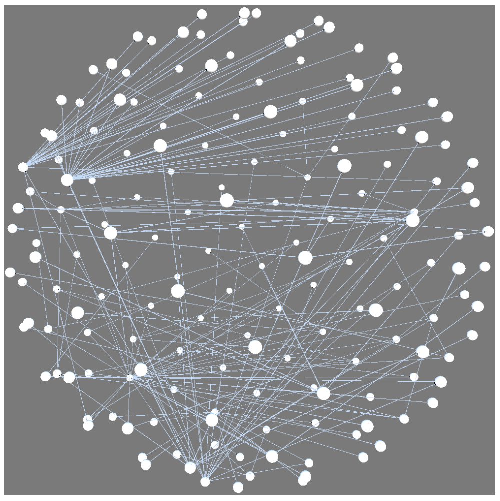


Figure 3. A nexus portraying artifact dependencies in a Git repository.

A Git code repository is depicted as a 3D graph, whereby the spherical nodes, representing artifacts (files), are placed on the surface of a 3D *nexus* sphere and edges connecting the nodes represent dependencies, as shown in Figure 3. Such a 3D nexus is spatially compact for navigating to and between nodes with the inside showing all dependencies. A 3D *radial tree* layout depicts child nodes as hierarchically-layered dependencies; here, all child nodes and their dependencies extend radially outwards from the nexus, leaving only the highest-level (parent or independent) artifacts in the nexus (reducing the number of nodes within the nexus), as shown in Figure 4. This permits dependency groupings and levels to be more easily followed. A separate Lines of Code (LOC) nexus is used to depict the relative sizes of source code files, as this graph structure represents a containing folders tree hierarchy rather than code dependencies; the node size corresponds to the LOC in that file for a selected commit. This boundary box for depicting LOC can stand for any metric of interest and could easily be switched to any other. We used LOC to exemplify this, since all text files have this relevant metric.

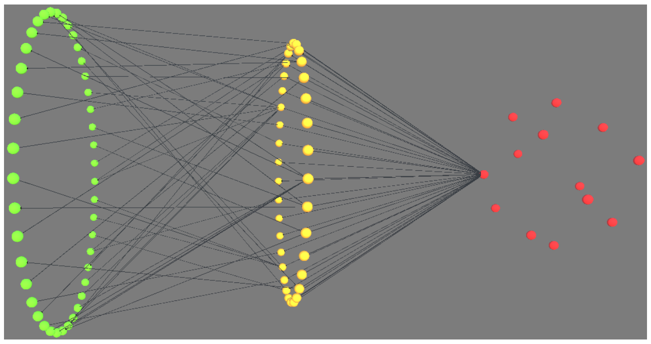


Figure 4. Child dependencies extracted as 3D radial tree from nexus.

D. *Navigation in VR*

Two navigation modes are incorporated in our solution: default gliding controls for fly-through VR, while teleporting instantly places the camera at a selected position via a selection on the VR-Tablet. Teleporting is potentially disconcerting, but may reduce the likelihood of VR sickness.

E. *Interaction in VR*

User-element interaction is supported primarily through VR controllers and a *VR-Tablet*. The VR-Tablet is used to provide detailed context-specific element information. Accordion visual elements permit more detailed information to be offered when desired. It includes a *virtual keyboard* for text entry via laser pointer key selection. On a hyperplane corner, an *anchor sphere* affordance (labeled with its pipeline ID) supports moving, hiding (collapsing), or showing (expanding) hyperplanes, as shown in the bottom left of Figure 2.

IV. Realization

The logical architecture for our prototype realization is shown in Figure 5. In our realization, the VR visualization aspects were implemented using Unity. It is supported by a Data Hub implemented in Python that integrates various Git and CI/CD data sources and stores them in JSON. All integrations with DevOps tools use their Web APIs via our tool-specific Adapters in our Data Hub for data conversion into our internal JSON format. MongoDB (locally or remotely using Atlas) is used for storage and is accessed via the MongoDB .NET/C# Driver from Unity or PyMongo from Python. To demonstrate the CI/CD tool/platform independence (i.e., heterogeneity) of our solution concept, it integrates with a local Jenkins [53] pipeline running in Docker, exemplifying a private cloud CI/CD server, as well as remote Semaphore [54] and Drone [55] CI servers to exemplify public cloud CI/CD tool integration using Web APIs.

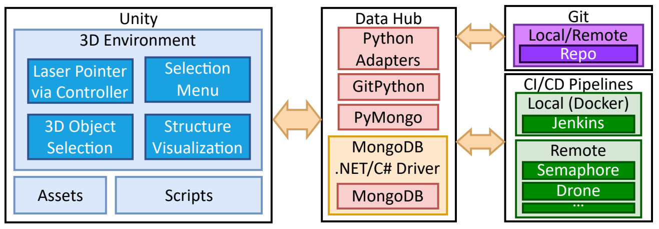


Figure 5. VR-GitEvo+CI/CD logical architecture.

The GitPython library is used to extract Git commit data: commit hash, author, timestamp, message, changed files, and changed files and metrics (insertions, deletions, lines). For Git commits, within the nexus, nodes outlined in red indicate new files while turquoise indicates changed files. Dependencies within JavaScript projects were extracted using the Node.js package manager "npm ls --all" command. For the 3D radial tree visualization, the various depth layers are colored to help differentiate them (yellow, light green, dark green, turquoise, blue, purple, pink, etc.). In the nexus layout, to highlight files affected by a commit, red indicates added and turquoise updated. In the LOC boundary box, black nodes are directories and files are red.

A CD pipeline is an automated expression of the process for preparing software for delivery. A Jenkins pipeline is a set of Jenkins plugins with a set of instructions specified in a text-based Jenkins file using Groovy syntax. It can be written in a scripted or declarative syntax, and typically defines the entire build process, including building, testing, and delivery. Concepts involved can include agents (executors), nodes (machines), stages (subset of tasks), and steps (a single task). Both Semaphore and Drone pipelines are described via a YAML syntax. We created our own common generic JSON format to store pipeline information, see Figure 6. A pipeline instance refers to a run. The refresh rates can be configured for Data Hub state retrieval from Unity and for each Adapter's Web APIs calls.

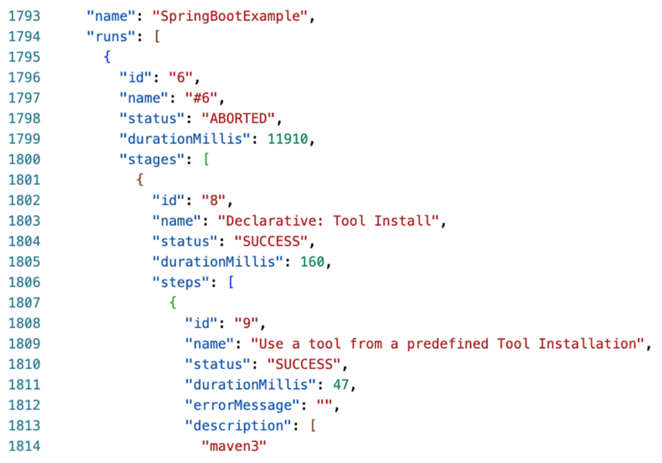

```
    "name": "SpringBootExample",
    "runs": [
      {
        "id": "6",
        "name": "#6",
        "status": "ABORTED",
        "durationMillis": 11910,
        "stages": [
          {
            "id": "8",
            "name": "Declarative: Tool Install",
            "status": "SUCCESS",
            "durationMillis": 160,
            "steps": [
              {
                "id": "9",
                "name": "Use a tool from a predefined Tool Installation",
                "status": "SUCCESS",
                "durationMillis": 47,
                "errorMessage": "",
                "description": [
                  "maven3"
```

Figure 6. Snippet of VR-GitEvo+CI/CD common run representation in JSON.

V. Evaluation

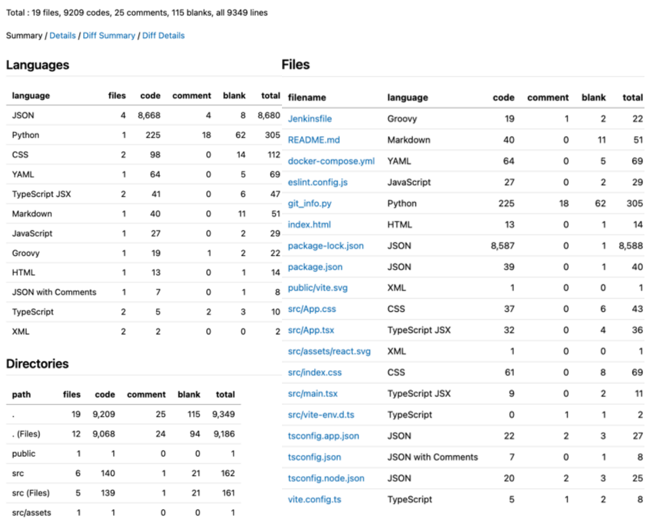
For the evaluation of our solution concept, we refer to design science method and principles [56], in particular a viable artifact, problem relevance, and design evaluation (utility, quality, efficacy). A scenario-based case study is used in evaluating the Git and CI/CD pipeline aspects separately to address these particular stakeholder concerns (for various stakeholders, not just developers):

1) *Status scenario:* focuses primarily on conveying status information, so that stakeholders can readily determine the current state,
2) *Analysis scenario:* focuses primarily on supporting analysis and investigation tasks via the provisioning of information towards understanding essential features, relations, constituent elements, issue identification, issue resolution, etc., and
3) *Evolution scenario:* focuses primarily on supporting comprehension of the evolution of the underlying structure (Git repository or CI/CD pipeline) via the provisioning of time-based change information to support comprehension regarding structural differences.

A. Git Code Repositories

1) Evaluation Setup

A very simple Vite-based HelloWorld Node.js app was used for the evaluation, the metrics shown in Figure 7. The app includes 477 Node modules that consist of 18K files and 1.8M LOC. Since one is often oblivious to all included modules, perhaps explicitly including some, which in turn include multiple others, we utilize Node.js to demonstrate the VR-GitEvo+CI/CD concept, in particular the 3D hierarchical radial tree, since such modules exhibit a much deeper dependency hierarchy then what is explicitly specified via inclusion. If modules or their dependencies are not of interest, then these can be hidden.


Total : 19 files, 9209 codes, 25 comments, 115 blanks, all 9349 lines

Summary / Details / Diff Summary / Diff Details

Languages

language	files	code	comment	blank	total
JSON	4	8,668	4	8	8,680
Python	1	225	18	62	305
CSS	2	98	0	14	112
YAML	1	64	0	5	69
TypeScript JSX	2	41	0	6	47
Markdown	1	40	0	11	51
JavaScript	1	27	0	2	29
Groovy	1	19	1	2	22
HTML	1	13	0	1	14
JSON with Comments	1	7	0	1	8
TypeScript	2	5	2	3	10
XML	2	2	0	0	2

Directories

path	files	code	comment	blank	total
.	19	9,209	25	115	9,349
. (Files)	12	9,068	24	94	9,186
public	1	1	0	0	1
src	6	140	1	21	162
src (Files)	5	139	1	21	161
src/assets	1	1	0	0	1

Files

filename	language	code	comment	blank	total
Jenkinsfile	Groovy	19	1	2	22
README.md	Markdown	40	0	11	51
docker-compose.yml	YAML	64	0	5	69
eslint.config.js	JavaScript	27	0	2	29
git_info.py	Python	225	18	62	305
index.html	HTML	13	0	1	14
package-lock.json	JSON	8,587	0	1	8,588
package.json	JSON	39	0	1	40
public/vite.svg	XML	1	0	0	1
src/App.css	CSS	37	0	6	43
src/App.tsx	TypeScript JSX	32	0	4	36
src/assets/react.svg	XML	1	0	0	1
src/index.css	CSS	61	0	8	69
src/main.tsx	TypeScript JSX	9	0	2	11
src/vite-env.d.ts	TypeScript	0	1	1	2
tsconfig.app.json	JSON	22	2	3	27
tsconfig.json	JSON with Comments	7	0	1	8
tsconfig.node.json	JSON	20	2	3	25
vite.config.ts	TypeScript	5	1	2	8

Figure 7. Git vite project language, directory, and file metrics.

2) Git Status Scenario

To determine the state of the Git repository, the LOC nexus on the right shows the files that are included in the project, the edges between them showing the relation between the containing directory and that file, with node size indicating relative LOC size, as shown on the right in Figure 8. In the boundary box on the left, the dependencies are shown between files and modules are shown, making apparent the many (often hidden) dependencies between included modules.

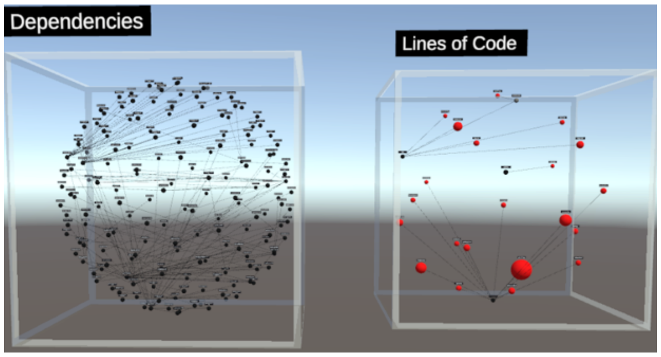
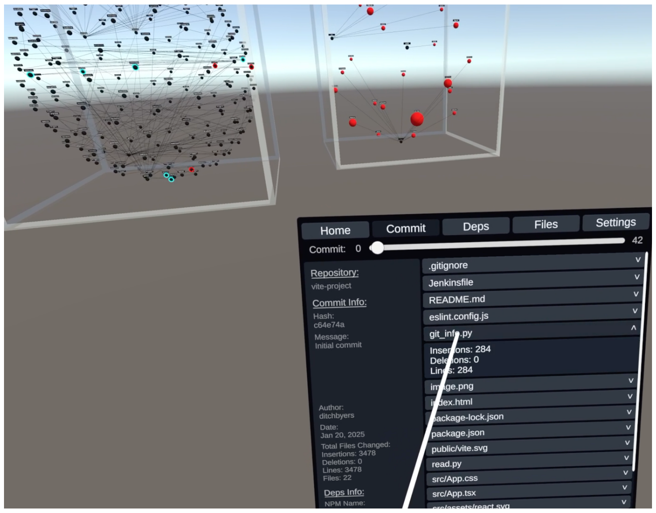


Figure 8. Git Dependency nexus (left) and LOC nexus (right).

To view details about the state of a specific commit, the Commit Tab on the VR-Tablet is shown in Figure 9. It provides commit status information as to the repository, specific commit info, author, date, total files and lines changed, and a scrollable list of the files making up the commit, which with an accordion can be expanded to show additional metrics of lines inserted, lines deleted, and total LOC size. The Dependencies Nexus on the top left shows highlighted nodes, red for files added and turquoise for files updated by this commit.



Commit: 0 42

Repository:
vite-project

Commit Info:
Hash:
c64e74a
Message:
Initial commit

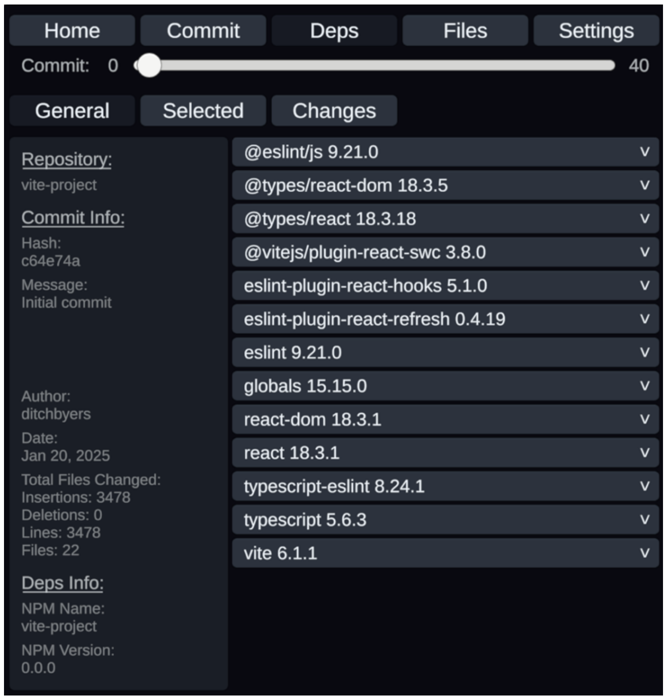
Author:
ditchbyers
Date:
Jan 20, 2025
Total Files Changed:
Insertions: 3478
Deletions: 0
Lines: 3478
Files: 22

Deps Info:
NPM Name:

.gitignore
Jenkinsfile
README.md
eslint.config.js
git_info.py
Insertions: 284
Deletions: 0
Lines: 284
image.png
index.html
package-lock.json
package.json
public/vite.svg
read.py
src/App.css
src/App.tsx
src/assets/react.svg

Figure 9. VR-Tablet: Git Commit Tab: Details.

The Dependencies Tab on the VR-Tablet is shown in Figure 10. It lists the explicit (first degree) included modules, and, using the accordion, one can drill down and, in turn, determine the included modules of each of these.



Commit: 0 40

Repository:
vite-project
Commit Info:
Hash:
c64e74a
Message:
Initial commit

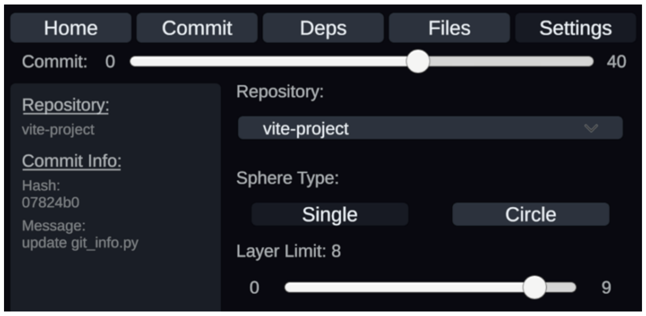
Author:
ditchbyers
Date:
Jan 20, 2025
Total Files Changed:
Insertions: 3478
Deletions: 0
Lines: 3478
Files: 22

Deps Info:
NPM Name:
vite-project
NPM Version:
0.0.0

@eslint/js 9.21.0
@types/react-dom 18.3.5
@types/react 18.3.18
@vitejs/plugin-react-swc 3.8.0
eslint-plugin-react-hooks 5.1.0
eslint-plugin-react-refresh 0.4.19
eslint 9.21.0
globals 15.15.0
react-dom 18.3.1
react 18.3.1
typescript-eslint 8.24.1
typescript 5.6.3
vite 6.1.1

Figure 10. VR-Tablet: Git Dependencies Tab: General status.

The Settings Tab on the VR-Tablet is shown in Figure 11. With it, the repository, the boundary box layout, and the depth limit can be specified. For a different repository (vite-project2), the sliders max layer limit and number of commits available are adjusted accordingly, as shown in Figure 12.



Commit: 0 40

Repository:
vite-project

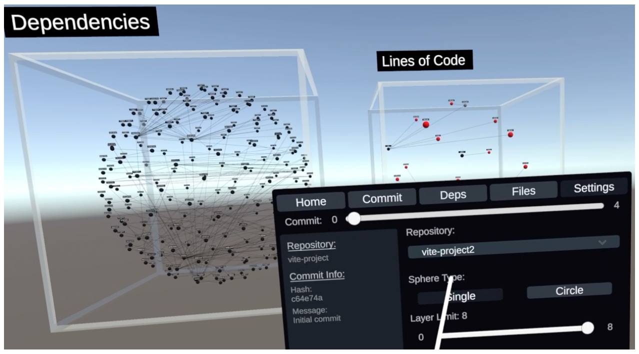
Commit Info:
Hash:
07824b0
Message:
update git_info.py

Repository:
vite-project

Sphere Type:
Single Circle

Layer Limit: 8
0 9

Figure 11. VR-Tablet: Git Settings Tab: selected repo, layout, and layer limit option status.



Commit: 0 4

Repository:
vite-project

Commit Info:
Hash:
c64e74a
Message:
Initial commit

Repository:
vite-project2

Sphere Type:
Single Circle

Layer Limit: 8
0 8

Figure 12. VR-Tablet: Git Settings Tab: selecting another repo adjusts commits available.

3) Git Analysis Scenario

To support analysis of Git repositories, multiple options are offered. By default, the Dependency Nexus shows all dependencies to make one aware of the entire set of dependencies, as shown in Figure 13.

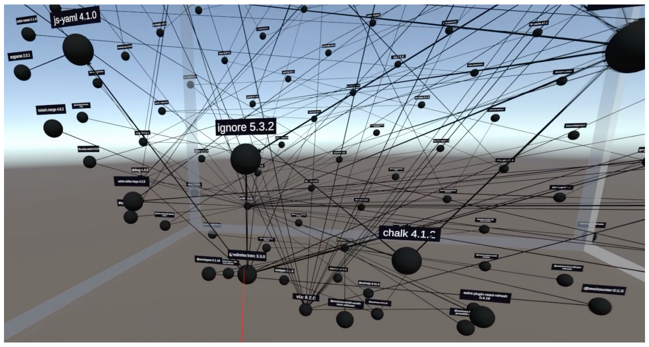
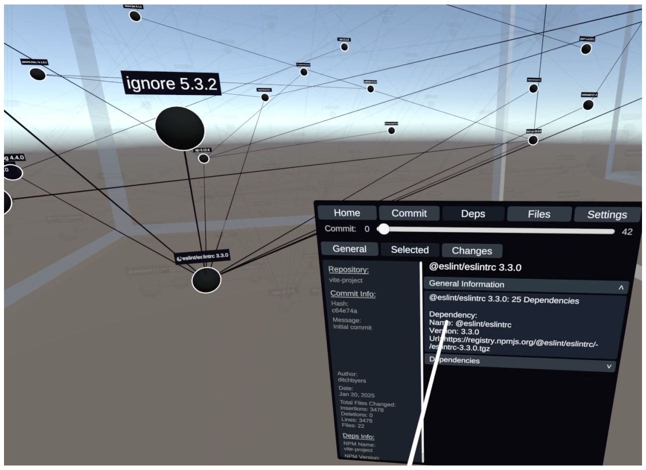


Figure 13. Git: Nexus: closeup of dependencies before node selection.

When a single node is selected, non-related dependencies and nodes are ghosted (to minimize visual clutter) and help focus on its specific dependencies, as shown in Figure 14. Moreover, the Deps Tab switches to Selected and offers specific information on that node and its dependencies. A view from further out shows the highlighted selected nodes and the ghosted can still be faintly seen, as depicted in Figure 15.



Commit: 0 42

@eslint/eslintrc 3.3.0

General Information

@eslint/eslintrc 3.3.0: 25 Dependencies

Dependency:
Name: @eslint/eslintrc
Version: 3.3.0
Url: https://registry.npmjs.org/@eslint/eslintrc/-/eslintrc-3.3.0.tgz

Dependencies

Figure 14. Git: Nexus: selecting a node leaves tree of dependencies and toggles ghosting of rest; VR-Tablet offers detailed information.

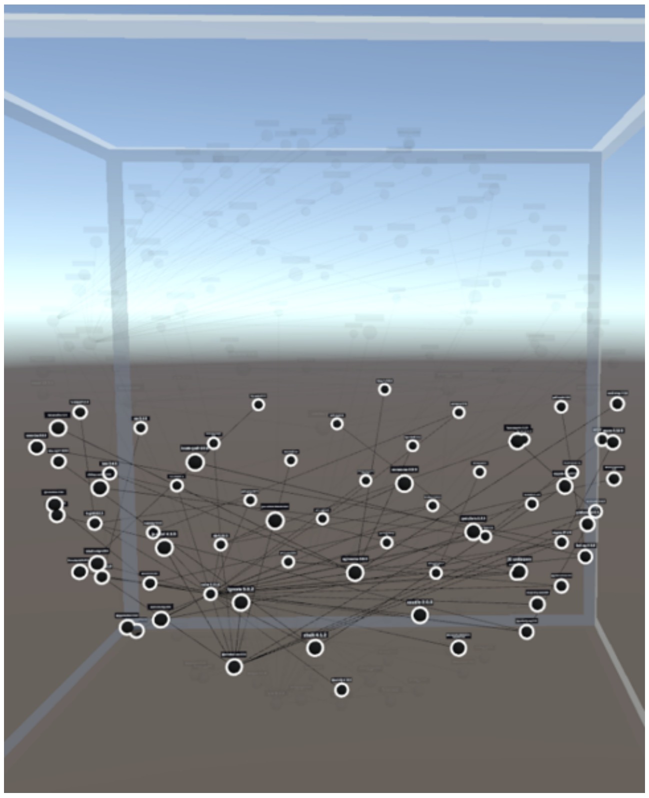


Figure 15. Git: Nexus: selected dependencies and rest of nexus ghosted.

Contingent on the depth of the dependencies, it may be beneficial to view the dependencies hierarchically and limiting the layers, as shown with our 3D radial tree layout (Circle in VR-Tablet) with the slider set to 2 layers of depth, as shown in Figure 16.

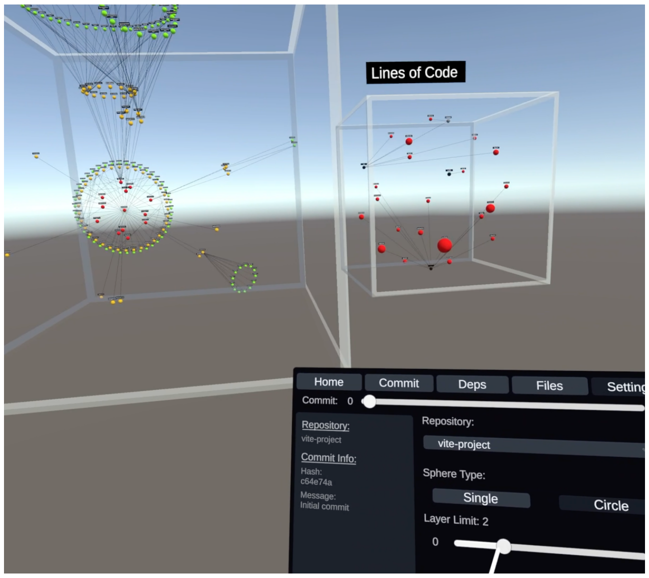


Figure 16. Git: 3D Radial Tree (Circle) layout with depth limit 2.

In contrast, with the setting to its maximum depth (reaches 9 for this repo at a certain commit date) is shown in Figure 17. These dependencies can be hierarchically navigated, with certain nodes having n-m incoming and outgoing dependencies.

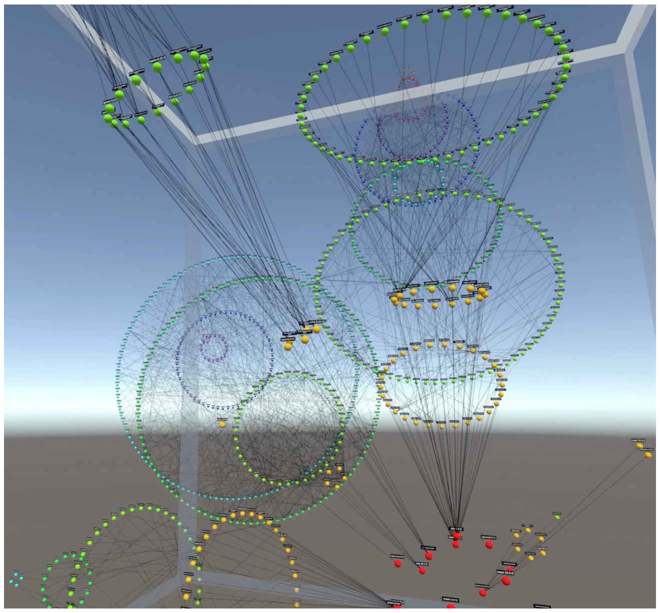


Figure 17. Git: 3D Radial (Circle) with depth limit 9.

Furthermore, to assist with analysis, a subset of dependencies can be highlighted (ghosting other nodes and dependencies) by selecting a specific node of interest, as exemplified with the jest node in Figure 18.

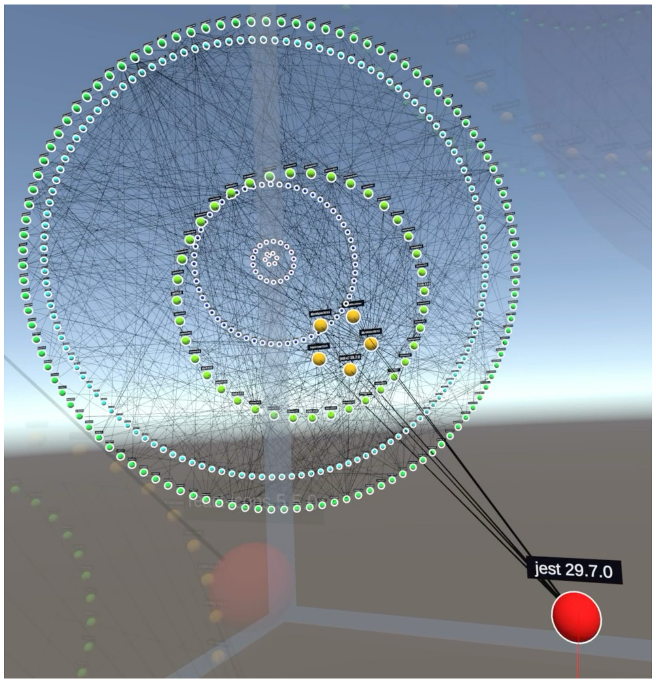
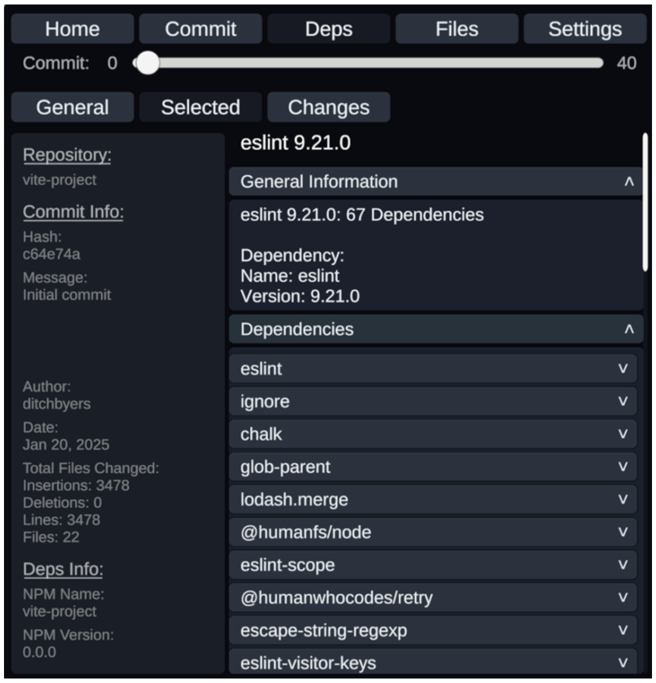


Figure 18. Git: Radial Layout: selecting node shows dependencies and toggles ghosting of rest.

A subset of dependencies can be highlighted (ghosting other nodes and dependencies) by selecting a specific node of interest, as exemplified with the jest node in Figure 18. This can also be done by selecting an element of interest via the VR-Tablet. The full set of dependencies can also be viewed and navigated in the VR-Tablet, as shown Figure 19.



Home | Commit | Deps | Files | Settings

Commit: 0 40

General | Selected | Changes

Repository:
vite-project

Commit Info:
Hash:
c64e74a
Message:
Initial commit

Author:
ditchbyers
Date:
Jan 20, 2025
Total Files Changed:
Insertions: 3478
Deletions: 0
Lines: 3478
Files: 22

Deps Info:
NPM Name:
vite-project
NPM Version:
0.0.0

eslint 9.21.0

General Information ^

eslint 9.21.0: 67 Dependencies

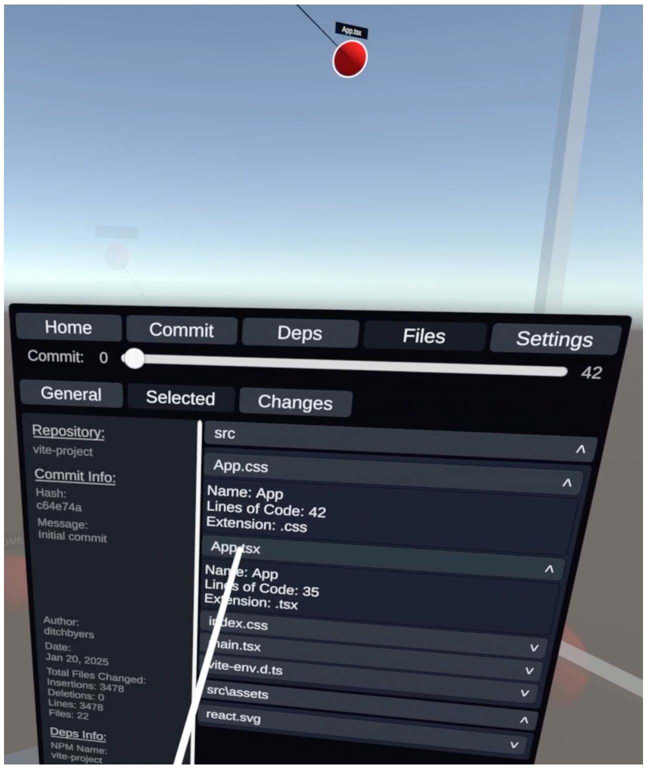
Dependency:
Name: eslint
Version: 9.21.0

Dependencies ^

eslint v
ignore v
chalk v
glob-parent v
lodash.merge v
@humanfs/node v
eslint-scope v
@humanwhocodes/retry v
escape-string-regexp v
eslint-visitor-keys v

Figure 19. VR-Tablet: Git Dependencies Tab: Selected Dependencies.

If one wishes to analyze the size of the files (or any other metric of interest if it were implemented), one can select a node of interest in the nexus in the LOC boundary box and the corresponding data is then shown in the VR-Tablet, as seen in Figure 20. The relation of a file (red node) to its containing directory (black nodes) is shown via edges within the nexus, while the relative size is conveyed via the sphere size, as shown in Figure 21.



Repository:
vite-project
Commit Info:
Hash:
c64e74a
Message:
Initial commit
Author:
ditchbyers
Date:
Jan 20, 2025
Total Files Changed:
Insertions: 3478
Deletions: 0
Lines: 3478
Files: 22
Deps Info:
NPM Name:
vite-project

src
App.css
Name: App
Lines of Code: 42
Extension: .css
App.tsx
Name: App
Lines of Code: 35
Extension: .tsx
index.css
main.tsx
vite-env.d.ts
src\assets
react.svg

Figure 20. VR-Tablet: Git Files Tab: Files contained in selected src folder.

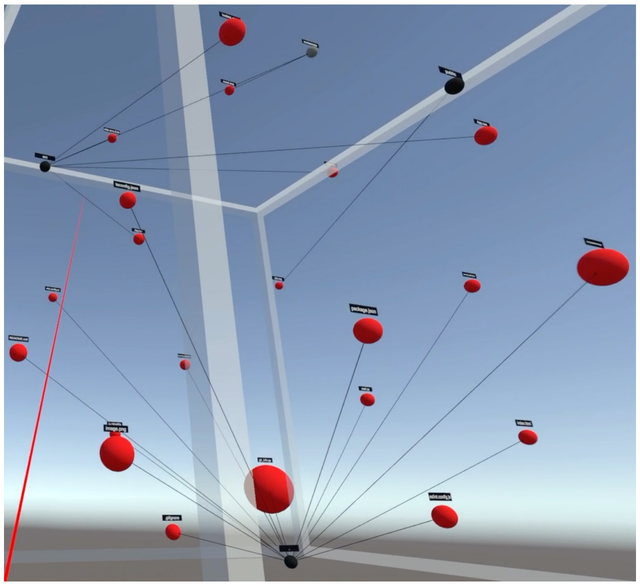
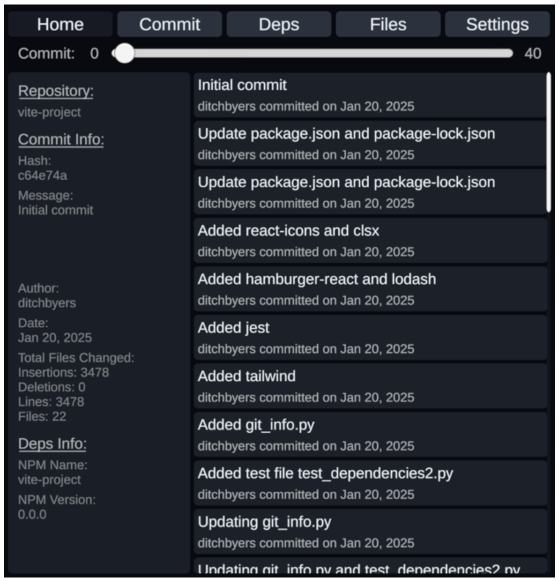


Figure 21. Git: Lines of Code nexus.

4) *Git Evolution Scenario*

For the evolution scenario, changes and time are of interest to the stakeholders. For this, the VR-Tablet offers on the Home Tab a list of all commits with the commit messages, author, date, and a slider showing the total number of commits, as shown in Figure 22. By moving this slider, the contents in the boundary boxes are redrawn to show the state at that evolution point and the changes to the repo by that commit, as shown in Figure 23. Hence, by sliding the slider, the history of changes of these nexuses are redrawn and thus animated and changes are dynamically apparent. Similarly, the changes to dependencies can be viewed as a 3D radial tree and time-adjusted, as shown in Figure 24.



Repository:
vite-project
Commit Info:
Hash:
c64e74a
Message:
Initial commit
Author:
ditchbyers
Date:
Jan 20, 2025
Total Files Changed:
Insertions: 3478
Deletions: 0
Lines: 3478
Files: 22
Deps Info:
NPM Name:
vite-project
NPM Version:
0.0.0

Initial commit
ditchbyers committed on Jan 20, 2025
Update package.json and package-lock.json
ditchbyers committed on Jan 20, 2025
Update package.json and package-lock.json
ditchbyers committed on Jan 20, 2025
Added react-icons and clsx
ditchbyers committed on Jan 20, 2025
Added hamburger-react and lodash
ditchbyers committed on Jan 20, 2025
Added jest
ditchbyers committed on Jan 20, 2025
Added tailwind
ditchbyers committed on Jan 20, 2025
Added git_info.py
ditchbyers committed on Jan 20, 2025
Added test file test_dependencies2.py
ditchbyers committed on Jan 20, 2025
Updating git_info.py
ditchbyers committed on Jan 20, 2025
Updating git_info.py and test_dependencies2.py

Figure 22. VR-Tablet: Home Tab: Git commit messages

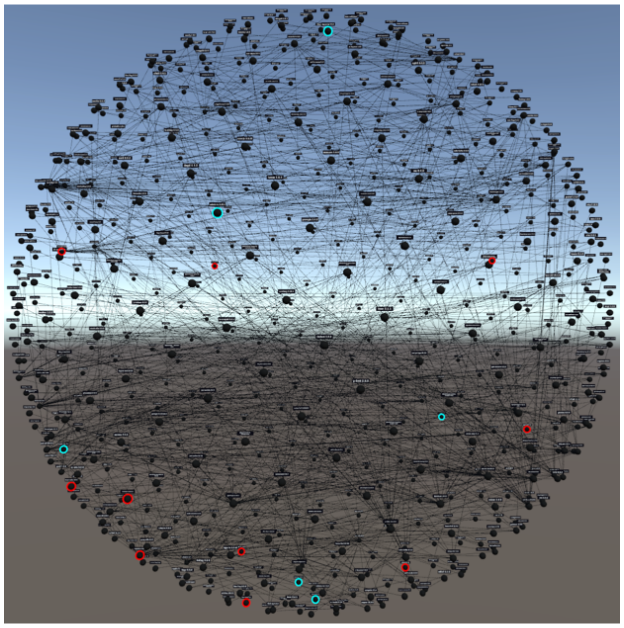


Figure 23. Git: Nexus: new (red) and updated (turquoise) nodes in commit.

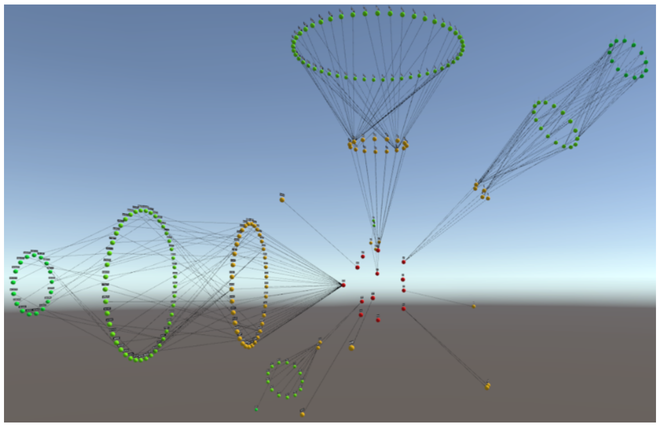
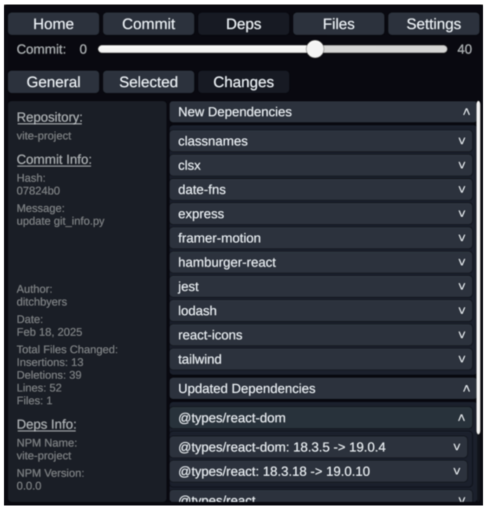


Figure 24. Git: 3D radial: dependencies as 3D radial tidy trees.

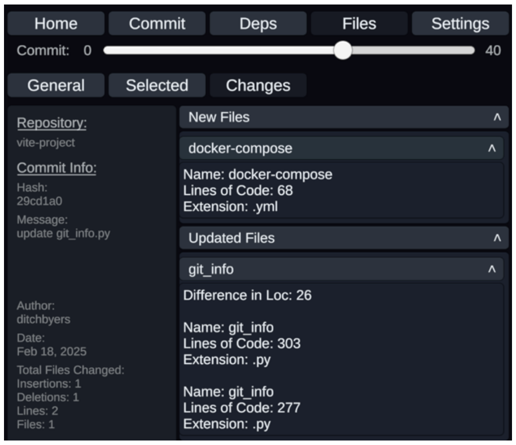
The detailed changes to dependencies can also be viewed in the VR-Tablet in the Deps Tab under Changes providing further details, as shown in Figure 25. If more interested in File changes, then in the VR-Tablet in the Files Tab under Changes further details are provided there, including changes to metrics, as shown in Figure 26.



Repository:
vite-project
Commit Info:
Hash:
07824b0
Message:
update git_info.py
Author:
ditchbyers
Date:
Feb 18, 2025
Total Files Changed:
Insertions: 13
Deletions: 39
Lines: 52
Files: 1
Deps Info:
NPM Name:
vite-project
NPM Version:
0.0.0

New Dependencies
classnames
clsx
date-fns
express
framer-motion
hamburger-react
jest
lodash
react-icons
tailwind
Updated Dependencies
@types/react-dom
@types/react-dom: 18.3.5 -> 19.0.4
@types/react: 18.3.18 -> 19.0.10
@types/react

Figure 25. VR-Tablet: Git Dependencies Tab: Changes.



Repository:
vite-project
Commit Info:
Hash:
29cd1a0
Message:
update git_info.py
Author:
ditchbyers
Date:
Feb 18, 2025
Total Files Changed:
Insertions: 1
Deletions: 1
Lines: 2
Files: 1

New Files
docker-compose
Name: docker-compose
Lines of Code: 68
Extension: .yml
Updated Files
git_info
Difference in Loc: 26

Name: git_info
Lines of Code: 303
Extension: .py

Name: git_info
Lines of Code: 277
Extension: .py

Figure 26. VR-Tablet: Git Files Tab: Changes.

B. *CI/CD Pipelines*

To demonstrate the heterogeneity of the solution, various screenshots of runs from Jenkins, Semaphore, or Drone are used interchangeably.

1) *Evaluation Setup*

For Jenkins, the SpringBoot PetClinic example pipeline [57] includes 39 Java files and 1335 Lines of Code (LOC). For pipeline error illustration purposes, an additional step with an artificial error was inserted into the SpringBoot example in a second version of the CI/CD pipeline, as shown in the listing in Figure 27. For Semaphore, the Android App example pipeline includes 13 Kotlin files and 287 LOC, as shown in the listing in Figure 28.

```
pipeline {
    agent any
    tools {
        maven 'maven3'
    }
    options {
        buildDiscarder logRotator(
                    daysToKeepStr: '15',
                    numToKeepStr: '10'
            )
    }
    environment {
        APP_NAME = "DCUBE_APP"
        APP_ENV  = "DEV"
    }
    stages {
        stage('Cleanup Workspace') {
            steps {
                cleanWs()
                sh """
                echo "Cleaned Up Workspace for ${APP_NAME}"
                """
            }
        }
        stage('Code Checkout') {
            steps {
                checkout([
                    $class: 'GitSCM',
                    branches: [[name: '*/main']],
                    userRemoteConfigs: [[url: 'https://github.com/spring-projects/spring-petclinic.git']]
                ])
            }
        }
        stage('Code Build') {
            steps {
                 sh 'mvn install -Dmaven.test.skip=true'
            }
        }
        stage('Printing All Global Variables') {
            steps {
                sh """
                env
                """
                error "Artificial Error"
```

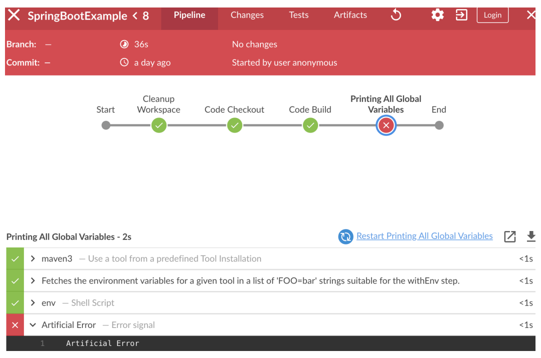
Figure 27. SpringBoot PetClinic Jenkins pipeline in Groovy (snippet).

2) *CI/CD Pipeline Status Scenario*

In 2D, dashboards are typically available for assessing a selected CI/CD pipeline instance state, yet each tool has its own unique interface, as exemplified for the Jenkins tool in Figure 29. The equivalent for Semaphore is shown in Figure 30. In our VR-GitEvo+CI/CD, a unified interface for heterogeneous CI/CD pipelines is provided, such that a stakeholder can readily comprehend and assess the current status and state of multiple pipeline runs. Any particular run may execute different steps and stages (e.g., due to an abort, error, option, etc.). Fully collapsed run planes provide a high-level overview, with black-lined cubes representing any pipeline instance and conveying details, as shown in Figure 34 and Figure 35.

```yaml
version: v1.0
name: Verification Pipeline
agent:
  machine:
    type: e1-standard-2
    os_image: ubuntu2004
  containers:
    - name: main
      image: 'registry.semaphoreci.com/android:29'
global_job_config:
  env_vars:
    - name: ADB_INSTALL_TIMEOUT
      value: '10'
  prologue:
    commands:
      - checkout
      - cache restore gradle-wrapper
      - cache restore gradle-cache
      - cache restore android-build
blocks:
  - name: Build
    task:
      jobs:
        - name: Build Project
          commands:
            - ./gradlew bundleDebug
      epilogue:
        on_pass:
          commands:
            - cache clear
            - cache store gradle-wrapper ~/.gradle/wrapper
            - cache store gradle-cache ~/.gradle/caches
            - cache store android-build ~/.android/build-cache
  - name: Verification
    task:
      jobs:
        - name: Analyze Code
          commands:
            - ./gradlew lint
        - name: Unit Tests
          commands:
            - ./gradlew testDebugUnitTest
      epilogue:
        always:
          commands:
            - artifact push job --expire-in 2w --destination reports/ app/build/reports/
promotions:
  - name: Start Integration tests
    pipeline_file: integration-tests.yml
```

Figure 28. Semaphore Android App pipeline in YAML format


SpringBootExample < 8 | Pipeline | Changes | Tests | Artifacts | Login

Branch: – 36s No changes
Commit: – a day ago Started by user anonymous

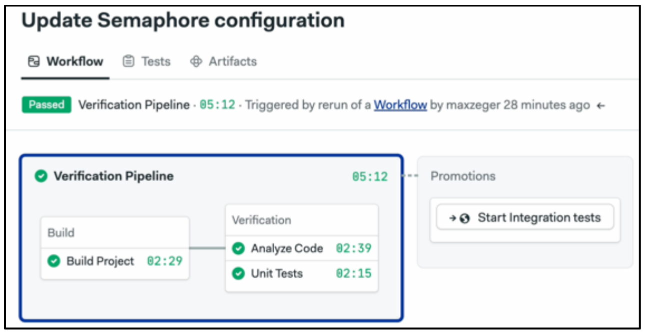
Start — Cleanup Workspace — Code Checkout — Code Build — Printing All Global Variables — End

Printing All Global Variables - 2s Restart Printing All Global Variables

✓	maven3 — Use a tool from a predefined Tool Installation	<1s
✓	Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step.	<1s
✓	env — Shell Script	<1s
✗	Artificial Error — Error signal	<1s

```
1   Artificial Error
```

Figure 29. Jenkins tool web screenshot.


Update Semaphore configuration

Workflow | Tests | Artifacts

Passed Verification Pipeline · 05:12 · Triggered by rerun of a Workflow by maxzeger 28 minutes ago

Verification Pipeline 05:12

Build
Build Project 02:29

Verification
Analyze Code 02:39
Unit Tests 02:15

Promotions
Start Integration tests

Figure 30. Semaphore tool web screenshot.

3) *CI/CD Pipeline Analysis Scenario*

VR-GitEvo+CI/CD supports analysis of issues via immersive visual patterns and contrasts, visually revealing differences in the detailed steps executions, as shown for the Android App in Figure 31. Here, each *run plane* represents a pipeline *run*, which is colored based on status (green=success, yellow=in progress, red=error, grey= aborted). The contrasts with a YAML (YAML Ain't Markup Language) pipeline definition, which contains many details that are difficult for certain stakeholders to grasp, while lacking status info, as in Figure 28. Alternatively, a web-based graphical status may be offered, yet lack comprehensive details for analysis, as seen in Figure 29 and Figure 30.

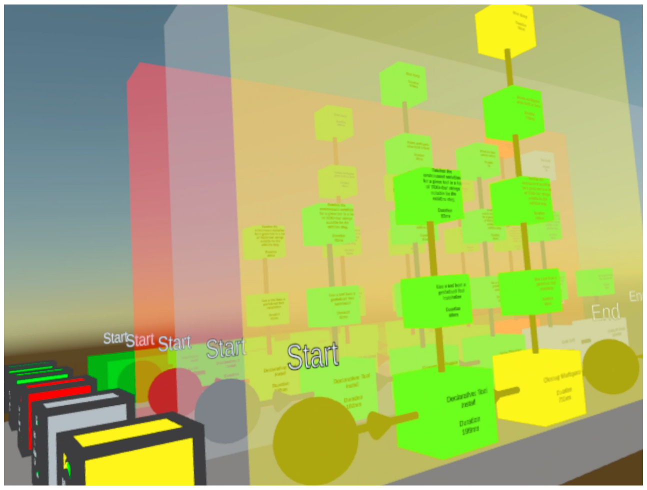
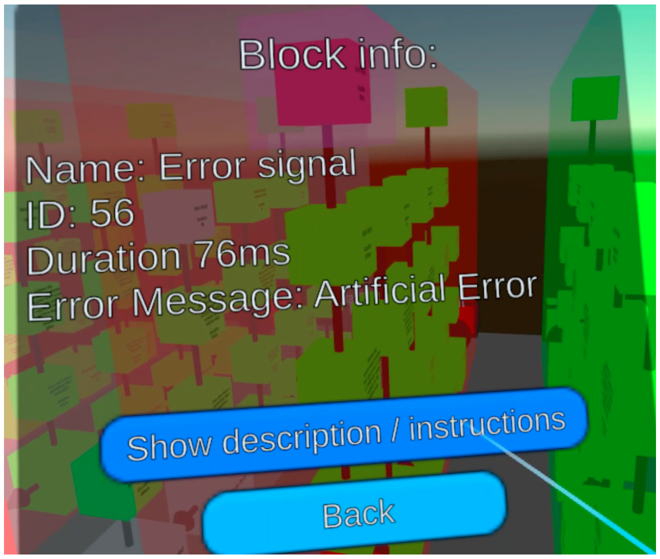


Figure 31. Immersive analysis via visual colored run comparison of stage/step status (for Semaphore pipeline).

Furthermore, to assist with analysis tasks, the VR-Tablet supports information retrieval, including additional context-specific *metadata* and *error messages* about a particular block as seen Figure 32.


Block info:

Name: Error signal
ID: 56
Duration 76ms
Error Message: Artificial Error

Figure 32. VR-Tablet shows contextual element details: metadata and instructions/description.

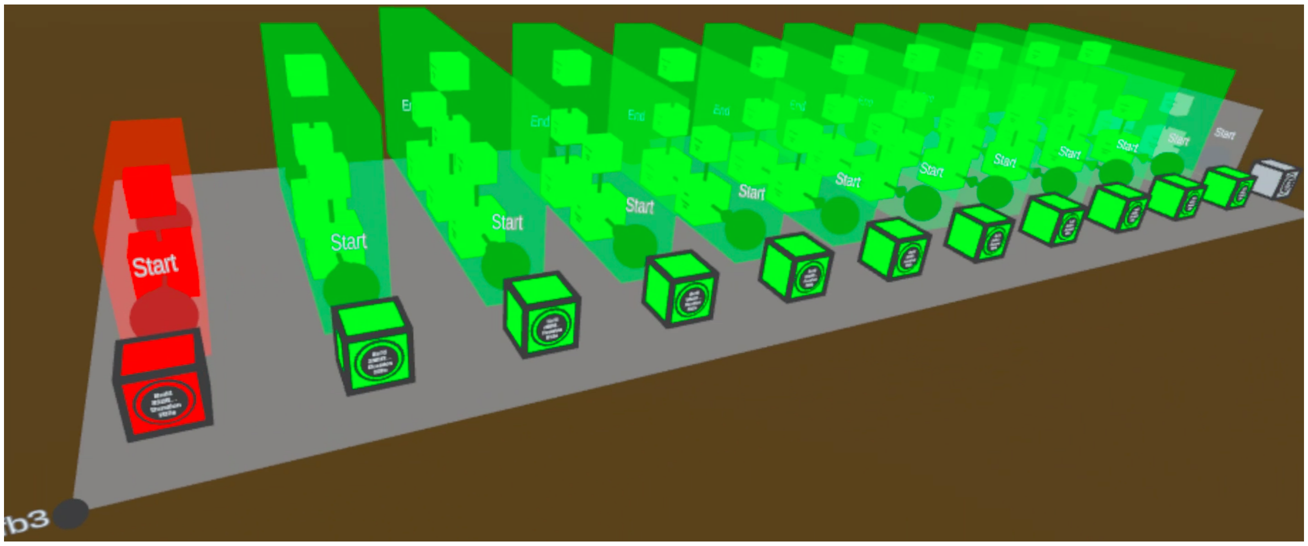


Figure 33. Pipeline run status for a set of Semaphore pipeline runs showing expanded step details and an aborted process in grey on the far right.

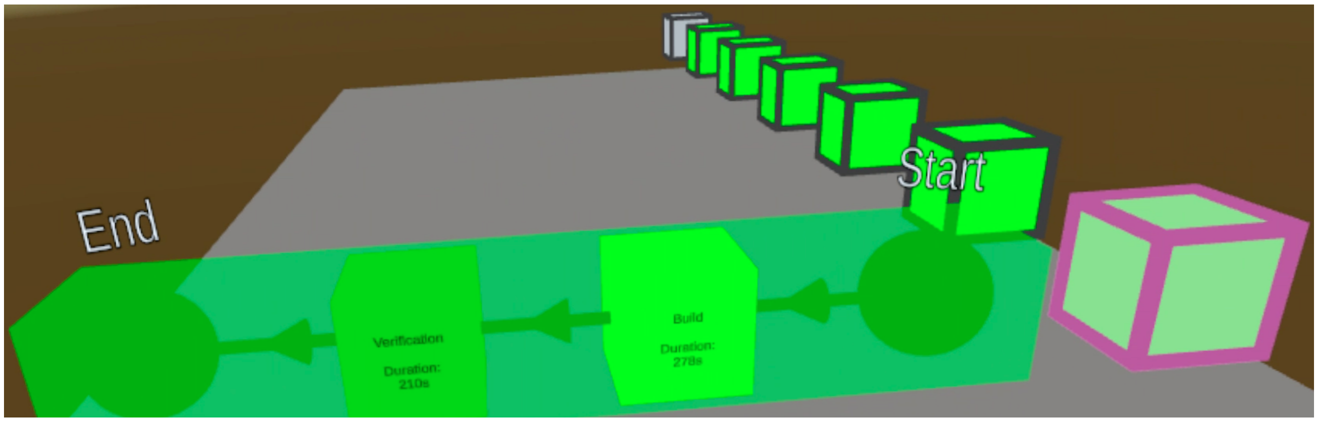
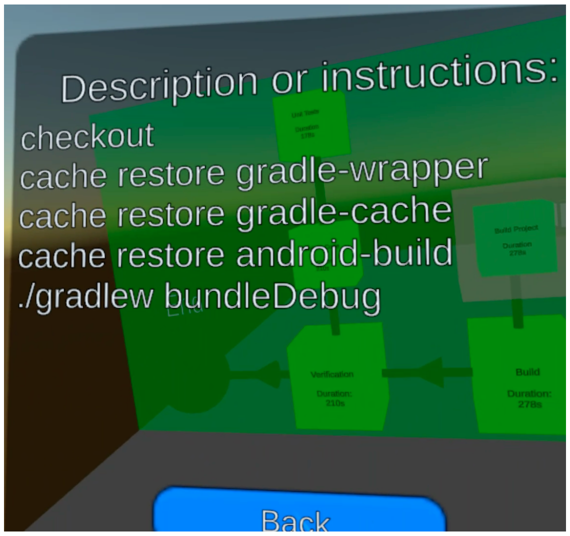


Figure 34. Collapsed Semaphore runs on a pipeline hyperplane with stages expanded (and steps collapsed) for a selected run.

The pipeline *stage* or *step task* instructions can be viewed, as shown in Figure 35.

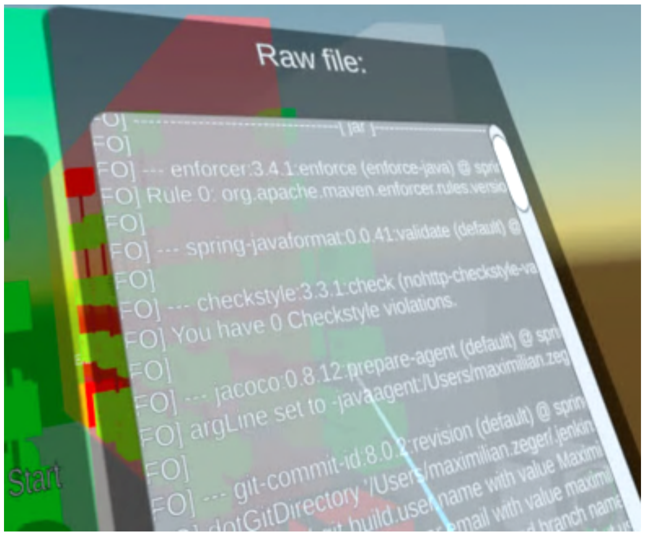


Description or instructions:

```
checkout
cache restore gradle-wrapper
cache restore gradle-cache
cache restore android-build
./gradlew bundleDebug
```

Figure 35. VR-Tablet shows contextual element details: instructions or description.

Raw pipeline *log* information is available, as shown in Figure 36.

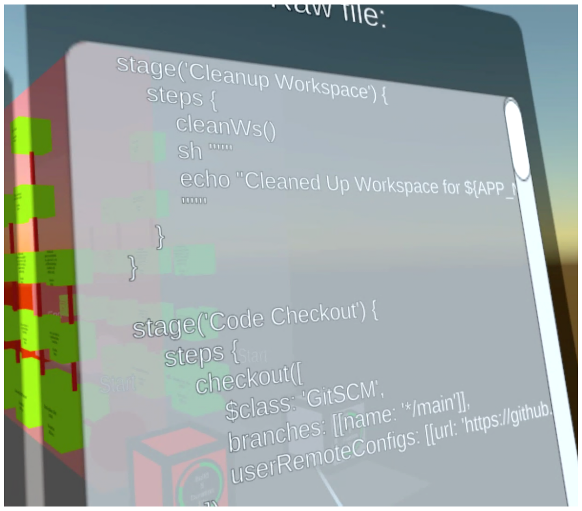


Raw file:

```
FO] ----------------------------------[ jar ]---------------------------------
FO]
FO] --- enforcer:3.4.1:enforce (enforce-java) @ spri
FO] Rule 0: org.apache.maven.enforcer.rules.versio
FO]
FO] --- spring-javaformat:0.0.41:validate (default) @
FO]
FO] --- checkstyle:3.3.1:check (nohttp-checkstyle-va
FO] You have 0 Checkstyle violations.
FO]
FO] --- jacoco:0.8.12:prepare-agent (default) @ spri
FO] argLine set to -javaagent:/Users/maximilian.zeg
FO]
FO] --- git-commit-id:8.0.2:revision (default) @ sprin
FO] dotGitDirectory '/Users/maximilian.zegerl.jenkin
    git.build.user.name with value Maximi
    git.build.user.email with value maximil
    branch name
```

Figure 36. VR-Tablet offers raw file access to log.

The raw *pipeline code specification* can also be viewed directly in the VR-Tablet, as seen in Figure 37.



```
Raw file:
stage('Cleanup Workspace') {
    steps {
        cleanWs()
        sh """
        echo "Cleaned Up Workspace for ${APP_
        """
    }
}

stage('Code Checkout') {
    steps {
        checkout([
            $class: 'GitSCM',
            branches: [[name: '*/main']],
            userRemoteConfigs: [[url: 'https://github.
```

Figure 37. VR-Tablet offers raw file access to pipeline definition.

This consolidation of information in the VR-Tablet, in conjunction with visual context in VR, could improve the utility and efficacy of analysis tasks, especially when considering increasing pipeline complexity, pipeline versions, and large scale-out of runs.

4) *CI/CD Pipeline Evolution Scenario*

To support pipeline evolution scenarios, changes and time are factors for stakeholder. To view versioning changes to stages or steps, an immersive visual differentiation of runs can be performed by choosing a perspective and alignment, as shown for the PetClinic pipeline in Figure 38. Comprehending the pipeline structure and any delta via its Groovy file would be more difficult for non-developer stakeholders. Visual depiction and differentiation can help support the inclusion of non-tech-savvy stakeholders, improving the speed of assessments and the quality of analysis tasks via the inclusion of diverse stakeholders.

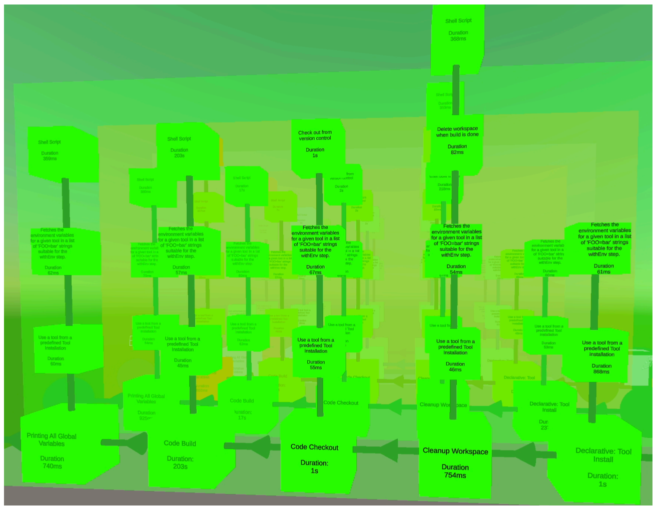


Figure 38. Immersive analysis of pipeline evolution via visual alignment of stage/steps (for Jenkins pipeline).

Moreover, any issues with pipelines over time can be readily viewed via the status of all instances. Run status details can be individually collapsed or expanded as desired for the analysis, vertically (Figure 33) and horizontally (Figure 34). The status of "All Builds" is shown in Figure 39. Via the VR-Tablet timeline slider, a specific pipeline instance can be selected. This is shown for a specific failure case in Figure 40. The equivalent for a successful case is shown in Figure 41.

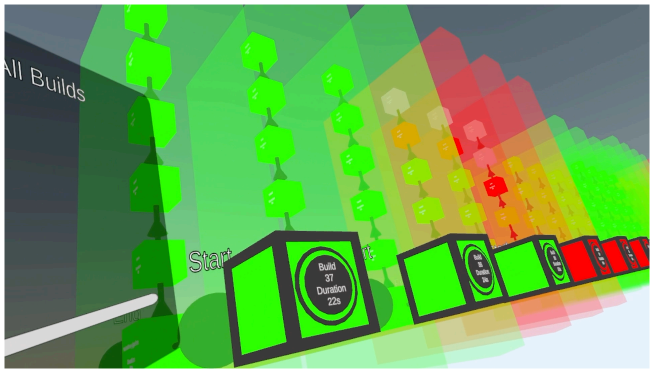


Figure 39. Timeline slider can be used to selectively view the history of all pipeline instances.

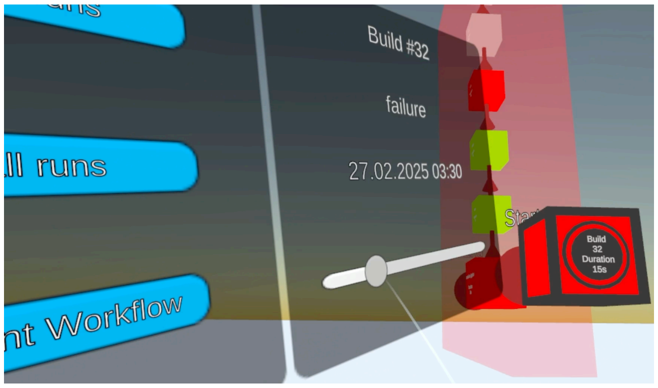


Figure 40. Timeline slider can be used to select a single execution.

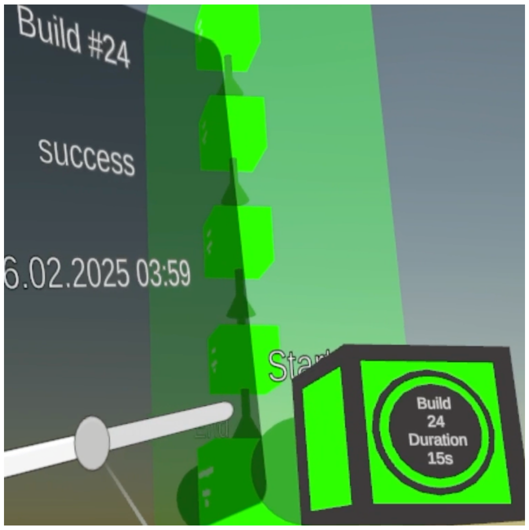


Figure 41. Timeline slider can be used to select a single.

C. Discussion

The scenario-based case study using our prototype realization showed the ability of the GitEvo+CI/CD solution concept to address DevOps stakeholders concerns regarding both Git codebase repositories and CI/CD pipelines. These scenarios included a status scenario, analysis scenario, and evolution scenario. The use of VR supports a collaborative, immersive experience without visual limitations, enabling it to scale across large projects and multiple projects concurrently with a relatively intuitive and simple homogeneous interface to diverse local and remote Git repository providers and CI/CD providers.

VI. Conclusion

Visualizing the evolution of both Git codebases and CI/CD pipelines remains a challenge and hinders comprehension, analysis, and collaboration among DevOps stakeholders. VR-GitEvo+CI/CD offers an immersive visualization solution concept for codebases, their dependencies, and CI/CD pipelines in VR. The realization prototype showed its feasibility, while the case-based evaluation showed its potential to support comprehension in typical scenarios such as status, analysis, and evolution. The solution concept is DevOps tool-independent, hiding the differences that the fragmented DevOps tool landscape might present to non-tech-savvy stakeholders. It thus provides a way towards broader inclusion of various DevOps stakeholders, and can thus support greater collaboration and communication to address a significant challenge facing DevOps.

For future work, we see the potential for more holistic DevOps insights via a deeper integration with our other existing VR solutions. Additional future work includes: support for GitOps, Infrastructure as Code, VR-native developer support, collaboration and annotation capabilities, and a comprehensive industrial empirical study.

Acknowledgment

The authors would like to thank Pascal Rene May, Maximilian Stricker, and Maximilian Zeger for his assistance with the design, implementation, and evaluation.

References

[1] R. Oberhauser, “VR-DevOps: Visualizing and Interacting with DevOps Pipelines in Virtual Reality,” In: Proc. 19th International Conference on Software Engineering Advances, pp. 43-48, 2024.

[2] F. Zampetti, S. Geremia, G. Bavota, and M. Di Penta, “CI/CD pipelines evolution and restructuring: A qualitative and quantitative study,” In: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2021, pp. 471-482.

[3] J. M. Robinson, “An introduction to early Greek philosophy: The chief fragments and ancient testimony, with connecting commentary,” Advanced Reasoning Forum, p. 90, Fragment 5.14, 2021.

[4] F. P. Brooks, Jr., *The Mythical Man-Month*, Boston, MA: Addison-Wesley Longman Publ. Co., Inc., 1995.

[5] Sonatype, “State of the Software Supply Chain,” 2024, https://sonatype.com/hubfs/SSCR-2024/SSCR_2024-FINAL-optimized.pdf 2025.05.10

[6] IT Revolution, “DevOps Guide: Selected Resources to Start Your Journey,” The IT Revolution, 2015. [Online]. Available from: https://web.archive.org/web/20211010072856/http://images.itrevolution.com/documents/ITRev_DevOps_Guide_5_2015.pdf 2025.05.10

[7] J. Micco, “Tools for continuous integration at google scale,” Google Tech Talk, Google Inc., 2012.

[8] DevOps Research and Assessment (DORA) Team, “Accelerate State of DevOps report,” 2021. [Online]. Available from: https://services.google.com/fh/files/misc/state-of-devops-2021.pdf 2025.05.10

[9] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," in IEEE Software, vol. 33, no. 3, pp. 94-100, May-June 2016.

[10] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect’s Perspective*, Addison-Wesley Professional, 2015.

[11] L. Dodd and B. Noll, “State of CI/CD Report 2024: The Evolution of Software Delivery Performance,” SlashData and the Continuous Delivery Foundation, 2024.

[12] GitLab, “A Maturing DevSecOps Landscape,” 2021. [Online]. Available from: https://about.gitlab.com/images/developer-survey/gitlab-devsecops-2021-survey-results.pdf 2025.05.10

[13] J. D'Souza, “GitHub Statistics by Developers, Git Pushes and Facts” [Online]. Available from: https://web.archive.org/web/20250226132029/https://www.coolest-gadgets.com/github-statistics/ 2025.05.10

[14] R. Potvin and J. Levenberg, “Why Google stores billions of lines of code in a single repository,” In: Communications of the ACM, 59(7), pp.78-87, 2016. https://doi.org/10.1145/2854146

[15] M. Tyson, “Linux kernel source expands beyond 40 million lines – it has doubled in size in a decade,” Tom's Hardware, January 26, 2025. [Online]. Available from: https://www.tomshardware.com/software/linux/linux-kernel-source-expands-beyond-40-million-lines-it-has-doubled-in-size-in-a-decade 2025.05.10

[16] M. S. Khan, A. W. Khan, F. Khan, M. A. Khan, and T. K. Whangbo, "Critical Challenges to Adopt DevOps Culture in Software Organizations: A Systematic Review," in IEEE Access, vol. 10, pp. 14339-14349, 2022.

[17] L. Giamattei et al., “Monitoring tools for DevOps and microservices: A systematic grey literature review,” Journal of Systems and Software, vol. 208, 2024, p.111906.

[18] R. Oberhauser, "VR-Git: Git Repository Visualization and Immersion in Virtual Reality," 17th International Conference on Software Engineering Advances (ICSEA 2022), IARIA, 2022, pp. 9-14.

[19] R. Oberhauser, “VR-GitCity: Immersively Visualizing Git Repository Evolution Using a City Metaphor in Virtual Reality,” International Journal on Advances in Software, 16 (3 & 4), 2023, pp. 141-150.

[20] H. Bjørklund, “Visualisation of Git in Virtual Reality,” Master’s thesis, NTNU, 2017.

[21] GitHub Skyline [Online]. Available from: https://skyline.github.com 2025.05.10

[22] L. Zhang, A. Agrawal, S. Oney, and A. Guo, “VRGit: A Version Control System for Collaborative Content Creation in Virtual Reality,” In: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23), ACM, Article 36, 2023, pp. 1–14. https://doi.org/10.1145/3544548.3581136

[23] K. Højelse, T. Kilbak, J. Røssum, E. Jäpelt, L. Merino, and M. Lungu, "Git-Truck: Hierarchy-Oriented Visualization of Git Repository Evolution," 2022 Working Conference on Software Visualization (VISSOFT), Limassol, Cyprus, 2022, pp. 131-140, doi: 10.1109/VISSOFT55257.2022.00021.

[24] A. Hoff, T. H. Kilbak, L. Merino, and M. Lungu, "GitTruck@Duck - Interactive Time Range Selection in Hierarchy-Oriented Polymetric Visualization of Git Repository

Evolution," 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2024, pp. 853-857, doi: 10.1109/ICSME58944.2024.00090.

[25] https://gource.io, last accessed 2025.05.10

[26] CodeFlower [Online]. Available from: https://github.com/fzaninotto/CodeFlower 2025.05.10

[27] Y. Kim et al., “Githru: Visual Analytics for Understanding Software Development History Through Git Metadata Analysis,” IEEE Transactions on Visualization and Computer Graphics, vol. 27, 2021.

[28] C. V. Alexandru, S. Proksch, P. Behnamghader, and H. C. Gall, “Evo-Clocks: Software Evolution at a Glance,” in 2019 Working Conference on Software Visualization (VISSOFT). IEEE, 2019.

[29] J. Feiner and K. Andrews, “Repovis: Visual overviews and full-text search in software repositories,” In: 2018 IEEE Working Conference on Software Visualization (VISSOFT), IEEE, 2018, pp. 1-11.

[30] Y. Kim et al., “Githru: Visual analytics for understanding software development history through git metadata analysis,” IEEE Transactions on Visualization and Computer Graphics, 27(2), IEEE, 2020, pp.656-666.

[31] S. Elsen, “VisGi: Visualizing git branches,” In IEEE Working Conf. on Software Visualization, IEEE, 2013, pp. 1-4.

[32] A. Ciani, R. Minelli, A. Mocci, and M. Lanza, “UrbanIt: Visualizing repositories everywhere,” In 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2015, pp. 324-326.

[33] B. Hensen and R. Klamma, “VIAProMa: An Agile Project Management Framework for Mixed Reality,” In: Augmented Reality, Virtual Reality, and Computer Graphics (AVR 2021), LNCS, vol 12980, Springer, Cham, 2021, pp. 254-272.

[34] A. Colantoni, L. Berardinelli, and M. Wimmer, “DevopsML: Towards modeling devops processes and platforms,” In: 23rd ACM/IEEE International Conference Model Driven Engineering Languages and Systems: Companion Proc., ACM, 2020, pp. 1-10.

[35] I. Koren, “DevOpsUse: A Community-Oriented Methodology for Societal Software Engineering,” In: Ernst Denert Award for Software Engineering 2020, Springer, 2022, pp. 143-165.

[36] I. J. Akpan and M. Shanker, "The confirmed realities and myths about the benefits and costs of 3D visualization and virtual reality in discrete event modeling and simulation: A descriptive meta-analysis of evidence from research and practice," Computers & Industrial Engineering, vol. 112, pp. 197-211, 2017.

[37] S. Narasimha, E. Dixon, J. W. Bertrand, and K. C. Madathil, "An empirical study to investigate the efficacy of collaborative immersive virtual reality systems for designing information architecture of software systems," Applied Ergonomics, vol. 80, pp. 175-186, 2019.

[38] A. Fonnet and Y. Prie, "Survey of immersive analytics," IEEE Transactions on Visualization and Computer Graphics, vol. 27, no. 3, pp. 2101-2122, 2019.

[39] R. Müller, P. Kovacs, J. Schilbach, and D. Zeckzer, "How to master challenges in experimental evaluation of 2D versus 3D software visualizations," In: 2014 IEEE VIS International Workshop on 3Dvis (3Dvis), IEEE, 2014, pp. 33-36.

[40] R. Oberhauser, C. Pogolski, and A. Matic, "VR-BPMN: Visualizing BPMN models in Virtual Reality," In: Shishkov, B. (ed.) Business Modeling and Software Design (BMSD 2018), LNBIP, vol. 319, Springer, 2018, pp. 83–97, https://doi.org/10.1007/978-3-319-94214-8_6.

[41] R. Oberhauser, "VR-SDLC: A Context-Enhanced Life Cycle Visualization of Software-or-Systems Development in Virtual Reality," In: Business Modeling and Software Design (BMSD 2024), LNBIP, vol 523, Springer, Cham, 2024, pp. 112-129, https://doi.org/10.1007/978-3-031-64073-5_8.

[42] R. Oberhauser, "VR-SBOM: Visualization of Software Bill of Materials and Software Supply Chains in Virtual Reality," In: Business Modeling and Software Design (BMSD 2025), LNBIP, Springer, Cham, 2025.

[43] R. Oberhauser, “VR-ISA: Immersively Visualizing Informed Software Architectures Using Viewpoints Based on Virtual Reality," In: International Journal on Advances in Software, Vol. 17, No. 3 & 4, pp. 282-300, IARIA, ISSN: 1942-2628.

[44] R. Oberhauser, “VR-V&V: Immersive Verification and Validation Support for Traceability Exemplified with ReqIF, ArchiMate, and Test Coverage,” International Journal on Advances in Systems and Measurements, 16 (3 & 4), 2023, pp. 103-115.

[45] R. Oberhauser, “VR-TestCoverage: Test Coverage Visualization and Immersion in Virtual Reality,” In: Proc. Fourteenth International Conference on Advances in System Testing and Validation Lifecycle (VALID 2022), IARIA, 2022, pp. 1-6.

[46] R. Oberhauser, “VR-UML: The unified modeling language in virtual reality – an immersive modeling experience,” International Symposium on Business Modeling and Software Design (BMSD 2021), Springer, Cham, 2021, pp. 40-58, doi.org/10.1007/978-3-030-79976-2_3

[47] R. Oberhauser, “VR-SysML: SysML Model Visualization and Immersion in Virtual Reality,” International Conference of Modern Systems Engineering Solutions (MODERN SYSTEMS 2022), IARIA, 2022, pp. 59-64.

[48] R. Oberhauser and C. Pogolski, "VR-EA: Virtual Reality Visualization of Enterprise Architecture Models with ArchiMate and BPMN," In: Business Modeling and Software Design (BMSD 2019), LNBIP, vol. 356, Springer, Cham, 2019, pp. 170-187, https://doi.org/10.1007/978-3-030-24854-3_11.

[49] R. Oberhauser, P. Sousa, and F. Michel, "VR-EAT: Visualization of Enterprise Architecture Tool Diagrams in Virtual Reality," In: Business Modeling and Software Design (BMSD 2020), LNBIP, vol 391, Springer, 2020, pp. 221-239. https://doi.org/10.1007/978-3-030-52306-0_14.

[50] R. Oberhauser, M. Baehre, and P. Sousa, "VR-EA+TCK: Visualizing Enterprise Architecture, Content, and Knowledge in Virtual Reality," In: Business Modeling and Software Design (BMSD 2022), LNBIP, vol 453, Springer, 2022, pp. 122-140. https://doi.org/10.1007/978-3-031-11510-3_8.

[51] R. Oberhauser, M. Baehre, and P. Sousa, "VR-EvoEA+BP: Using Virtual Reality to Visualize Enterprise Context Dynamics Related to Enterprise Evolution and Business Processes," In: Business Modeling and Software Design (BMSD 2023), LNBIP, vol 483, Springer, 2023, pp. 110-128, https://doi.org/10.1007/978-3-031-36757-1_7.

[52] R. Oberhauser, "VR-ProcessMine: Immersive Process Mining Visualization and Analysis in Virtual Reality," Fourteenth International Conf. on Information, Process, and Knowledge Management (eKNOW 2022), IARIA, 2022, pp. 29-36.

[53] https://www.jenkins.io, last accessed 2025.05.10

[54] https://semaphore.io, last accessed 2025.05.10

[55] https://www.drone.io, last accessed 2025.05.10

[56] A.R. Hevner, S.T. March, J. Park, and S. Ram, “Design science in information systems research,” MIS Quarterly, 28(1), 2004, pp. 75-105.

[57] B. Wilson. *Jenkins Pipeline Tutorial For Beginners*. [Online]. Available from: https://devopscube.com/jenkins-pipeline-as-code/ 2025.05.10

[58] GitHub. *Semaphore demo CI/CD pipeline for Android*. [Online]. Available from: https://github.com/Semaphore-demos/semaphore-demo-android/ 2025.05.

Scalable Software Distribution for HPC-Systems with Software Pools Using MPI and File Systems in User Space

Jakob Dieterle
GWDG
Göttingen, Germany
e-mail: `jakob.dieterle@gwdg.de`

Hendrik Nolte
GWDG
Göttingen, Germany
e-mail: `hendrik.nolte@gwdg.de`

Julian Kunkel
Department of Computer Science
Georg-August-Universität Göttingen
Göttingen, Germany
e-mail: `julian.kunkel@gwdg.de`

Abstract—**Despite the increasing computing power of high-performance computing (HPC) systems, complex tasks on large-scale clusters can still be hindered by significant waiting times when loading large software packages and dependencies. These delays are often caused by network bandwidth bottlenecks, which can severely impact application performance. To address this challenge, this paper presents a new way of distributing software in HPC systems. Our software pools can hold whole software stacks in a single file, while our implemented tools can distribute software pools efficiently to large clusters while reducing bandwidth usage to a minimum. Software pools offer additional advantages, such as portability, reproducibility, and security, while seamlessly integrating into existing environments using Lmod.**

Keywords-file distribution; optimized reading; containerization.

I. INTRODUCTION

This paper extends our previous work on scalable software distribution for high performance computing systems with MPI-based file systems in user space, which introduced a design for a file system with the main goal to reduce bandwidth usage when reading large files [1]. While the original paper showed promising results, the presented file system had some limitations that were caused by the general design approach to tackle the given problem. Thus, this paper presents the new concept of software pools, instead of iterating on the work from the previous paper. Software pools are designed to improve performance while distributing software in high performance computing (HPC) environments, by reducing complex software stacks into compact image files. This allows the usage of collective communication, instead of one-sided communication used in the previous paper, potentially improving the performance and reducing complexity. This paper also provides new benchmarking results, which are compared to the results of the previous paper.

The challenge of distributing large files in HPC environments is becoming more important, as the increase in demand for computing power in data centers is unbroken. This is especially true, with the recent surge of artificial intelligence and the popularization of large language models. Because of the slowdown of Moore's law, HPC systems have to keep up with the demand by increasing the total number of cores and nodes inside the systems [2]. However, the increasing level of parallelization also leads to a greater demand for networking bandwidth inside HPC systems. Distributing data, container files, or software packages to hundreds or thousands of nodes for a single job can lead to long waiting times before any processing can even begin [3], [4].

We believe that the distribution of large files to many nodes could be organized more efficiently by using tools such as the Message Passing Interface (MPI) and Filesystem in Userspace (FUSE) [5]. In our previous paper [1], we presented a design for a file system in user space that uses MPI to distribute data between nodes on demand while the nodes access the file. This version of our file system used MPI's `MPI_Get` method to directly access memory on other nodes, and was thus called the One-Sided-Reading (OSR) file system. While this approach reduced bandwidth usage to a minimum and showed promising scaling in our performance test, it had some technical limitations, and the performance was not competitive against optimized file systems. The biggest limitation of the previous design was the on-demand communication approach. The overhead introduced by FUSE and MPI were the biggest performance factors, and both are related to the number of total read calls to the file system. The amount of individual read operations on a file is only influenced by the size of the blocks with which the application is accessing the file. We cannot control the block size the application is working with, so the room for improvement is relatively limited. The implementation of the file system also lacked some basic functionality that would be needed for real-world applications. Including those features would have increased complexity and possibly reduced performance as well.

In this paper, we want to investigate a different approach to the problem. Instead of distributing data on demand, files will be efficiently distributed before the user application accesses them. We will present our concept of software pools, which are image files containing whole directory trees which can be easily distributed and mounted anywhere. Multiple tools are implemented to build, distribute, and mount software pools. Those tools are benchmarked to evaluate their performance and viability for real-world applications and to compare this approach to the design introduced in our previous paper.

The main contributions of this work are:

- Presenting the concept of software pools
- Implementing tools that enable the utilization of software pools in HPC environments
- Benchmarking these tools and comparing them to previous designs

```
def my_open(path):
    file_handler = open(path)

    file_buffer = MPI_Win_allocate
        (file_handler.size, char)
    meta_buffer = int[file_handler.size]

    meta_buffer = calculate_distribution
        (file_handler.size, world_size)

    offset, size = calculate_my_range
        (file_handler.size, my_rank)
    data = read(offset, size)
    file_buffer[offset:offet+size] = data

    return file_handler
```

Figure 1. Pseudo code for open method.

```
def my_read(file_handler, offset, size):

    if (in_buffer(offset, size)):
        return file_buffer[offset:offset+size]

    targets = get_targets(offset, size)
    for target in targets:
        t_offset, t_size = caclulate_target_range
            (meta_buffer, offset, size)
        data = MPI_Get(t_offset, t_size, target)
        file_buffer[t_offset:t_offset+t_size] = data

    meta_buffer[offset:offset+size] = my_rank

    return file_buffer[offset:offset+size]
```

Figure 2. Pseudo code for read method.

The remainder of the paper is organized as follows: Section II presents related work and used technology. Section III presents the concept of software pools and related tools. Section IV outlines the methods used for for benchmarking the implemented tools. The results of these benchmarks are presented in Section V. In Section VI, we discuss the presented results. Finally, Section VII contains the conclusion, and we discuss future work.

II. Background

In our previous paper, we presented a file system in user space, which uses MPI to reduce bandwidth usage when accessing large files in the context of HPC systems [1]. This design used MPI's one-sided communication methods to transfer data between nodes on demand. Upon opening a file, it is split up between the nodes, and each node loads its associated part of the file into memory, which is available to be used with one-sided communication (see Figure 1). When accessing a certain part of the file, the file system checks whether it is already in the node's memory. If that is not the case, the node will read the missing range of bytes from the node that initially loaded that part of the file using the `MPI_Get` method (see Figure 2).

In our performance tests, the file system was able to match or even outperform a slower existing file system providing the

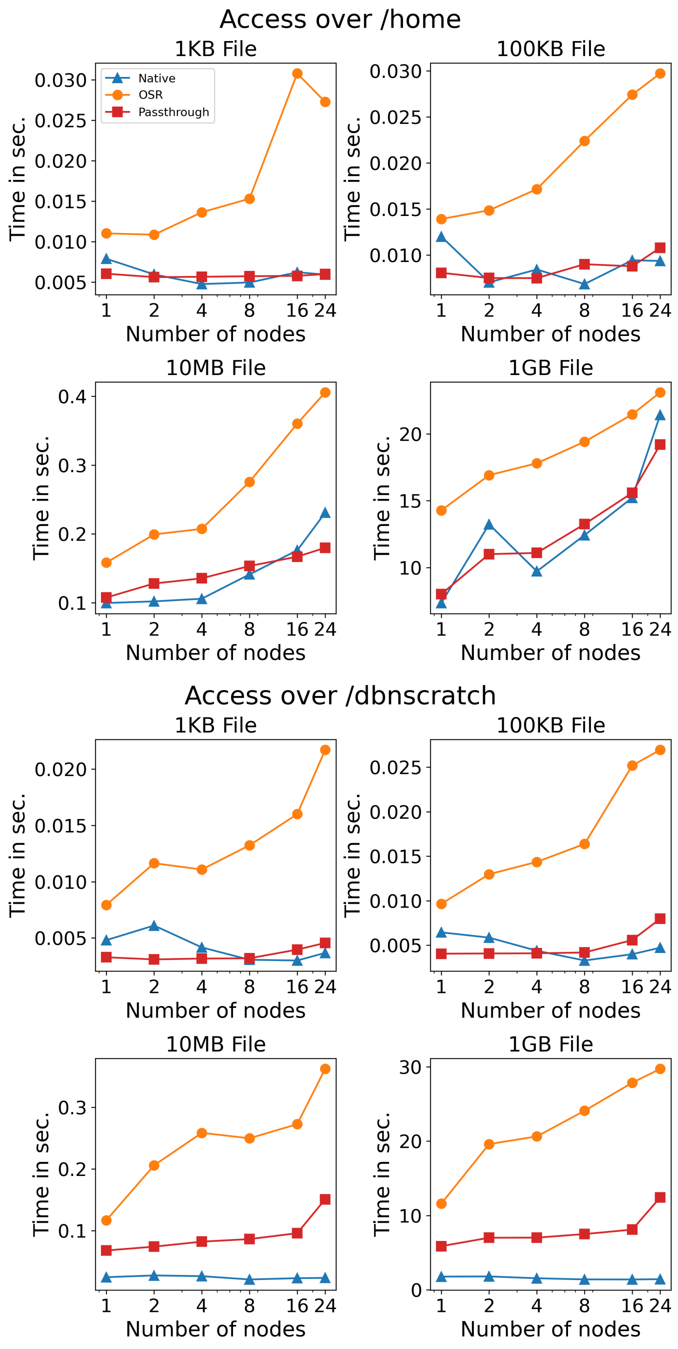


Figure 3. Time measurements on Sofja system.

home directory with about 1.4 Gb/s bandwidth. However, it was not able to reach the performance level of a more optimized scratch file system. At least not with the workloads we tested during our benchmarks (see Figure 3).

We conducted further tests with more granular timing measurements to analyze what factors were most impacting the performance of our file system. The goal was to isolate the impact MPI and FUSE have, specifically. The results show that the MPI communication calls are the biggest factor affecting the performance of the file system (see Figure 4). Since the

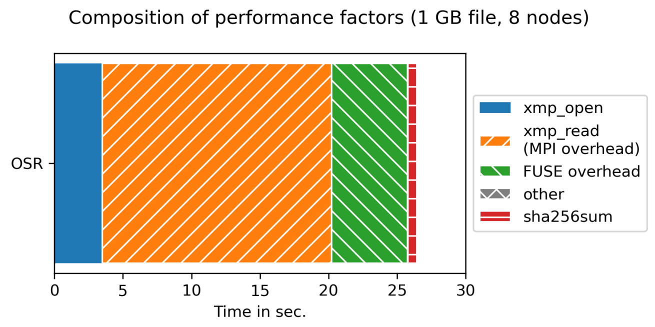


Figure 4. Visualization of performance factors for the OSR file system.

overhead by FUSE is only dependent on the file size and block size while accessing the file, it is constant for varying numbers of nodes. At the same time, the amount of overhead introduced by MPI will increase with a larger number of nodes.

Originally, file systems in Linux exclusively operated on the kernel level. Thus, developing new file systems requires changing and compiling the kernel, which can be very cumbersome and is only possible for privileged users. To make developing new file systems more accessible, the FUSE (Filesystem in Userspace) kernel module was incorporated into the Linux kernel with version 2.6.14 [5]. The FUSE project consists of the kernel module and the libfuse userspace library. By linking the libfuse library to a program, a non-privileged user can mount their own file system by writing their own open/read/write, etc. methods. When the library issues a syscall, the VFS in the kernel handles it and passes it on to the FUSE kernel module. The FUSE kernel module then calls the linked user space program, the FUSE file system, which finally executes the call.

FUSE is mainly used to implement virtual file systems, which don't actually store data but provide a different view or translation of an existing file system or storage device. Other accessible resources can also be used by FUSE file systems. For example, Fuse-archive by Google [6] allows the user to mount different types of archive file types (.tar, .zip, etc) and access it like a normal directory while decompressing the data on the fly. It uses buffers to increase performance and prevent decompressing the same files multiple times. The file system implemented later for this work will be a virtual file system using FUSE. Some popular distributed file systems also use FUSE, such as *GlusterFS* [7].

The most commonly used standard for passing messages between nodes is the *Message Passing Interface* (MPI). MPI provides different concepts for communication like Point-to-Point communication, Collective communication, and One-Sided communication. Point-to-point communication involves two specific processes to send a message from one to the other. It requires both processes to call respective methods, such as `MPI_Send` and `MPI_Recv`. Developers need to ensure that these methods are actually called by both processes. Otherwise, one process may get stuck while waiting for the other involved process, possibly resulting in a deadlock. There are also non-blocking alternatives allowing asynchronous Point-to-Point communication (`MPI_Isend` and `MPI_Irecv`). Collective communication always involves a group (or groups) of processes to share data. For example, processes can send a message to all other processes of the specified group using `MPI_Bcast` or gather data from all other processes with `MPI_Gather`, while `MPI_Allgather` combines both operations into one. These examples also have non-blocking alternatives. For synchronization when using point-to-point or collective communication, the `MPI_Barrier` method is most commonly used. As the name suggests, it acts as a barrier for the specified group of ranks, at which all ranks have to wait until all ranks reach this point in the program. One-sided communication uses the concept of Remote Memory Access (RMA), which allows a process to share data with another process without interrupting it. During initialization, all processes need to specify a memory 'window' that will be accessible by other processes. The method `MPI_Win_allocate` creates the window and allocates its memory. For the actual communication, the following two methods are of main interest. `MPI_Get` reads a range of bytes from the window on a specified target, `MPI_Put` writes into the memory of a specified target, and `MPI_Accumulate` realizes a reduction operation over the same memory over multiple targets. When using any of these RMA communication methods, they need to be encapsulated by synchronization methods, mainly `MPI_Win_lock` and `MPI_Win_unlock`. These methods start and terminate a RMA communication epoch, in which accessing the specified window is possible. `MPI_Win_lock` takes a specifier as an argument with the options `MPI_LOCK_SHARED` and `MPI_LOCK_EXCLUSIVE`. When using the keyword `MPI_LOCK_EXCLUSIVE` the window is completely locked for any other operations on the window. Other `MPI_Win_lock` calls trying to access the window on the same rank have to wait until the current lock is released again by `MPI_Win_unlock`. The keyword should be used when writing operations are executed on the window. Using `MPI_LOCK_SHARED` is safe as long as only read operations are issued during the RMA epoch, and it allows multiple processes to start an epoch on the same window at the same time. In this situation, the usage of `MPI_Win_lock` is still necessary, as the method also has to initialize the communication between the two ranks since RMA is not entirely one-sided in MPI. MPI provides many more methods and concepts, the ones presented here are a selection that will be relevant for the rest of this work.

III. Related Work

FUSE is said to significantly affect the performance of file operations due to the additional context switches introduced between user space and kernel space, as described above. Rajgarhia and Gehan evaluated the performance of FUSE using the Java bindings as an example [8]. They found that for block sequential output, FUSE adds a lot of overhead when dealing with small files and a lot of metadata but becomes quite efficient with larger files. When running the PostMark benchmark, FUSE added less than 10% in comparison to native ext3. Vangoor et al. also analyzed the performance of FUSE and its kernel module design in greater detail [9]. According to Vangoor et al., FUSE can perform with only 5% performance loss in ideal circumstances, but certain workloads can result in 83% performance loss. Additionally, a 31% increase in CPU utilization was measured. This negative effect on performance will be relevant when we compare our file system's performance to the native file system later.

The performance of MPI can vary depending on the environment and implementation that is being used. Hjelm analyzed the performance of one-sided communication in OpenMPI [10]. OpenMPI has supported one-sided communication since version 1.8.0 but emulated it using point-to-point communication. With version 2.0.0, OpenMPI introduced an implementation of one-sided communication using actual RMA concepts. The paper provides an overview of OpenMPI's RMA implementation and evaluates its performance by benchmarking the `Put`, `Get`, and `MPI_Fetch_and_op` methods for latency and bandwidth. The benchmarks showed basically constant latency for `Put` and `Get` for messages of up to 2^{10} bytes and a drastic increase of latency for messages larger than 2^{15} bytes. Analog to the latency results, the bandwidth performance plateaus with message sizes larger than 2^{15} bytes.

There are also alternative APIs to MPI for message passing and I/O management. One of them is Adios2, presented by Godoy et al. [11]. Adios2 is designed as an adaptable framework for managing I/O on a wide range of scales, from laptops to supercomputers. Adios2 provides multiple APIs with its MPI-based low-level API being designed for HPC applications. It realizes both, parallel file I/O and parallel intra/interprocess data staging. Adios2 adopts the Open Systems Interconnection (OSI) standard and is designed for high modularity. Each provided component can be mapped to OSI layers 4 to 7. At the core of its concept are abstract engines, which describe I/O workflows by bundling components from OSI layers 4 to 7. Engines are highly adaptable to different use cases (mainly parallel file I/O and parallel data staging) and performance needs.

Also important to mention here is OpenMP (Open Multi-Processing). OpenMP provides libraries for multi-threaded processing using a shared memory model. In C/C++ `#pragma`'s are used to start multi-threaded processing, they are often used to parallelize loops without data dependency, thus multiple iterations of a loop can be calculated concurrently. OpenMP is limited to multi-threaded processing on a single node, so we cannot use it for the current project. However, MPI and OpenMP are often used together with MPI handling inter-node communication and OpenMP used for parallelization inside the nodes.

As High-Performance Computing (HPC) applications become increasingly complex, the size and complexity of the software packages used to support them have also grown. Research has shown that the number of package dependencies in HPC has skyrocketed in recent years [4]. This paper, written by Zakira et al., explores various software deployment models, including store models like Spack, which is used on the HPC systems hosted by the GWDG. The authors also introduce their own solution, Shrinkwarp, which aims to reduce loading times for highly dynamic applications. However, the paper's primary focus is on software distribution models and package management, rather than optimizing loading times through improved I/O efficiency.

The concept of creating file systems in user space to enhance I/O performance is not a novel idea. Several existing file systems, such as FusionFS, have already explored this approach. FusionFS, in particular, is a user-space file system that optimizes metadata operations by storing remote file metadata locally and is designed to handle high-volume file writes [12]. In contrast, in our work, we focus on optimizing file reads to facilitate the distribution of files and software.

In HPC Systems, resources like memory and CPUs are tightly coupled into nodes. Systems may offer different configurations of nodes to serve different use cases. Still, studies show, that HPC jobs often underutilise the available memory [13]. The concept of disaggregation in HPC systems aims to decouple resources like memory and processing power by allowing direct memory access over network interfaces. Peng et al. conducted a study on memory utilization, showing that 90% of all jobs utilized less than 15% of node memory and 90% of the time the node less than 35% of node memory is used [13]. These results highlight the under utilization of resources by a majority of jobs running on HPC Systems, while only few jobs profit from well equipped compute nodes. A possible approach to improve resource utilization could be the concept of disaggregated architectures, as they aim to disconnect different resources like computing power and memory capacity from each other. Thus, the paper introduces different disaggregated architectures, including a centralized design with dedicated memory nodes and a decentralized design, in which nodes share their local memory. Finally, Peng et al. present their implementation of a remote paging library *rMap*. It uses a centralized approach to allow nodes with little local memory to run memory-intensive jobs by requesting memory pages of dedicated nodes and swapping them into local memory.

Creating and distributing software environments is a common problem, especially in the HPC environment. A popular solution for this problem is containerization. A container is a lightweight and portable way to package an application, its dependencies, and a runtime environment into a single unit that can be easily deployed and managed. Containers allow the execution of software in an environment totally isolated from the host system. Similar to virtual machines, while not requiring dedicated resources and using the kernel of the host operating system. The best containerization software in the HPC environment is Apptainer. Apptainer is designed with a focus on security and reproducibility. Rajesh Pasupuleti et. al. discussed what advantages Apptainer containers offer for running AI applications on HPC systems [14].

In HPC Systems, software packages are often managed by module systems. The HPC systems hosted by the GWDG use Lmod. Lmod is a Lua-based module system that uses module files to dynamically change the user's environment [15]. This paper aims to offer a solution for distributing software that seamlessly integrates into the existing Lmod environment on the GWDG systems.

In this section we reviewed different frameworks for communication including OpenMP, Adios2 and MPI. OpenMP does not fit our given problem, as it only allows communication between processes on the same node, and not between multiple nodes. Adios2 allows for communication between nodes, but mainly provides more high level API which is unnecessarily complex for our use case. We also talked about other file systems that aim to improve file I/O performance like FusionFS. However, in contrast to our goal of improving performance when reading large files, FusionFS is focused on improving performance for writing to files. Containerization applications like Docker and Apptainer were also mentioned. They allow running software in isolated systems, similarly to virtual machines, but with less overhead by using the same kernel as the host system. We considered using container files as a base for our software pools, but decided against it, because container images often include all the files of a operating system and can be difficult to handle.

Ultimately, the goal of this paper is to create a tool set that includes features of many of the before mentioned technologies, such as file I/O performance improvement, image file handling and software distribution.

IV. Design

The proposed design of our original paper introduced a lot of overhead caused by the on-demand communication approach. Using MPI calls to transfer data while files are being accessed introduces a lot of extra latency, which is a problem for performance-critical applications. Especially in the HPC environment, where users might be charged per used core hour.

For this reason, we want to investigate whether it is more efficient to distribute the files before the application starts. Specifically, we want to use MPI to broadcast the files and store a copy locally so that the local copy can be accessed during runtime. However, this becomes quite complicated and inefficient when dealing with large directory trees and large amounts of files. For example, simple Python environments can easily contain tens of thousands of files. To simplify the task of broadcasting whole software stacks and huge directory trees, we want to pack them into a single file that is easy to distribute and mount on the target nodes. These files, which can hold complex software stacks, will be called software pools. Once the software pool is mounted on the target nodes, we want to integrate it automatically into an existing software module system. In our case, that would be Lmod.

Being able to load multiple software packages by mounting a single file has multiple other advantages. It reduces the Inode usage and metadata operations on the original file system. This is especially true when dealing with a lot of files, as is the case when using Python, Conda, or similar environments. Software pools can make it easier to manage collections of software packages on multiple levels. We plan to provide software pools on personal, project, and data center level, so that users can easily modify their environment for their current tasks. Software pools make sharing and reproducing software collections easy, as they can include binaries and source code. That allows the software pools to be shared between users and different HPC Systems. Software pools offer a simple way to add a trust factor for integrity and trustworthiness by offering a built-in method to sign and verify pool files using existing algorithms with private and public key pairs.

```
# create file pool.ext2 which contains the
# software pool
dd if=/dev/zero of=pool.ext2 bs=<block-size> \
    count=<number-blocks> conv=sparse

# create file system and copy given directory
# as root directory into the pool file
mke2fs pool.ext2 -d <source-directory>
```

Figure 5. Bash commands for building a software pool.

Choosing the correct file format for the project is crucial. The first idea was to use apptainer containers as software pools. Being able to quickly build containers from a simple definition file can be very beneficial for software pools' use cases. The directory tree of an apptainer container can be mounted by dumping the container's content using apptainer's `sif` tool. The resulting squashFS image could then be mounted using `squashfuse`. However, while testing this workflow, multiple drawbacks became obvious. Most importantly, apptainer images often take quite a long time to build and cannot be modified without completely rebuilding them. That makes the process of creating a software pool unnecessarily cumbersome and time-consuming.

Instead, we decided to focus on simpler file types and we used image files as containers for ext2, ext3, or ext4 file systems. To handle all operations in the context of the files representing software pools, we implemented our own tool called `sw-pool`. This tool offers the following commands: build, sign, verify, and load.

The build command creates a software pool from a given source directory. It takes the output file and source directory as arguments and the file size as an option. This directory should already include everything the pool should contain, most importantly, the binaries of the applications included in the pool. First, a file with a fixed size has to be created. This file should be sparse to ensure it only occupies as much disk space as needed. Then, the file can be written to using the `mke2fs` tool. It is part of the E2fsprogs package, which includes various utilities to handle ext2 (or ext3, ext4) file systems. With `mke2fs`, we can initialize the file system and copy the content of the software pool in one step, as you can see in Figure 5. This image can later be mounted using the `load` command.

The sign and verify commands add a factor of trustworthiness and integrity to the software pools. Internally, we use OpenSSL with the `-sign` and `-verify` to sign and verify the file with a given private and public key pair.

The load command broadcasts the given software pool, writes it to local temporary storage, and mounts it to the given mount point. The broadcasting and mounting procedures are done with other standalone tools we implemented for this use case. These tools are separated from the primary software pool tool, as broadcasting and mounting files with dedicated tools can also be very helpful outside of the context of software pools.

When the load command is called, it first determines the ideal temporary directory to store the local copy of the software pool. Then, our tool `scalable-fs-cp` is called. It takes an input file or directory and the target path as arguments. Before it does anything else, it checks if the input file already exists in all of the nodes it is running on. If the file is already present, nothing has to be done, so the tool can simply exit. Only if the file doesn't exist on all nodes does it start the procedure to broadcast the file using MPI.

Before the file can be broadcasted, it has to be read into memory. This is not done by one node, instead the file is split into N sections, with N being the number of nodes in our job. This is more efficient, as we know from our previous paper. After the sections of the file are loaded into memory, it can be broadcasted. For that, we iterate over the nodes, and each node broadcasts its section to the other nodes. This is not done with a single big chunk, however. Since our software pools are sparse files, the file size can be quite large, even though only a small part of the file was actually written. The rest of the file is filled with zeros, as we used `/dev/zero` as input when creating the file. To avoid broadcasting all the empty parts of the file, each section is processed block by block, with a default block size of 4096 bytes. Each will only be broadcasted if it contains a byte that is not zero. This should significantly reduce the time it takes to broadcast large pool files that are partly empty without having a big impact on broadcasting fully written pools, as we check for the first nonzero byte for each block. For this to work, the array to store the file in memory should be initialized to zeros. This can be done easily by simply using `calloc` instead of `malloc` in C.

Once the file is broadcasted, it can simply be written to a given target location on the nodes. If the tool detects that it is only running on a single node, it skips the broadcasting procedure and simply uses the `cp` command to copy the file to the given target location. Pseudo code for this procedure is listed in Figure 6. Copying directories is also supported. This comes with a performance loss, however, since the directory has to be compressed to and extracted from an archive file before and after broadcasting.

Once the distribution of the software pool is done, the software pool tool uses our `scalable-fs-mount` tool to mount the local copy to a given mount point. This tool supports different file types, in the case of our software pool file, it uses `fuse2fs`, which is also part of the E2fsprogs package.

Now that we have successfully distributed and mounted the software pool, the included software must be made available to the user. Theoretically, the user could just add the binary locations manually to the path. While this might be feasible for small software pools, it would not work for more complex software stacks, since most HPC systems already use module systems that can solve this problem for us, especially. In our case, the HPC systems hosted by the GWDG use the Lmod module system. As described in Section II, Lmod uses module files to load modules into the user's environment. Lmod looks for module files based on the `MODULEPATH` environment variable. To make our software pool visible for Lmod, we need to provide the correct module files and add the corresponding paths to the `MODULEPATH` variable. This happens in the `gwdg-sw load` command after mounting the software pool. For this process to reliably work, the pools need to have a standardized format so that the `gwdg-sw` tool knows where to find the module files. Our proposed pool structure is presented in Figure 7.

The directory structure for the module files might seem unnecessarily complex. However, it is needed for Lmod to detect the correct module and software package hierarchy. This is complicated by the fact that Lmod requires the module files

```
def bcast_cp(input_file, output_path,
        rank, world_size):
  if file_exists:
    return 0
  if world_size > 1:
    file_buffer = zeros(file_size(input_file))
    offset, range = get_section(rank)
    file_buffer[offset:offset+range] =
            input_file[offset:offset+range]
    for curr_rank in world_size:
      blocks = split(file_buffer, curr_rank)
      if not is_empty(block):
        MPI_Bcast(block, curr_rank)
    write(output_path, file_buffer)
  else:
    copy(input_file, output_path)
```

Figure 6. Pseudo code for scalable-fs-cp.

```
<Pool-Name>
└─<Pool-Version>
   ├─modulefiles
   │  └─<Pool-Name>
   │     └─<Pool-Version>
   │        └─<SW-Name>
   │           └─<SW-Version>.modulefile
   ├─install
   │  └─<SW-Name>
   │     └─<SW-Version>
   │        └─binaries...
   ├─source
   │  └─<SW-Name>
   │     └─<SW-Version>
   │        └─source code...
   └─config
      └─...
```

Figure 7. Proposed Software Pool Structure.

```
local version = myModuleVersion()
local pkgName = myModuleName()
local mountPoint = os.getenv("GWDG_SW_MOUNT")
local poolName = "test-pool"
local poolVersion = "0.5"
local pkg = pathJoin(mountPoint,poolName,poolVersion
                    ,"install",pkgName,version,"bin")
prepend_path("PATH",pkg)
```

Figure 8. Module File Example.

to be in a different directory than the module that the module file itself points to. The module file, in this case, would add the path of the binaries of the corresponding software in the `install` directory. An example for what a module file should look like can be seen in Figure 8.

The `source` and `config` directories are optional. However, pool authors should consider making the pools reproducible. To support this, any source code would go into the `source` directory, and any other files that are required for building the pool should go into the `config` directory.

With this in place, all the software packages included in the software pool are visible for Lmod after loading the pool with `gwdg-sw load <pool-name>/<pool-version>` and can be activated with Lmod using `module load <SW-name>/<SW-version>`.

V. METHOD

It is difficult to compare the performance of the tools we introduced in this paper to the file system we implemented in our previous paper, as they follow two completely different approaches. The file system from our previous paper impacted the latency of each read operation during the execution of the user's application. The software pool tools we implemented in this paper distribute the software packages before the user application starts. Thus, the effect on read latency is reduced because we still have overhead caused by FUSE, but no overhead from MPI. The overhead from MPI is shifted to the loading procedure which happens before the start of the user application. To see if this improves the overall situation, we can compare the total time it takes to read a file with the OSR file system with the time it takes to distribute the file in a software pool and read it afterward.

The tests will be run on the Emmy system hosted by the GWDG [16]. The system consists of 1.423 nodes with 111.464 cores. The system scored 5,95 PetaFlop/s during the LINPACK benchmark.

For comparison with the OSR file system, the results from the previous paper will be used. In this paper, we will focus our benchmarks on files with a size of 1 GB since the project aims to improve performance with large software stacks, and the largest test case from the previous paper is 1 GB.

For the first part of our benchmarks, we will measure the time it takes to distribute a pool using our introduced `sw-pool` tool with 2, 4, 8, 16, and 32 nodes. Ten runs will be conducted for each number of nodes, and the average of the ten runs will be calculated to obtain a robust result. To match the benchmarks of our previous paper, we will create a software pool, that holds a file with the size of 1 GB filled with random data. The commands to create our pool for tests are listed in Figure 9. Note that the file for the software pool has to be slightly larger than the test file since it needs some space to create the ext2 file system. To ensure that caching mechanisms do not affect our testing results, the test data will be regenerated on a different node before each run.

This software pool will be saved to the scratch file system. We will measure the overall time the command for mounting the software pool takes. The exact command we will use to execute and measure the test is listed in Figure 10. Here, the `$LOCAL_TMPDIR` variable points to the local SSDs of the nodes, and the `$SCRATCH` variable points to the SSDs of the remote scratch file system. In order to isolate how much of the total time is used for reading and broadcasting the file, additional time measurements are added to the code using the `MPI_Wtime` method.

```
# create test file
head -c 1GB /dev/urandom > test-pool/random.data

# create pool inlcuding test file
sw-pool build -s 1020MB test-pool.ext2 test-pool
```

Figure 9. Procedure to create software pool for testing.

```
time mpirun sw-pool -v load -m $LOCAL_TMPDIR/mnt \
                $SCRATCH/random.ext2
```

Figure 10. Command to time and execute test for broadcasting and mounting a software pool.

Secondly, we will measure the time it takes to calculate a hash sum of the test file inside the container after the software pool is copied to local SSDs and mounted using our tools. We only have to run this test on one node since this operation would always happen independently of all the nodes inside a bigger job, as the software is copied to local SSDs on the nodes. The results of this test can then be added to the results of the previous test for any number of nodes. Again, there will be 10 runs, to produce a dependable result. The same test will be conducted, but without accessing the test file through a fuse mount. By comparing these two results, we can isolate the effect that the fuse mount has on our setup's overall performance.

VI. RESULTS

The results of our first benchmark can be seen in Figure 11. In this figure, we compare the performance of the `sw-pool` tool, including the time to calculate a hash sum of the test file over the mount point, to the results from our previous paper. We can see a clear improvement when comparing the new results to the performance of the first implementation of the OSR file system. The software pool tools are almost twice as fast overall: 10.224 seconds against 19.606 seconds with two nodes and 16.159 seconds against 27.866 seconds with

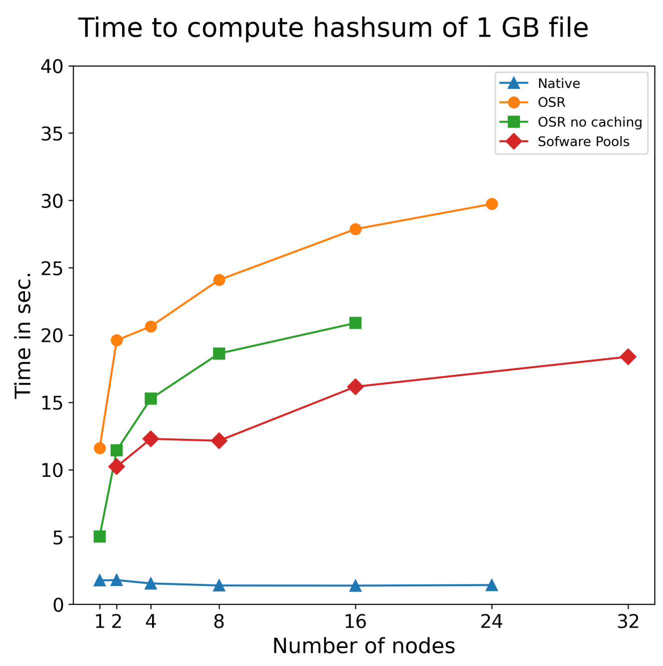


Figure 11. Timings of Software Pool Tool, OSR, OSR without caching and native file system.

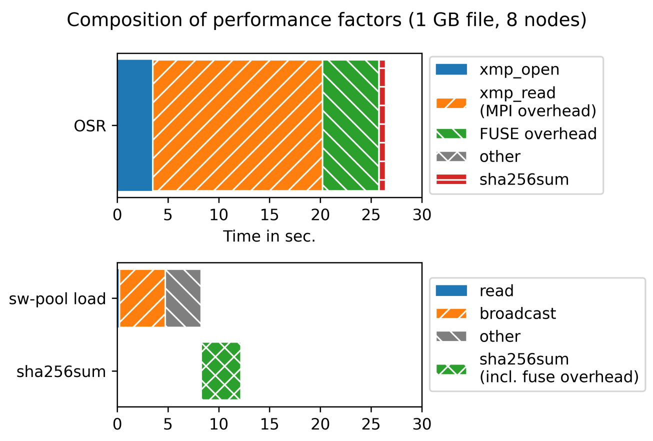


Figure 12. Composition of performance factors for the OSR file system and Software Pools.

16 nodes. With more than 2 nodes, the software pools are almost 25 % faster than the improved OSR implementation (16.159 seconds against 20.889 seconds with 16 nodes). The native scratch file system is still much faster than all the other options.

The results from the performance factor analysis can be seen in Figure 12. Here we compare the results when using software pools against the first implementation of the OSR file system. The biggest difference is clearly the time needed for the communication procedures. We were able to reduce that from 16.711 seconds to 4.512 seconds for handling a 1 GB sized file. The overhead caused by using a fuse also seems slightly improved (3.921 seconds including hash sum calculation against 5.557 seconds excluding hash sum calculation).

VII. Discussion

The software pools and corresponding tools that we presented in this paper showed much-improved performance compared to the OSR file system we presented in our previous paper. This was achieved while still reducing the load put on the storage nodes and network infrastructure to a minimum. The performance increase was achieved by shifting the critical communication procedure to before the user application starts. That allowed us to use MPI's broadcasting method instead of one-sided communication. This change also results in much better latency during runtime, which can lead to an even bigger performance advantage in real-world applications. Especially since with the new approach, existing caching mechanisms in `fuse2fs` and the underlying native file system on the node can have a positive effect, while the OSR file system did not have a working caching mechanism. Additionally, the concept of software pools and the implemented tools are much more mature than the existing implementation of the OSR file system. This was possible since we were able to use existing tools, such as `make2fs` and `fuse2fs` for handling and mounting our software pool image files and `openssl` for signing and verifying our software pools. With our integration into the existing Lmod environment, the software pool tools are almost ready to go into production, while the OSR implementation is missing basic features, such as a working caching mechanism, handling multiple files at once, and multi-threading.

VIII. Conclusion and Future Work

In this paper, we presented a new way to distribute software in HPC environments with our software pools. Software pools offer a simple way to create software environments on HPC systems. By being able to represent whole software stacks in a single file, software pools offer multiple additional benefits. Software pools can be shared easily between users and HPC systems, are reproducible, and reduce the load on stage nodes and network infrastructure. By using a single file, software pools also reduce the usage of Inodes and offer an additional trust factor by being able to sign and verify them with private/public key pairs. The corresponding tools we implemented showed greatly improved performance when compared to the OSR file system from our previous paper. This was achieved by shifting the critical communication procedures to before the user application runs. We were able to seamlessly integrate software pools into the existing module management software Lmod. The tools we presented are already very mature and can go into production without much additional work.

In the future, we want to put software pools into production on the HPC systems hosted by GWDG. To that end, we need to work on documentation, user support, and user training. The implemented tools should also be further improved and expanded. For example, we want to explore other file types, such as squashFS images, that could be used as software pools.

References

[1] J. Dieterle, H. Nolte, and J. Kunkel, "Scalable software distribution for HPC-systems using MPI-based file systems in user space", in *SCALABILITY 2024 : The First International Conference on Systems Scalability and Expandability*, Valencia, Spain, Nov. 2024, pp. 14–20, ISBN: 978-1-68558-216-6.

[2] C. E. Leiserson *et al.*, "There's plenty of room at the top: What will drive computer performance after moore's law?", *Science*, vol. 368, no. 6495, eaam9744, 2020. DOI: 10.1126/science.aam9744.

[3] W. Frings *et al.*, "Massively parallel loading", in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ser. ICS '13, Eugene, Oregon, USA: Association for Computing Machinery, 2013, pp. 389–398, ISBN: 9781450321303. DOI: 10.1145/2464996.2465020.

[4] F. Zakaria, T. R. W. Scogland, T. Gamblin, and C. Maltzahn, "Mapping out the HPC dependency chaos", in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–12. DOI: 10.1109/SC41404.2022.00039.

[5] T. kernel development community, "The Linux kernel documentation - FUSE", [Online]. Available: https://www.kernel.org/doc/html/next/filesystems/fuse.html?highlight=fuse (visited on 10/24/2024).

[6] Google, "Fuse-archive repository", [Online]. Available: https://github.com/google/fuse-archive (visited on 10/24/2024).

[7] R. Hat, "Gluster", [Online]. Available: https://github.com/gluster/glusterfs (visited on 05/26/2025).

[8] A. Rajgarhia and A. Gehani, "Performance and extension of user space file systems", in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC '10, Sierre, Switzerland: Association for Computing Machinery, 2010, pp. 206–213, ISBN: 9781605586397. DOI: 10.1145/1774088.1774130.

[9] B. K. R. Vangoor, V. Tarasov, and E. Zadok, "To FUSE or not to FUSE: Performance of User-Space file systems", in *15th USENIX Conference on File and Storage Technologies (FAST 17)*, Santa Clara, CA: USENIX Association, Feb. 2017, pp. 59–72, ISBN: 978-1-931971-36-2.

[10] N. Hjelm, "An evaluation of the one-sided performance in Open MPI", in *Proceedings of the 23rd European MPI Users' Group Meeting*, ser. EuroMPI '16, Edinburgh, United Kingdom: Association for Computing Machinery, 2016, pp. 184–187, ISBN: 9781450342346. DOI: 10.1145/2966884.2966890.

[11] W. F. Godoy *et al.*, "ADIOS 2: The adaptable input output system. a framework for high-performance data management", *SoftwareX*, vol. 12, p. 100 561, 2020, ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2020.100561.

[12] D. Zhao *et al.*, "FusionFS: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems", in *2014 IEEE International Conference on Big Data (Big Data)*, 2014, pp. 61–70. DOI: 10.1109/BigData.2014.7004214.

[13] I. Peng, R. Pearce, and M. Gokhale, "On the memory underutilization: Exploring disaggregated memory on HPC systems", in *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2020, pp. 183–190. DOI: 10.1109/SBAC-PAD49847.2020.00034.

[14] R. Pasupuleti, R. Vadapalli, C. Mader, and V. J. Milenkovic, "Apptainer-based containers for legacy and platform dependent machine learning applications on HPC systems", in *2024 IEEE International Conference on Big Data (BigData)*, 2024, pp. 6083–6091. DOI: 10.1109/BigData62323.2024.10825376.

[15] R. McLay, "Lmod: A new environment module system", [Online]. Available: https://lmod.readthedocs.io/en/latest/ (visited on 03/14/2025).

[16] G. für wissenschaftliche Datenverarbeitung mbH Göttingen, "NHR-NORD@Göttingen systeme "Emmy"", [Online]. Available: https://gwdg.de/hpc/systems/emmy/ (visited on 03/14/2025).