


VMTools-RA: a Reference Architecture for Software Variability Tools


Ana P. Allian

(State University of Maringá, Maringá, Paraná, Brazil)

 <https://orcid.org/0000-0001-9399-0944>, ana.allian@gmail.com)


Leandro F. Silva

(State University of Maringá, Maringá, Paraná, Brazil)

 <https://orcid.org/0000-0001-8860-5968>, leandroflores7@gmail.com)


Edson Oliveira Jr

(State University of Maringá, Maringá, Paraná, Brazil)

 <https://orcid.org/0000-0002-4760-1626>, edson@din.uem.br)

Elisa Y. Nakagawa

(University of São Paulo, São Carlos, São Paulo, Brazil)

 <https://orcid.org/0000-0002-7754-4298>, elisa@icmc.usp.br)

Abstract: Currently, software systems must be appropriately developed to support an amount of variability for accommodating different requirements. To support such development, a diversity of tools has already been designed for variability management (i.e., identification, modeling, evaluation, and realization). However, due to this diversity, there is a lack of consensus on what in fact software variability tools are and even what functionalities they should provide. Besides that, the building of new tools is still an effort- and time-consuming task. To support their building, we present VMTools-RA, a reference architecture that encompasses knowledge and practice for developing and evolving variability tools. Designed in a systematic way, VMTools-RA was evaluated throughout: a controlled experiment with software developer practitioners; and an instantiation of the VMTools-RA architecture to implement a software variability tool, named SMartyModeling. As a result, VMTools-RA is evidenced to be feasible and it can be considered an important contribution to the software variability and developers of variability-intensive software systems community, which require specific tools developed in a faster manner with less risk, what a reference architecture could provide.

Keywords: software variability, reference architecture, SMartyModeling, VMTools-RA, tool, variability management

Categories: D.2, D.2.10, D.2.11, D.2.13

DOI: 10.3897/jucs.97113

1 Introduction

Variability is perceived as the ability of a system to be efficiently extended, changed, or adapted for a specific context in a preplanned manner [Galster et al., 2014]. It involves all system life cycles throughout the identification of variation points, variants, and constraints among such variants [Bosch et al., 2015]. Whereas variants represent a

design option (i.e., mandatory, optional, alternative), variation points refer to areas affected by several options where system feature constraints are stated. The management of variability to the development of such systems is not therefore a trivial activity and, hence, it demands automated tools for identifying, representing, configuring, and deriving software products [Capilla et al., 2013]. Several software variability tools have been developed by both industry and academia [Bashroush et al., 2017] and certain capabilities are still missing [Allian et al., 2020]. We are able to verify this as outstanding software engineering conferences have dedicated tracks for demos and tools as, for instance, the Software and Systems Product Line Conference (SPLC) and the International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS), in which several works discuss on the need for software variability tools.

In another context, reference architectures refer to a special type of software architecture that captures the essence of architectures of a collection of systems in a given domain [Martínez-Fernández, 2013]. Their main purpose is to guide the development, standardization, and evolution of systems [Angelov et al., 2013, Nakagawa et al., 2014]. Reference architectures can embody lessons learned, best practices, architectural principles, and design patterns, therefore, they can “operationalize” knowledge and good practices. The main benefits of these architectures are the shared understanding of the current architecture across multiple organizations, reuse of software elements combined with best practices, increased productivity, reduced time-to-market, reduced development costs, enhanced quality, facilitation in the interoperability, improved communication, and the elaboration of missions [Martínez-Fernández, 2013]. Examples of successful reference architectures from the industry are AUTOSAR (AUTomotive Open System ARchitecture)¹ and European Interoperability Reference Architecture (EIRA) for interoperability services in Europe [EIRA, 2018].

With regard to software variability tools, there is not still a reference architecture that can support their development. Besides that, existing tools miss important features/functionalities including interoperability with other development tools, collaborative features, and derivation facilities to implement variability in code [Bashroush et al., 2017]. There is also a lack of consensus on the main functionalities these tools should provide. In addition, the development of new tools is still an expensive, time/effort consuming, and error-prone task [Allian et al., 2020].

Motivated by this scenario, this paper presents a reference architecture for software variability tools, named VMTools-RA, which aims at reducing the time and effort spent on the development of software variability tools. The conception of VMTools-RA was based on ProSA-RA, a systematic process for the design, representation, and evaluation of reference architectures [Nakagawa et al., 2014]. The evaluation of VMTools-RA involved two studies: a controlled experiment with software developer practitioners and an instantiation of VMTools-RA and its implementation for a new software variability tool, named SMartyModeling. As the main result, we provide preliminary evidence VMTools-RA can be considered an important contribution for software developers, who require specific tools for handling software variability in a more efficient way.

The remainder of the paper is organized as follows: Section 2 addresses the background of software variability and reference architectures; Section 3 presents our adopted methodology for this research; Section 4 presents the steps of ProSA-RA for the designing of VMTools-RA; Section 5 reports the experiment conducted to evaluate VMTools-RA with practitioners; Section 6 presents the implementation of a variability tool as an instance of VMtools-RA; Section 7 provides discussion on the results of this paper; and

¹ <http://www.autosar.org/>

Section 8 presents a summary of contributions and main perspectives for future work.

2 Background and Related Work

This section summarizes concepts related to software variability and reference architectures.

2.1 Software Variability

Most software systems must support a large amount of variability, which enables the customization and reuse of software-intensive systems for specific domains [Capilla et al., 2013, Bosch et al., 2015]. Over the past decades, the Software Product Line (SPL) engineering has consolidated variability management as one of its essential activities for successful non-opportunistic reuse [Capilla et al., 2013]. Several approaches have been developed for handling variability in software systems [Chen et al., 2009, Bashroush et al., 2017, Galster et al., 2014, Raatikainen et al., 2019], whereas most studies have reported experience in SPL with feature model and its extensions.

Software variability represents the product features in terms of variants and variation points [Capilla et al., 2013]. Features can be selected and configured at different development stages and variability management facilitates the realization and configuration of variants for different products. Figure 1 depicts an example of a feature diagram for a Mobile Phone with variabilities. A **MobilePhone** must provide **Calls**, as it is the essence of a mobile phone. It also must have a **Screen**, either a **Basic** one or **HD** (High Definition). Such a mobile phone might have a **GPS** and might play **Media**, in this case, **Camera** or **MP3**. However, in case a mobile phone has a **Camera** it must have an **HD** screen specified by the constraint “ $Camera \Rightarrow HD$ ”.

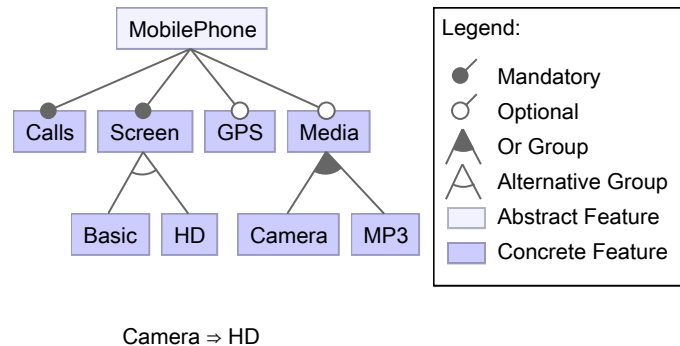


Figure 1: Mobile Phone feature diagram

Variability management encompasses a number of activities [Pohl et al., 2005], such as identification and modeling of system variants, implementation (realization), and selection and configuration (product derivation).

The International Organization for Standardization (ISO) proposed ISO/IEC 26550 [ISO/IEC26550, 2015], which encompasses a set of activities for the development and

maintenance of variants in all software development phases. It also provides a reference model containing abstract representations of the main variability management processes and details the domain engineering that defines and implements domain assets and application engineering [Pohl et al., 2005, ISO/IEC26550, 2015]. Although both standards, ISO 26550 and ISO 26555, address important information about the variability management process, they do not provide activities or technical support for the development of software variability tools. In addition, several capabilities are still missing in the existing tools [Allian et al., 2020].

2.2 Reference Architectures

Reference architecture is considered a predefined standard designed for a specific business context [Nakagawa et al., 2014, Angelov et al., 2013, Martínez-Fernández, 2013]. It facilitates the design of concrete architectures for new systems and new versions or extensions of similar products. In summary, it covers three concepts, namely *Technical information*, which provides technological solutions to instance design patterns, *Business models*, which guides decisions based on domain business rules, and *Customer context*, which recognizes customer and user considerations.

In general, reference architectures have been built using an *ad-hoc* approach, i.e., without following a systematic process [Nakagawa et al., 2014]. However, systematizing their building allows one to achieve more effective reference architectures, i.e., architectures that could better achieve their purpose. In this perspective, it is possible to find several initiatives [Angelov et al., 2013, Bayer et al., 2004, Cloutier et al., 2010, Dobrica and Niemela, 2008, Galster et al., 2014, Muller and Laar, 2008, Nakagawa et al., 2014] to provide guidelines, principles, and recommendations to build reference architectures.

[Nakagawa et al., 2014] proposed ProSA-RA, a process that systematizes reference architecture design, representation, and evaluation in four steps (Figure 2). In Step 1, information sources are selected and investigated, whereas requirements are identified and common functionalities and configurations are described in Step 2. The description and views are established in Step 3 and an evaluation is conducted in Step 4. In this work, we adopted ProSA-RA, which supports the organization of domain knowledge, systematizing the architecture establishment (i.e., requirement identification, creation, representation, and evaluation of the architecture).

ProSA-RA focuses on how to design, represent, and evaluate such architectures. It is the result of the experience in the establishment of architectures for several domains, such as robotics systems [Feitosa, 2013], digital television applications [Duarte, 2013], software testing [Nakagawa et al., 2007, Oliveira and Nakagawa, 2011], software engineering environments [Nakagawa et al., 2011], and medical systems [Rodriguez et al., 2015]. In this perspective, and considering the experience acquired with the ProSA-RA application to develop reference architectures, the ProSA-RA process is detailed in the remainder section of this work.

2.3 Related Work

To the best of our knowledge, there is no study aiming at specifying a reference architecture for software variability tools.

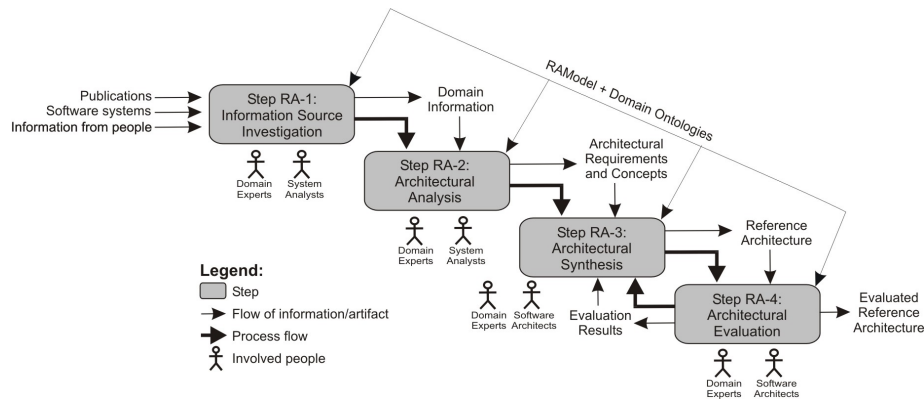


Figure 2: ProSA-RA steps [Nakagawa et al., 2014]

3 Research Methodology

In this work, we adopted a multi-method research methodology to combine techniques to complement each other. This may take a number of forms to help to confirm research findings. Thus, collecting different types of data by means of different methods should result in wider coverage of the problem space [Wood et al., 1999]. This type of research methodology combines quantitative and qualitative approaches, thus it balances the limitations of each method and provides stronger evidence and more granular results than each individual method.

Therefore, our methodology encompasses three phases (Figure 3): **Phase 1:** VMTools-RA conception using ProSA-RA; **Phase 2:** VMTools-RA feasibility controlled experiment with practitioners (Ev.2); and **Phase 3:** VMTools-RA instantiation and implementation of SMartyModeling, a UML-based SPL tool (Ev.3).

4 Phase 1: Conception of VMTools-RA

The main focus of VMTools-RA is to provide a general framework to support the development of new software variability tools. Among its main objectives are: to support the development of software variability tools; support and improve the systematic reusability of software; support the maintenance and evolution of software variability tools based on VMTools-RA; specify a standardized structure based on the processes and reference models provided by ISO/IEC 26555; and provide integration mechanisms with domain analysis and requirements specification tools.

We followed the four steps of ProSA-RA [Nakagawa et al., 2014] during the conception of VMTools-RA, according to Figure 4. We then present the first three steps (RA-1, RA-2, and RA-3) in Sections 4.1, 4.2, and 4.3, respectively. We performed step RA-4 with an empirical study (Section 5) and the instantiation and implementation of a software variability tool (Section 6).

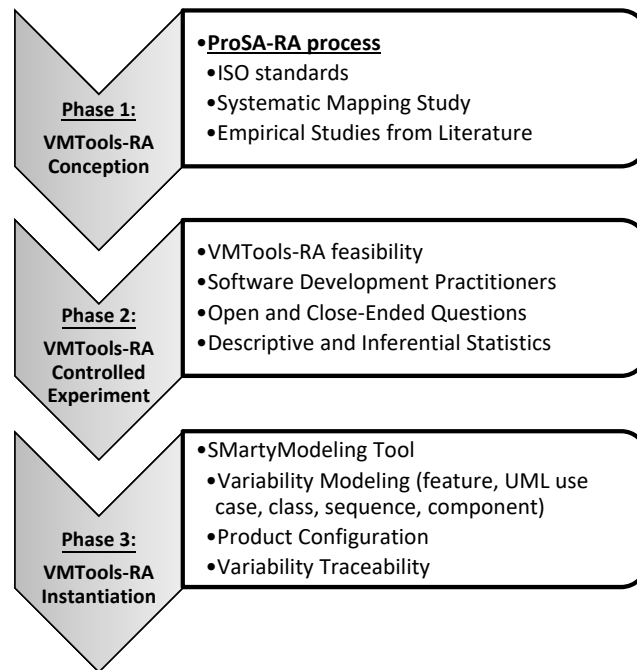


Figure 3: Adopted research methodology

4.1 RA-1: Information Source Investigation

The following three groups of information sources were analyzed: (G1) existing standards in the context of variability management, (G2) a Systematic Mapping Study (SMS) on software variability tools and a survey with experts, and (G3) existing secondary studies on software variability tools in the literature.

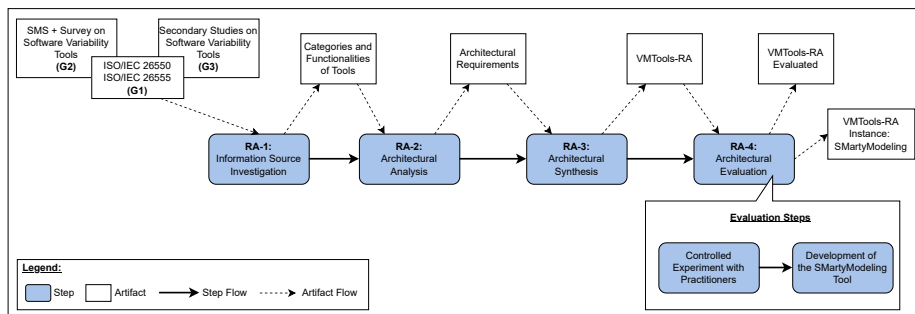


Figure 4: ProSA-RA Followed Methodology

Regarding standards (G1), we extracted information from [ISO/IEC26550, 2015] and

[ISO/IEC26555, 2015]. Such standards deal with processes of variability management and, although they focus on the SPL context, they were relevant for VMTools-RA, once they provide important information about variability management processes (e.g., variability modeling, binding information, traceability, documentation, and variability control and evolution). This information was essential for the understanding of key concepts for the design of VMTools-RA.

An SMS combined with a survey with experts in software variability tools [Allian et al., 2020] was conducted in relation to (G2). The SMS and the survey revealed no single software variability tool can offer all capabilities required by practitioners; few tools support runtime variability, and only two commercial and five research tools are the most preferred ones, as pointed out by practitioners. Moreover, most of the software variability tools support the modeling of variability through Feature-Oriented Domain Analysis (FODA) [Kang et al., 1990]. Several tools were implemented with Java programming language as plugins and have support for XML/XMI files. Integration mechanisms based on ports and interfaces under Internet/Intranet protocols such as HTTP and TCP were also identified. APIs, such as REST and SOAP were used for integrating software variability tools with requirements management tools i.e., Doors². Java APIs (i.e., BIRT for generating reports and Eclipse Modeling Framework (EMF) for modeling variability with feature models) were also identified in such tools. According to the survey, 75% of the tools found in the SMS was acknowledged by industrial practitioners and require more interoperability support for new technologies, collaborative support, and distinct views for assisting specific stakeholders' concerns.

Regarding (G3), relevant Systematic Literature Reviews (SLR) on software variability and SPL tools conducted by [Pereira et al., 2015], [Bashroush et al., 2017], and [Chen et al., 2009] were taken into account, as well as the survey of [Berger et al., 2013]. Such studies were considered, as they provide a set of important categories and functionalities extracted from tools and grouped as follows:

- **Realization:** It is associated with the configuration and derivation of software products making use of variability models. The functionalities identified in this category are (1) Product derivation - which consists of selecting assets to derive a product; (2) Product configuration - which consists of modifying the assets to personalize and derive software product; (3) Binding time - it allows the realization of variability at different development life cycle (i.e., runtime, deploy time, design time, compile time).
- **Interoperability:** It represents the means used to perform the integration between software variability tools. The main functionalities are: (4) Import/Export - it provides import/export function; (5) XML/XMI files - it allows the use of XML/XMI file formats; (6) Integration - it allows interoperability between applications.
- **Modeling:** It represents the mechanisms used to model variability: (7) Feature attributes - they allow the inclusion of specific information for each feature; (8) Composition rules - they represent constraints between features; (9) Mandatory features - they represent features that will always be presented in the products; (10) Variability - it defines the variability information (i.e., Optional, OR, XOR) from each feature.
- **Planning:** It allows collecting, identifying, and representing variability information: (11) Pre-analysis documentation - it allows storing information about the identifica-

² <https://www.ibm.com/docs/en/ermd/9.7.0?topic=overview-doors>

tion of features; (12) Scope definition - it identifies features that should be part of the software infrastructure;

- **Technical information:** It represents functionalities that support the process of variability management, such as code generation and documentation; (13) Product documentation - it provides documentation for each product including the system version; (14) Online availability - it identifies whether the tool is available for online access; (15) Open source - it allows access to the source code of the software; (16) Generation of source code - it is responsible for generating source code based on the variability model.
- **Usability:** It provides mechanisms to assist the user in the process of variability management. It consists of the following functionalities: (17) User guide - it provides documentation to guide the user to use the software variability tool; (18) User interface - it provides a more intuitive environment for users;
- **Validation:** It provides functionalities to perform the validation and analysis of software products. The features identified are: (19) Reports - it allows the generation of reports about variability information; (20) Traceability - it links the existing features of a domain with the requirements; (21) Consistency check - it verifies if the generated domain follows composition rules and dependencies between features.

Most tools include modeling, configuration, and validation features, as shown in Table 1. These features give us an overall view of the main variability management activities covered by software variability tools, which were used as a source of information during the design of VMTools-RA.

4.2 RA-2: Architectural Analysis

Information identified in Step 1 was used for specifying the architectural requirements (AR). The following functionalities were selected for the first group of information sources (G1): domain asset acquisition, variability modeling, composition rules, traceability among the models and domain assets, documentation, consistency check, binding mechanism, control and evolution of variability management, feedback, and trade-off analysis. For the second group (G2), technical information including interoperability among tools and other support solutions (i.e., requirements specification for the collection of domain asset information), functionalities related to usability, persistence, middleware, and versioning of variability models were identified. Finally, the third group of information (G3) involved functionalities related to variability modeling, composition rules, traceability, documentation, consistency check, and binding information.

The three groups of information (G1, G2, and G3) served as the basis for establishing 21 architectural requirements for VMTools-RA. These requirements were divided into two groups: (i) nine architectural requirements related to variability management implementation and (ii) 12 architectural requirements addressed to organizational support and market analysis for the maintenance of quality and evolution of variability management, as shown in Table 2. Each requirement refers to a source of information and most requirements are based on the first group of information corresponding to international standards.

Features																					
	Realization			Interop.			Modeling				Planning		Technical Info				Usability		Validation		
Tools	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)	(21)
CaptainFeature	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				✓	✓			✓		✓	✓
Clafer	✓	✓			✓	✓	✓	✓	✓	✓				✓	✓	✓		✓		✓	✓
Covamof-VS	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓					✓	✓		✓	✓
CVL Tool	✓	✓	✓		✓		✓	✓	✓	✓				✓	✓			✓	✓		✓
CVM	✓	✓			✓	✓		✓	✓	✓			✓	✓	✓		✓	✓		✓	✓
DecisionKing	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓						✓		✓	✓
DOPLER	✓			✓			✓	✓	✓	✓	✓	✓						✓		✓	✓
FAMA		✓	✓	✓	✓		✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓		✓
FeatureIDE	✓			✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓			✓	✓
FMP		✓	✓		✓	✓	✓	✓	✓	✓				✓	✓	✓		✓			✓
FMT		✓		✓	✓	✓	✓	✓	✓	✓					✓	✓		✓		✓	✓
GEARS		✓	✓	✓	✓		✓	✓	✓	✓			✓	✓		✓	✓	✓	✓	✓	✓
GENARCH	✓	✓		✓	✓	✓	✓	✓	✓	✓			✓					✓			✓
Hephaestus	✓	✓		✓	✓	✓				✓								✓		✓	✓
Hydra		✓		✓	✓	✓	✓	✓	✓	✓								✓			✓
Kumbang		✓	✓		✓	✓	✓	✓	✓	✓			✓	✓	✓		✓	✓	✓		✓
LISA toolkit				✓			✓	✓	✓	✓	✓	✓				✓	✓	✓		✓	
Metadoc				✓	✓	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓		✓	✓
PLUM		✓		✓	✓	✓				✓	✓	✓	✓	✓			✓	✓			✓
pure variants		✓		✓	✓	✓	✓	✓	✓	✓			✓	✓		✓	✓	✓	✓		✓
s2t2		✓		✓	✓	✓		✓	✓	✓				✓	✓			✓			
SOASPL				✓	✓	✓				✓						✓		✓		✓	✓
SPLIT		✓		✓			✓	✓	✓	✓				✓				✓	✓	✓	✓
Variamos		✓		✓	✓	✓		✓	✓	✓			✓	✓	✓		✓	✓	✓		✓
Visit-FC		✓						✓	✓	✓								✓			✓
V-Menage		✓						✓	✓	✓								✓			
VMWT		✓	✓				✓	✓	✓	✓								✓			
WeCoTin		✓		✓	✓	✓	✓	✓	✓	✓								✓			✓
XFEATURE		✓		✓	✓	✓	✓	✓	✓	✓	✓			✓	✓			✓			✓

Table 1: Features of variability tools based on functionalities from [Pereira et al., 2015, Lisboa et al., 2010, Bashroush et al., 2017, Berger et al., 2013]

4.3 RA-3: Architectural Synthesis

In the third step of ProSA-RA, requirements previously identified were used for the design of VMTools-RA. Goals, risks, stakeholders, and concerns were also defined for better documentation of the reference architecture and are described as follows: **Goals:** (i) support to the development of software variability tools; (ii) support to software reuse; (iii) support to maintenance and evolution; and (iv) support to integration mechanisms for the collection of domain assets. **Stakeholders:** (i) executives responsible for the organization's business goals; (ii) marketing executives responsible for market demands; (iii) technical managers responsible for available personnel; (iv) software architects responsible for identifying the needs of systems and further reference architecture instantiation; (v) domain experts responsible for providing specific domain information and verifying whether related requirements have been met; (vi) QA

First group of architectural requirements

- [AR.1.1] Manage domain assets (e.g., features, models, requirements, and test cases).
 - [AR.1.2] Build variability models independent from the approach or notation.
 - [AR.1.3] Consider composition rules (e.g., cardinalities and dependencies).
 - [AR.1.4] Manage trace links of variability models with domain assets.
 - [AR.1.5] Document variability models in detail.
 - [AR.1.6] Validate conformance among variability models by using an automatic consistency check.
 - [AR.1.7] Generate reports for checking conflicts and inconsistencies within the variability model.
 - [AR.1.8] Resolve variability appropriately related to binding time.
 - [AR.1.9] Select a variability mechanism for implementing binding time.
-

Second group of architectural requirements

- [AR.2.1] Support change feedback for variability models.
 - [AR.2.2] Implement impact analysis to determine what changes are required.
 - [AR.2.3] Deal with organizations' capabilities for improvement of variability management.
 - [AR.2.4] Support communication and sharing environment.
 - [AR.2.5] Provide guidance to support variability management.
 - [AR.2.6] Get feedback and notify users about variability changes.
 - [AR.2.7] Support trade-off analysis among alternatives of binding time.
 - [AR.2.8] Manage and store multiple versions of variability information.
 - [AR.2.9] Support import/export of variability assets and relevant information.
 - [AR.2.10] Manage the repository of variability information.
 - [AR.2.11] Offer efficiency scalability of feature models.
 - [AR.2.12] Integrate software variability tools through middleware.
-

Table 2: VMTools-RA Architectural Requirements

Manager responsible for ensuring quality requirements; (vii) developers responsible for developing the systems; and (viii) client that uses the system developed. Table 3 shows the **concerns** raised by each stakeholder.

#	Concern
CN1	VMTools-RA must enable efficient reuse of components, documentation, requirements, and knowledge
CN2	Traceability among VMTools-RA and software architecture instances
CN3	VMTools-RA must facilitate the evolution of software variability tools
CN4	VMTools-RA must deal with risk identification and mitigation towards avoiding failures that inhibit business

Table 3: Concerns identified by each stakeholder

[ISO/ISO42010, 2011] represents an important effort toward describing software architectures. As proposed by this standard, the usage of multiple architectural viewpoints is a common practice to document the architectures of software systems. Views are abstractions of particular concerns in the whole reference architecture framework [ISO/ISO42010, 2011]. They are useful to stakeholders who are interested in different cross-sections of a reference architecture.

For this reason, this article introduces three architectural views, namely³: (i) Cross-cutting Viewpoint, (ii) Source Code Viewpoint, and (iii) Deployment Viewpoint.

The following sections present the updated version of VMTools-RA from different viewpoints after its evaluations (see Sections 5 and 6).

4.3.1 General View

The VMTools-RA overview, represented in Figure 5, depicts the main components, their relationship, and their hardware dependencies. We chose not to define the architectural style of VMTools-RA, since most of the software variability tools from our SMS were developed as a plugin for the Eclipse IDE. However, as a suggestion, it is possible to adopt a layered architectural style, a client-server style, and a *Model View Control* (MVC) architectural pattern, as these architectural styles were used to develop some of the software variability tools identified in MS.

In the VMTools-RA overview, four sets of elements can be seen: (i) Variability Management; (ii) Domain Analysis; (iii) Support; and (iv) Organizational. In addition, there is a layer of Middleware and possible types of tools available in the industry/academia to be integrated. Another important element is the Repository which provides mechanisms to persist application information. To identify such elements, we used the architectural requirements from RA-2.

Table 4 presents a mapping among the architectural requirements from Section 4.2 and the elements of the VMTools-RA general view.

4.3.2 Crosscutting Viewpoint

This viewpoint provides general information about reference architectures, including terms and concepts, which are transversal for other views. **Variability View**, which is based on the Cardinality-Based Feature Model (CBFM) notation (Figure 6), describes elements of a reference architecture and the way they can be exercised for building instances of the reference architecture. Such a feature model notation supports the identification of reusable components as optional, mandatory, and OR (select zero or more elements). Some elements were defined as mandatory, e.g. Domain Analysis, responsible for the identification of assets used across all applications, and Variability Modeling, responsible for modeling variability through different modeling techniques. Alternative elements (OR) were also represented, e.g., Consistency Check, which provides different reasoners mechanisms that check the consistency of variability models, and Approach, which supports multiple approaches, (i.e., both Feature Modeling and Ontologies).

Such a variability view is a representation of VMTools-RA through feature models and does not represent a product-line architecture (PLA). Its consistency was verified by SPLOT⁴ a web tool that models, evaluates, verifies consistency, debugs, and shares variability models. According to SPLOT, the variability view of VMTools-RA is consistent and has up to 983,040 valid configurations. This view is concerned with the efficient reuse of components and architectural knowledge (CN1), traceability (CN2), evolution (CN3), and risk identification (CN4).

³ The complete VMTools-RA documentation is available at <http://doi.org/10.5281/zenodo.2210358>

⁴ <http://www.splot-research.org/>

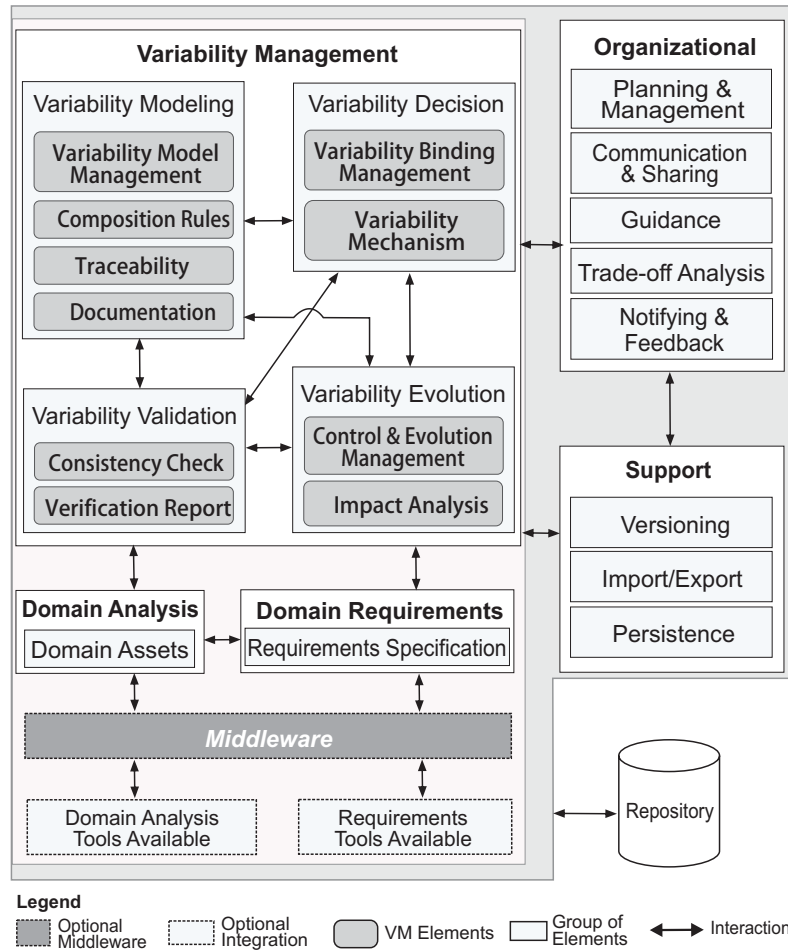


Figure 5: VMTools-RA General View

4.3.3 Source Code Viewpoint

This viewpoint provides specific details, including software structures and modules, regarding the implementation of systems that result from the reference architecture. **Module View**, represented in a UML Class Diagram, can describe specific functionalities through modules (packages), data flow, and interfaces. Technologies presented in this view were extracted from our SMS on software variability tools (Section 4.1), shown in Figure 7. Four modules encapsulate the functionalities of VMTools-RA.

Support module provides technologies for the storage of system data (e.g., information, models, and domain assets) and importing and exporting information. The Organizational Management is responsible for the organizational process of maintaining the quality of software products. It supports an environment of communication and sharing of variability management information through different stakeholders. Domain Analysis is responsible for domain asset acquisition and management and is designed

Requirements VMTools-RA Elements	
AR.1.1	Domain Analysis and Domain Asset Management
AR.1.2	Variability Model Management
AR.1.3	Composition Rules
AR.1.4	Traceability
AR.1.5	Documentation
AR.1.6	Consistency Check
AR.1.7	Verification Report
AR.1.8	Variability Binding Management
AR.1.9	Variability Mechanism
AR.2.1	Control & Evolution Management
AR.2.2	Impact Analysis
AR.2.3	Planning & Management
AR.2.4	Communication & Sharing
AR.2.5	Guidance
AR.2.6	Notifying & Feedback
AR.2.7	Trade-off Analysis
AR.2.8	Versioning
AR.2.9	Import/Export
AR.2.10	Persistence
AR.2.11	Depends on the adopted technologies by the stakeholder
AR.2.12	Middleware

Table 4: Mapping between Architectural Requirements (RA-2) and VMTools-RA Elements

with middleware for integration to different domain analyses and requirement tools. Table 5 shows more details about the domain analysis module and Table 6 provides key activities and technologies required for the acquisition of domain assets.

Variability Management involves four sub-modules: Variability Modeling, which identifies and models variations, traceability, documentation, and composition rules related to constraints dependencies, Variability Validation, which validates variability models by consistency check using arithmetical checkers or logic solvers (Table 7 shows the technologies that can be used in those submodules), Variability Decision, which realizes variability in different binding times (see Table 9), and Variability Evolution, which manages variability evolution. More details are provided in Table 8. Concerns CN1, CN2, CN3, and CN4 from Table 3 were captured for this view.

4.3.4 Deployment Viewpoint

This viewpoint describes hardware elements i.e., server machines, database servers, and client machines.

Deployment View (Figure 8) presents six elements to be connected through interfaces over Internet/Intranet protocols i.e., Application Server supports deploying and system logic. Database Server provides technologies for the storage of domain

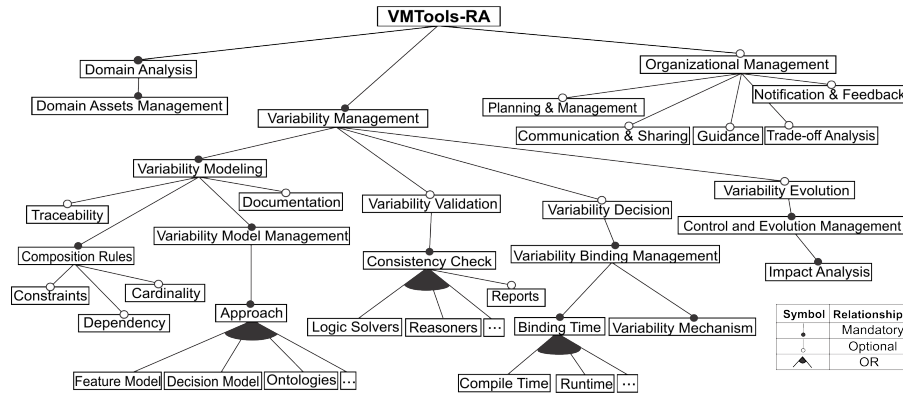


Figure 6: Variability View of VMTools-RA

1 - ASSET BASE

Main Activities:	a) Identification of domain assets (i.e., features, models, requirements, architectural elements, and process description);
	b) Creation/management of different repositories for assets (i.e., a repository of features, models, architectural elements);
	c) Availability of information for stakeholders involved in the project, regardless of the geographic region;
	d) The asset base must contain security policies (i.e., access control, encryption) for maintaining the integrity and availability;
Important:	Our focus is not on the creation and management of assets. ISO/IEC 26555 provides more information about the asset base.

Table 5: Activities for the creation of an asset base

assets, information, and general information. Middleware is an optional element implemented to facilitate the exchange of data among Application Server, Domain Server, and Requirements Server. This view concerns the efficient reuse of components (CN1) and traceability among VMTools-RA and software architecture instances (CN2) (from Table 3).

5 Phase 2: VMTools-RA Evaluation with Practitioners

This section reports on a controlled experiment for investigating the feasibility of VMTools-RA and whether it could benefit the architecture design of software variability tools in terms of time, accuracy, effectiveness, and efficiency. We followed experimental guidelines for controlled experiments from [Wohlin et al., 2012].

5.1 Planning

We provide the planning of our experiment in the next subsections, discussing goals, hypotheses formulation, variables selection, experimental design, instrumentation, and threats to validity.

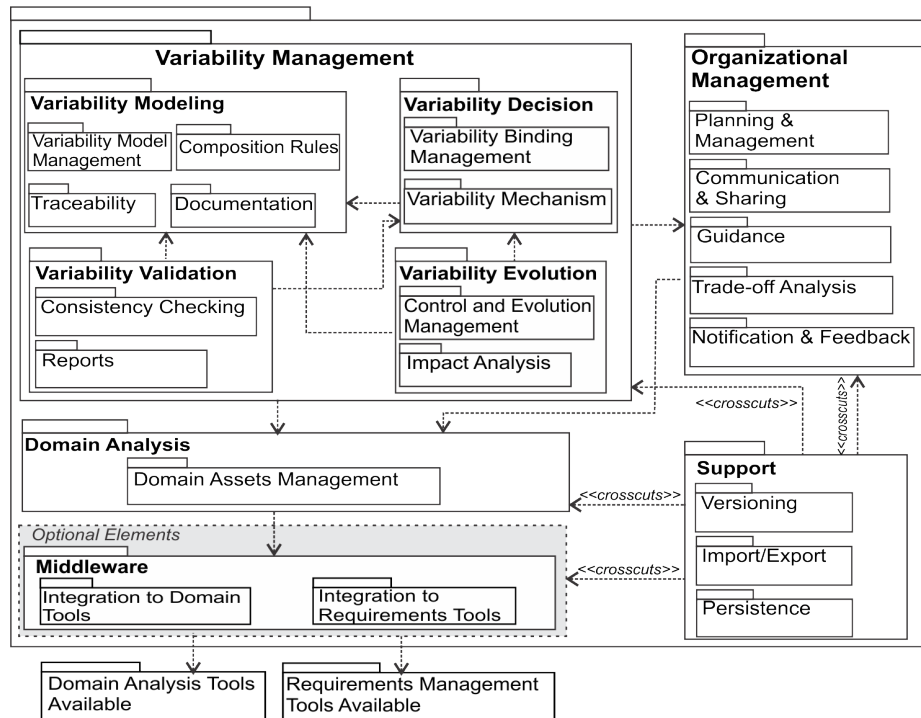


Figure 7: Module View of VMTools-RA

2 - INTEGRATION

Main Activities:	a) Integration with requirements specification tools (i.e., IBM Rational DOORS, CaliberRM, PTC Integrity, Microsoft Office);
	b) Definition of contracts and integration policies with requirements tools for ensuring availability of information;
	c) Definition of intranet/internet ports and technologies for integration (i.e., Rest, Soap, and Webservices) and data security technologies (i.e., encryption and user authentication);
Examples:	Tools pure::variants and Gears tools provide more information about this integration.

Table 6: Activities for the integration of domain analysis tools

5.1.1 Goals

The objective of this experiment is **to analyze** the VMTools-RA feasibility **with the purpose of** characterize it, **with respect to** its use for the design of software architectures in terms of time, accuracy, effectiveness, and efficiency, **from the point of view** of software variability researchers **in the context of** graduate and undergraduate students from the State University of Maringá (UEM) and the University of São Paulo (USP) and practitioners, taking into consideration the Architectural Documentation set (*ArcDoc*).

3 - VARIABILITY MODELING AND VALIDATION

Main Activities:	a) Identification of variability elements (i.e., variation points and variants);
	b) Definition of the approach (i.e., feature model, decision model and ontologies) for modeling variabilities;
	c) Model representation through graphical user interfaces (i.e., UML diagrams) or modeling tools (i.e., EMF or Simulink);
	d) Management of traceability link between variability models and domain assets (i.e., navigation links and notation);
	e) Storage of variability models in repositories (i.e. online repository and cloud);
	f) Management of information on dependency and constraint (i.e., requires and excludes), cardinalities (i.e., min ... max) and variability dependencies (i.e., optional, mandatory, alternative 'OR', 'XOR' or exclusive);
	g) Development or use of consistency checking mechanisms with logical solvers or arithmetic verifiers (i.e., propositional logics with SAT solvers, BDD, CSP, and Descriptive Logic) for validating variability models;
	h) Supply of variability models in different formats (i.e., XML, HTML, and JPG) for modeling variability;
Examples:	Software variability tools with consistency check (i.e., S.P.L.O.T, FAMA, FMP, Hydra, S2T2 and Variamos);

Table 7: Activities for variability modeling and validation

5 - VARIABILITY EVOLUTION

Main Activities:	a) Identification and analysis of requests (i.e., feedback notification) on the impact of information changes;
	b) Changes in variability information (i.e., remove/add variants, variation points) in accordance with business rules;
	c) Storage of versions of variabilities for enabling restoration (i.e., Rollback) in case of conflicts;
	d) Supply of feedback analysis on the evolution of information and variability model for interested stakeholders;
Important:	The evolution of variability is related to the organizational decision and market analysis of the product to be generated;

Table 8: Activities for the variability evolution

An important issue was the choice of the baseline for the comparison of VMTools-RA. Among the possibilities for the design of architectures, we decided to adopt some steps from the software architecture design process, which we named as *ArcDoc*. ArcDoc is defined by the following actives to design software architectures: 1) Analyze scope; 2) Analyze requirements; 3) Identify patterns and modules that could be requiring the design of the software architecture; 4) Identify the relationship of each module; and 5) Design the software architecture in a high-level representation.

We formulated the following research questions (RQs):

RQ1: Does VMTools-RA reduce the time required for the completion of an architecture design of variability tools?

4 - VARIABILITY DECISION

Main Activities:	a) Establishment of policies (i.e., process, documentation) for binding time execution (i.e., compile time, build time and execution time) at different stages of the development lifecycle;
	b) Establishment of an environment with trade-off analysis for evaluation of the alternatives that best fit the business;
	c) Sharing of binding time information with stakeholders;
	d) Definition of mechanisms for implementation of variabilities. Depends on the development lifecycle (Requirements Phase: models diagrams; Architectural Design Phase: composition diagrams and deployment diagrams; Implementation Phase: entity model stereotypes, model-driven approaches, and polymorphism; Test Phase: macros, <code>#ifdef</code> , directives);
	e) Storage of information on binding time, variability mechanisms, and trade-off analysis;
Examples:	Use of binding time: Compilation time (i.e., preprocessed directives); Execution time (i.e., it depends on the condition implemented in the code where features are activated or deactivated); Update Time (i.e., update utilities add functionalities);
Important:	The choice of binding time is independent of the variability model, but a consequence of decisions made from the requirements phase to the execution time. The requirements for flexibility and supporting tools enable the postponement of binding time.

Table 9: Activities for variability decision

RQ2: Does VMTools-RA increase the accuracy of the architectural design of software variability tools?

RQ3: Does VMTools-RA increase the effectiveness of the architectural design of software variability tools?

RQ4: Does VMTools-RA increase the efficiency of the architectural design of software variability tools?

5.1.2 Hypotheses Formulation

The experiment encompasses the following hypotheses related to each RQ (replace “Crit” with “Time”, “Accuracy”, “Effectiveness”, or “Efficiency”):

- **Null Hypothesis (H0Crit):** there is no significant difference in the use of *ArcDoc* or *VMTools-RA* for the design of a software architecture for variability tools (*H0Crit*: $\mu_{ArcDoc} = \mu_{VMTools-RA}$);
- **Alternative Hypothesis (H1Crit):** there is a significant difference in the use of *ArcDoc* or *VMTools-RA* for the design of a software architecture for variability tools (*H1Crit*: $\mu_{ArcDoc} \neq \mu_{VMTools-RA}$).

5.1.3 Correction Criteria

We established the following criteria to evaluate *accuracy* of architecture design:

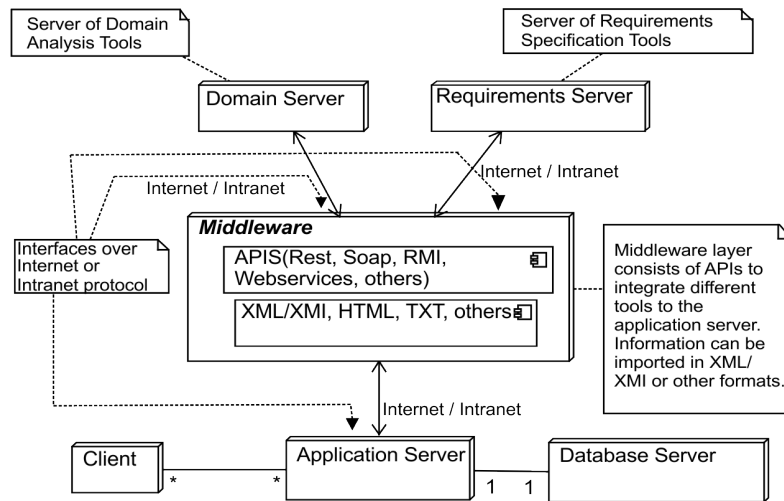


Figure 8: Deployment View of VMTools-RA

1. Do the results of the architectural design satisfy the requirements and architectural description? (Maximum grade of this question is 1.5, where each sub-question has a value of 0.25).
 - (a) Does the architectural design handle variability modeling?
 - (b) Does the architectural design address variability validation?
 - (c) Does the architectural design handle variability decision?
 - (d) Does the architectural design manage domain information?
 - (e) Does the architectural design integrate with an external tool?
 - (f) Does the architectural design make use of middleware?

2. Do the results of architectural design have any conflict compared to requirements and architectural description?

A design conflict refers to situations where the architectural design, description, and requirements are incompatible with each other. When design conflicts are found, we remove 0.25 points for each conflict. Thus, this can lead to negative values when there are more conflicts compared to the total points.

5.1.4 Variables Selection

The Independent variable is the **architecture documentation** used by each participant for the design of a software variability tool architecture. Therefore, it is a factor with two treatments, namely (i) **Document A (VMTools-RA)**; and (ii) **Document B (ArcDoc)**, which is the control.

The experiment also consisted of the following dependent variables: *time* spent by participants for designing a software architecture for a software variability tool, *accuracy*, *efficiency*, and *effectiveness* of design results.

Time spent was measured according to the difference between the final and initial times (converted into minutes) of each participant in resolving tasks of the experiment. The whole time spent on the experiment, including the reading of documentation and designing of architecture was then quantified. We did not allow participants to think-aloud protocols as they were located in the same room at the same time, in their respective universities.

Accuracy is the total number of correct answers (calculated by the sum of all answers given by each participant). $ACCU = \sum correctAnswers$

Effectiveness reflects the correctness of answers (calculated by dividing accuracy by the number of criteria (6 criteria) defined in subsection 5.1.3. $EFFE = \frac{ACCU}{6}$

Efficiency reflects the effectiveness of design results score divided by time in minutes. $EFFI = \frac{ACCU}{TIME}$

5.1.5 Pilot Evaluation

Although our experiment was related to specific research areas including reference architecture and software variability tools, we did not require that participants should have this knowledge. We assume students from software engineering classes have the ability to design software systems by following requirements and scope information. Furthermore, we verified this belief by running two pilot evaluations at USP university with: i) an undergraduate student from the last year of Computer Science. This student declared basic knowledge of UML and architecture design. He also claimed no experience with software variability tools; and ii) a master's degree student in Software Engineering who had advanced knowledge in UML and moderate knowledge about architecture design and software variability. None of the subjects had knowledge of reference architecture. Students had to design an architecture for software variability tool according to the 6 criteria (subsection 5.1.3). Both of them concluded the design activity in almost 30 minutes.

5.1.6 Selection of Participants

The experiment was designed for participants who were taught software architecture design concepts during attendance to advanced Software Engineering disciplines (one year of basics in software architecture, plus additional classes in the next years). By convenience (not randomly), we chose 36 students, 18 from the State University of Maringá (UEM)⁵ (15 undergraduate students attending the last year of Bachelor in Computer Science and three Master's degree students), and 18 students from the University of São Paulo (ICMC-USP)⁶ (four Master's degree students, nine Ph.D. candidates, and five undergraduate students attending the last year of Bachelor in Computer Science).

5.1.7 Choice of Experimental Design

According to the variables selection (Section 5.1.4), the type of design of this experiment is one factor with two treatments. As independent variables have no relation to each

⁵ <http://www.uem.br>

⁶ <http://www.icmc.usp.br>

other, T-test or Wilcoxon hypothesis tests might be used, according to the normality of samples.

We divided the experiment into two parts: the first one was performed at UEM by Ana P. Allian under the supervision of Prof. Edson Oliveira Jr, during a software engineering course in the classroom, whereas the second one was performed at ICMC-USP by Bruno B. Sena and Ana P. Allian under supervision of Prof. Elisa Y. Nakagawa in the Software Engineering Laboratory (LABES).

The use of VMTools-RA is quantified by the *time* spent to design an architecture of software variability tool, and the *accuracy* of results. The detailed steps to conduct this experiment are specified below (see Figure 9): Divide the experiment into two parts: Part 1 performed at UEM University, and Part 2 performed at USP University; Conduct a pilot evaluation with two students at USP University; Give training course for at least 50 minutes to level students; Give interval of 10 minutes for students; Distribute two types of documents to provide objects randomization; Evaluate *accuracy* of design results submitted by participants quantitatively to accept or reject the hypothesis.

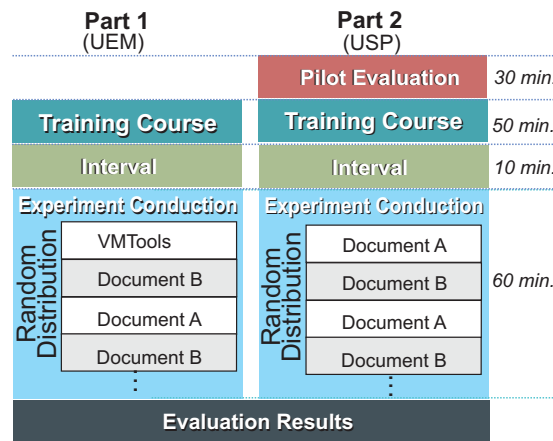


Figure 9: Experiment Design Steps

We could not apply randomization for selecting participants due to their availability for this experiment. However, we applied randomization during the interleaved distribution of “Document A” and “Document B” (see Section 5.1.8) for participants. Thus, we had a balanced sample: 18 participants for treatment “Document A” and 18 for treatment “Document B”.

5.1.8 Instrumentation

We created the following instruments for this experiment: a participant’s Characterization Questionnaire with regard to experience in software architecture, software variability, and UML; a Free Consent Term of Participation in the experiment for each participant; training session materials on software variability, architecture design, software variability tools, and UML package diagrams; “Document A” with VMTools-RA guidelines,

including a software variability tool to design, scope definition, glossary of terms, requirements, and VMTools-RA documentation; “Document B” with ArcDoc guidelines with a software variability tool to design, scope definition, glossary of terms, and requirements without VMTools-RA documentation; a questionnaire with multiple-choice and open questions to gather participants qualitative perspective during architectural designing.

5.1.9 Validity Evaluation

This section discusses threats to the validity of the controlled experiment, including mitigation plans for handling each of them.

An important threat to **internal validity** might be the experiment duration, which might cause fatigue effects on the participants. The whole experiment was designed to be performed in at least 110 minutes: 50 minutes for training session plus 60 minutes for experiment tasks. We believed 60 minutes is enough due to the characteristics of the single software architecture to be designed. To mitigate such threat, we allow participants to have a brake of 30 minutes to have a meal, go to the bathroom, or just “take a breath”. In addition, the very same training course offered with the same material and time at both universities help to mitigate this internal threat, as well as perform the experiment in only one day. We ask participants to not communicate with each other during break and experiment tasks to avoid influencing the outcome of the study. Thus, we introduced a human being observer, which might be a threat to **construct validity**. To control subjects’ different behavior when observed, we performed pre-tests during the training session to adapt them to the experiment tasks. Another threat to **internal validity** is related to measurements of time and accuracy for the design of software architectures. With regard to the time measurement, we set the same start time for all participants in the first part of the experiment performed at UEM. At USP, once the participants could start at different times, they were asked to register their starting times. Once the experiment had started, we monitored the participants for avoiding any communication among them. The accuracy of results was measured by two criteria and six subcriteria, which reflects a common understanding of the researchers about the software architecture design, as defined in Section 5.1.4.

A threat to **external validity** is related to the total number of participants that may have affected the representativeness of participants. We tried to invite as many students and practitioners as available with minimum knowledge. Furthermore, there are no participants exclusively from the industry. Students from two different universities are invited towards mitigating this threat as they form a heterogeneous sample. Master’s students, Ph.D. candidates, and undergraduates in the last year of the Computer Science course are invited. As stated by [Falessi et al., 2017], the participation of students in experiments is a valid resolution of the real-world need in a lab context. Furthermore, [Salman et al., 2015] claimed in their controlled experiment that when a new approach or technique is applied for both participants (students and professional from industry) no significant difference in their performances are usually observed. Heterogeneity of sample might be a threat to **conclusion validity**. According to [Höst et al., 2000], in the context of project impact assessment, only minor differences are observed between the conception of students and practitioners and there is no significant difference between the correctness of experimental tasks of students and professionals.

A threat to **conclusion validity** is concerned about the experiment involving a reduced number of participants from academia, the result may not be conclusive, but indicative. Empirical evaluations with practitioners from the industry who deal with software variability tools would be more conclusive, however, we believe our results are

important for the software engineering community and for industrial sets because they provide insights from undergraduate, Master's, and Ph.D. candidates who are going to be the next generation of industrial practitioners.

5.2 Operation

Operation is discussed in the next subsections in terms of preparation, execution of the experiment itself, and data validation.

5.2.1 Preparation

We performed the following activities as preparation for the experiment execution: **Training Session:** we conducted a training session on basic concepts of software variability, architecture design, software variability tools, and UML package diagrams. All concepts were summarized in a paper sheet and distributed to all participants. The session took at least 50 minutes and was conducted by the same instructor and with the same contents in both universities. After the training session, another 10 minutes were spent on a pre-test of five simple questions about the concepts taught to the students for verifying whether they had understood them. **Note, we did not introduce reference architecture concepts during our training course. The aim of this experiment was for participants to design a concrete architecture based on documentation they have available, which could or not include a reference architecture.**; and **Distribution of Instrumentation:** we distributed instrumentation from Section 5.1.8 to each participant in a random and interleaved way, thus reducing potential threats and providing a balanced sample. All participants started at the same time.

5.2.2 Participation Procedures

Participants followed several steps during the execution of this experiment: participants were oriented to design a concrete architecture according to the documentation they have available (ArcDoc or VMTools-RA); the experimenter annotated the initial experiment time at UEM and asked participants to annotate their initial time at USP; participants read and signed the Consent Term; participants read and filled in the Characterization Questionnaire; experimenter distributed "Document A" and "Document B" randomly to each participant; participants read respective "Document A" or "Document B"; participants design an architecture for a specific software variability tool according to the documentation they have available, which included an example scenario, a list of requirements, and specific information related to the software variability tool. Participants were oriented to finish this activity when they felt all requirements defined in the documentation were attended; participants answered a questionnaire with multiple-choice and open questions to gather participants' qualitative perspective during architectural designing; the experimenter annotated the experiment end time of each participant; the experimenter validated data produced by each participant.

5.2.3 Data Validation

Validation of data was performed by following criteria defined in Section 5.1.4 for time, accuracy, efficiency, and effectiveness.

5.3 Analysis and Interpretation

Table 10 shows *Time (TIME)* spent, *accuracy (ACCU)*, *efficiency (EFFI)*, and *effectiveness (EFFE)* observed values for each participant. Particip. means participant ID and Type is the Documentation set used: VMTools-RA is “Document A” and ArcDoc is “Document B”.

Particip. ID	Venue	Doc. Type	TIME	ACCU	EFFI	EFFE
S1	UEM	VMTools-RA	30	0.50	0.01667	0.08333
S2	UEM	ArcDoc	25	0.50	0.02000	0.08333
S3	UEM	VMTools-RA	30	0.75	0.02500	0.12500
S4	UEM	ArcDoc	17	0.50	0.02941	0.08333
S5	UEM	VMTools-RA	30	1.25	0.04167	0.20833
S6	UEM	ArcDoc	09	-1.25	-0.13889	-0.20833
S7	UEM	VMTools-RA	18	1.25	0.06944	0.20833
S8	UEM	ArcDoc	06	0.50	0.08333	0.08333
S9	UEM	VMTools-RA	18	0.50	0.02778	0.08333
S10	UEM	ArcDoc	15	0.50	0.03333	0.08333
S11	UEM	VMTools-RA	16	1.25	0.07813	0.20833
S12	UEM	ArcDoc	15	0.25	0.01667	0.04167
S13	UEM	VMTools-RA	19	0.75	0.03947	0.12500
S14	UEM	ArcDoc	10	-0.50	-0.05000	-0.08333
S15	UEM	VMTools-RA	19	1.25	0.06579	0.20833
S16	UEM	ArcDoc	15	-1.00	-0.06667	-0.16667
S17	UEM	VMTools-RA	16	1.00	0.06250	0.16667
S18	UEM	ArcDoc	15	-0.75	-0.05000	-0.12500
S19	USP	VMTools-RA	28	1.00	0.03571	0.16667
S20	USP	ArcDoc	63	0.25	0.00397	0.04167
S21	USP	VMTools-RA	50	0.50	0.01000	0.08333
S22	USP	ArcDoc	53	-1.25	-0.02358	-0.20833
S23	USP	VMTools-RA	44	0.50	0.01136	0.08333
S24	USP	ArcDoc	14	0.50	0.03571	0.08333
S25	USP	VMTools-RA	30	1.25	0.04167	0.20833
S26	USP	ArcDoc	33	1.00	0.03030	0.16667
S27	USP	VMTools-RA	48	0.75	0.01563	0.12500
S28	USP	ArcDoc	38	0.50	0.01316	0.08333
S29	USP	VMTools-RA	45	1.00	0.02222	0.16667
S30	USP	ArcDoc	35	0.25	0.00714	0.04167
S31	USP	VMTools-RA	40	1.25	0.03125	0.20833
S32	USP	ArcDoc	41	0.50	0.01220	0.08333
S33	USP	VMTools-RA	47	0.75	0.01596	0.12500
S34	USP	ArcDoc	45	-1.00	-0.02222	-0.16667
S35	USP	VMTools-RA	48	1.00	0.02083	0.16667
S36	USP	ArcDoc	29	0.50	0.01724	0.08333

Table 10: Descriptive Statistics of Results from UEM and USP.

5.3.1 Descriptive Statistics

Table 11 shows the descriptive statistics for the experiment. Mean, median, standard deviation, maximum, and range values were calculated from each dependent variable. Descriptive statistics enable measurements of the difference between two treatments for each variable. For instance, mean values of variable *TIME* from VMTools-RA (32.00000)

and *ArcDoc* (26.55556) were used to find the percentage⁷ of difference between the two treatments. For variable *TIME*, the VMTools-RA group spent 9.3% more time designing architecture in comparison to *ArcDoc*. However, in terms of accuracy *ACCU*, efficiency *EFFI*, and effectiveness *EFFE*, the participants who used VMTools-RA achieved better results, as shown in Table 11. Some values for *ACCU*, *EFFE*, and *EFFI* are negatives, and they refer to situations where the architectural design, description, and requirements were incompatible with each other. When design conflicts were found, we removed 0.25 points for each conflict.

VMTools-RA	Mean	Median	Min	Max	Range	Lower Quartile	Upper Quartile	Quartile Range	Std.Dev.	Standard Error
TIME	32.00000	30.00000	16.00000	50.00000	34.00000	19.00000	45.00000	26.00000	12.63050	2.977036
ACCU	0.91667	1.00000	0.50000	1.25000	0.75000	0.75000	1.25000	0.50000	0.29704	0.070014
EFFI	0.03506	0.02951	0.01000	0.07813	0.06813	0.01667	0.04167	0.02500	0.02121	0.004999
EFFE	0.15278	0.16667	0.08333	0.20833	0.12500	0.12500	0.20833	0.08333	0.04951	0.011669

ArcDoc	Mean	Median	Min	Max	Range	Lower Quartile	Upper Quartile	Quartile Range	Std.Dev.	Standard Error
TIME	26.55556	21.00000	6.00000	63.00000	57.00000	15.00000	38.00000	23.00000	16.47894	3.884124
ACCU	0.00000	0.37500	-1.25000	1.00000	2.50000	-0.75000	0.50000	1.25000	0.73264	0.172685
EFFI	-0.00272	0.01268	-0.13889	0.08333	0.22222	-0.02358	0.02941	0.05300	0.04978	0.011734
EFFE	0.00000	0.06250	-0.20833	0.16667	0.37500	-0.12500	0.08333	0.20833	0.12211	0.028781

Table 11: Descriptive Statistics for the two Treatments (VMTools-RA and ArcDoc)

The boxplots summarize the architecture design results for each treatment showing the median and the bottom and top quartiles (25 percentiles and 75 percentiles), respectively [Wohlin et al., 2012]. Values outside this range are known as outliers and indicate a strong deviation from other observations in the sample. The VMTools-RA group, showed no outliers, as illustrated in Figures 10a), 10b), 10c), and 10d); however, the *ArcDoc* group found a negative outlier from a participant (S6) (see Figure 10c) for variable *EFFI*. This participant had superficial knowledge about software variability, which might explain an unsatisfactory result. Despite this deviation value, the outlier was not removed because it exerted no adverse effect on the amount of data available.

We plot a radar graph represented in Figure 11 to display an overview of variables *TIME*, *ACCU*, *EFFI*, and *EFFE* from the two treatments used in this experiment. Values were converted to a scale from -10 to 100 (i.e. *ACCU* multiplied by 10, *EFFI* multiplied by 100, and *EFFE* multiplied by 100) aiming to fit all variables and treatments in the same graph.

The dotted line indicates that VMTools-RA clearly achieves more effectiveness, accuracy, efficiency, and more time during the controlled experiment compared to *ArcDoc*, represented with a darker line.

5.3.2 Normality and Hypothesis Tests

This section details our statistical hypothesis test that compared two independent samples (VMTools-RA and *ArcDoc*). Shapiro-Wilk test [Wohlin et al., 2012] was applied for identifying the most adequate statistical hypothesis test sample. Data are normally

⁷ The percentage calculus for variable *TIME*: $\frac{((32.0000 - 26.55556) * 100)}{(32.0000 + 26.55556)} = 9.3\%$

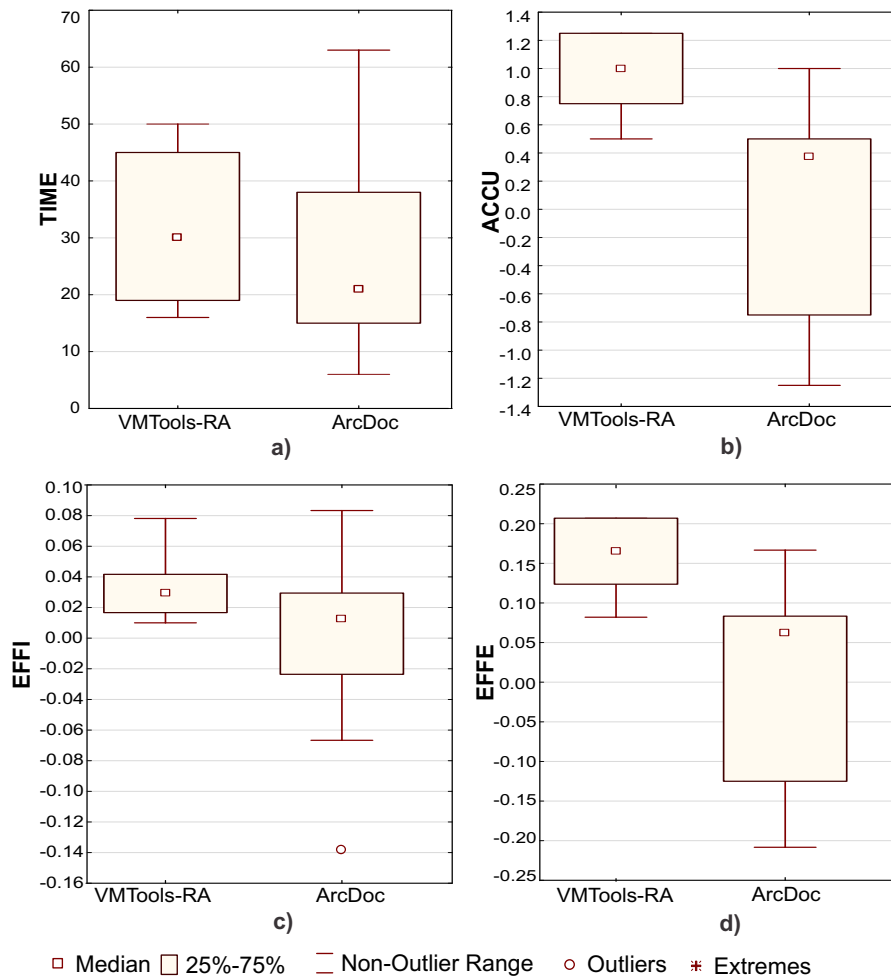


Figure 10: Boxplots: a) Time; b) Accuracy; c) Efficiency; d) Effectiveness

distributed when p -value is higher than 0.05 ($\alpha > 0.05$); therefore, parametric tests, for instance, t -test can be applied. The data distribution is not normal when p -value is lower or equal to 0.05. In this case, nonparametric tests, including Mann-Whitney [Wohlin et al., 2012], are used. Table 12 shows *Shapiro-Wilk* normality test. Once different types of distribution were provided for our variables, we applied t -test for *EFFE* and *Mann-Whitney test* for *TIME*, *ACCU*, and *EFFI*. According to [Wohlin et al., 2012], when there is the need to compare two types of distribution (normal and non-normal) as we have in this study for variable *TIME*, a nonparametric technique is applied. We performed the statistics tests with IBM Statistics SPSS tool⁸.

Results from t -test for variable *EFFE* (Table 13) indicated p -value was lower than

⁸ <https://www.ibm.com/products/spss-statistics/support>

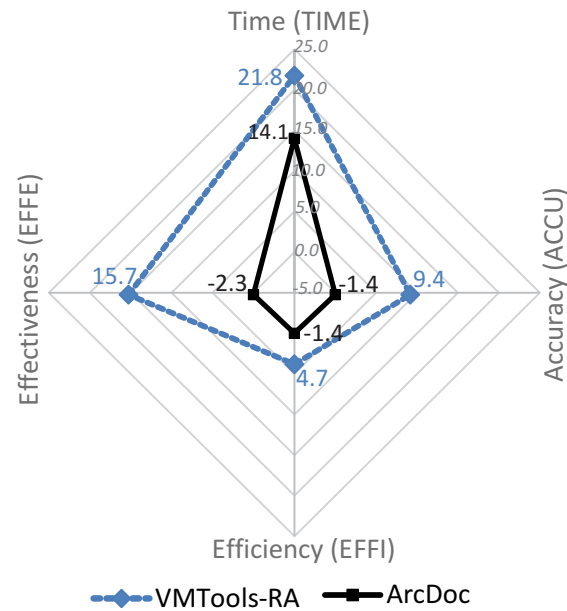


Figure 11: Radar chart for architectural design analysis (Values converted to a scale ranging from -5 to 25)

Variables	VMTools-RA		ArcDoc	
	<i>p</i> -value	Distribution	<i>p</i> -value	Distribution
TIME	0.02054	non-normal	0.10003	normal
ACCU	0.00726	non-normal	0.00181	non-normal
EFFE	0.05459	normal	0.05247	normal
EFFI	0.00726	non-normal	0.00181	non-normal

Table 12: Shapiro-Wilk normality test

0.05, therefore, null hypothesis $H_0 Effe$ could be rejected for *EFFE*, once VMTools-RA and *ArcDoc* showed a statistically significant difference between them.

Results from *Mann-Whitney test* for variable *TIME* produced a *p*-value higher than 0.05 ($p = 0.09$), which indicates null hypothesis $H_0 Time$ for variable *TIME* could not be rejected. It means there is no statistical difference at using VMTools-RA and *ArcDoc* in terms of time to design a software architecture. However, when variables *ACCU*, *EFFE*, and *EFFI* are considered, *p*-value is significantly smaller than 0.05. The results in Table 13 and those from descriptive statistics (Table 11) revealed null hypotheses $H_0 Accu$, $H_0 Effe$, $H_0 Effi$ can be rejected due to a statistically significant difference between the two treatments. Moreover, the mean and median of each variable are much higher for VMTools-RA treatment.

Variables	<i>p</i> -value	Hypothesis test	Reject Null Hypothesis?
TIME	0.099928	Mann-Whitney test	No
ACCU	0.000018	Mann-Whitney test	Yes, accept H1Accu <i>ArcDoc</i> < VMTools
EFFE	0.006205	t-test	Yes, accept H1Effe <i>ArcDoc</i> < VMTools
EFFI	0.000022	Mann-Whitney test	Yes, accept H1Effi <i>ArcDoc</i> < VMTools

Table 13: Results from Hypothesis Tests

5.3.3 Qualitative Analysis

Our experimental study also included means of collecting qualitative insights from participants. We designed a short questionnaire with multiple-choice questions and open questions to gather participants' perspectives during architecture design. As the data is qualitative, the results are indicative.

With respect to the multiple-choice question, we asked participants about the level of difficulties faced by them during the design of the software architecture with and without VMTools-RA. Most participants (77.8%) declared they found the design of software architecture a moderate to difficult task and only 22.2% of participants found it easy. Some answers given by participants related to their difficulty to design a concrete architecture are summarized as follows: *S11, S35, and S13 - I missed more information to design a concrete architecture with VMTools-RA; S23 - Deeper knowledge about software architecture area and UML package diagrams; S1, S25, and S33 - Description of what each component is and what each component does, what are the dependency relationships between them. S7 - A Tool to model instead of drawing on paper.*

Table 14 summarizes our results.

Level of difficulty	VMTools-RA	ArcDoc	Total
Hard	9	12	21
Moderate	7	0	7
Easy	2	6	8

Table 14: Level of difficulty to design the software architecture

We also elaborated on specific questions to understand participants' perceptions of each treatment. With respect to *ArcDoc*, we asked 18 participants whether they missed a support structure to guide them during architecture design. As result, 55.5% of the participants declared that an infrastructure was not necessary, and 44.4% of participants said they missed a supporting infrastructure during architecture design.

With respect to participants who used VMTools-RA, we asked them whether the reference architecture was useful during the design activity. 95.0% of participants (out of 18 participants) stated it helped them during design tasks and only one participant (5.5%) declared VMTools-RA was not useful, which is detailed as follows: *S21 - "I*

have no knowledge to present the elements that are part of the application layer of the architecture, as I have no knowledge about sw architecture”.

For the open questions, we gathered qualitative data from all answers discussed by students for each treatment. We employed Grounded Theory procedures by using Open Coding and Axial Coding. Open Coding is an analytical process in which concepts are identified and separated into discrete parts for analysis, comparison, and categorization of data. It can be performed manually through the reading of recovered data and grouping similar information into codes. Axial Coding handles connections among codes identified in the previous process (Open Coding) and groups them according to their properties for representing categories [Strauss and Corbin, 1998]. We performed the identification of categories and codes using a tool for qualitative analysis⁹ for each treatment. We reported in this section some quotes of statements declared by participants.

With respect to VMTools-RA, we identified two categories and 15 codes as presented in Figure 12. In the such figure, VMTools-RA is connected to categories “provide support to” and “missing information”. These categories were identified according to the questions: 1) *How did VMTools-RA help you?* and 2) *Did you miss any support infrastructure?*, respectively.

Following are the main categories and codes reported through participants’ discussions. We identified the following codes for category **Provide support to**: Everything: “S35: *It helped in everything, it was the basis for the design.*”; Organization: “S11: *It supports the organization of modules that the software needs to perform in the application*”; Dependencies: “S31: *It helped me understand what elements are needed, how they are connected, and the dependency between them.*”; Representation: “S7: *It helped to get an idea of how to organize and how to represent the module.*”; Relationship: “S3: *It helped to understand the cycle of interactions between the elements*; Identification: “S1: *It helped me to identify the key elements to be considered at the application layer.*”.

For category **Missing information**, we have the following codes: Knowledge about software architecture: “S11: *I missed further software architecture knowledge for better design a tool.*” Training course: “S29: *I missed a training course step-by-step about how to design the elements in a package diagram.*”; Dependencies: “S15: *I missed definition about integration an dependencies.*”; Reference Architecture Concepts: “S13: *I missed an explanation about how to use the VMTools-RA.*”; Integration: “S15: *It misses definition about integration and dependencies.*”; Tool for modeling: S7: “*I missed a tool for modeling.*”; Granularity level: “S3: *I found it difficult to understand the level of abstraction for the modeling.*”; Relationship: “S1: *A set of possible relationships could be presented for package diagrams. In my evaluation, I would only consider a persistent relationship.*”.

Overall, we have observed that VMTools-RA supports many steps related to the design of a software architecture for VMTools-RA; however, some participants still missed more information about the reference architecture concept. This is explained due to the fact that we did not treat concepts about reference architecture in our training course. In addition, we believe an instantiation guide to using VMTools-RA is necessary.

Regarding *ArcDoc*, most of the participants did not answer the open question; thus, we only identified one category and five codes as shown in Figure 13. In such figure, *ArcDoc* is connected to the category “software artifact” that was identified according to the question: 1) *What kind of software artifact would support you during the design task?*.

Considering *ArcDoc*, we identified the category **Software artifact** and the following

⁹ <http://www.maxqda.com>

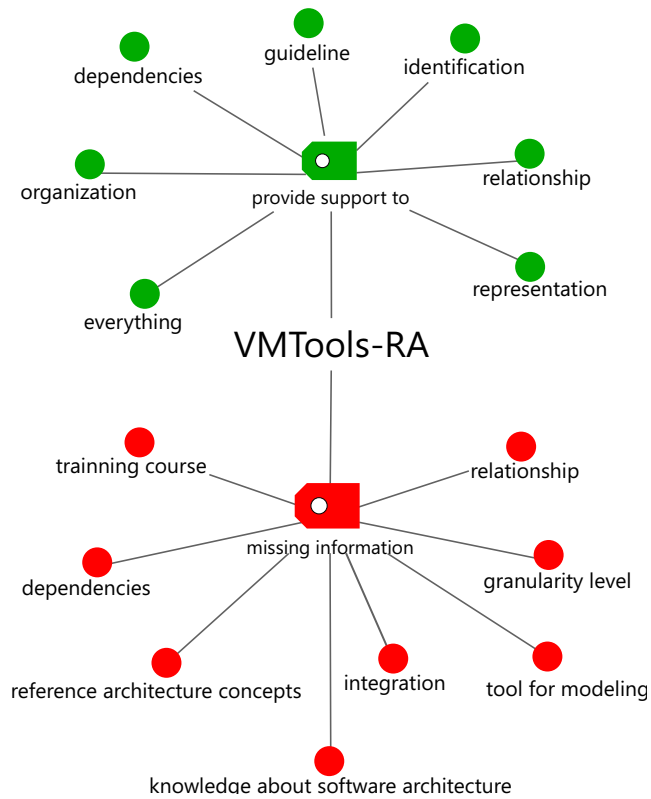


Figure 12: Graph representation from the two categories and codes identified for VMTools-RA

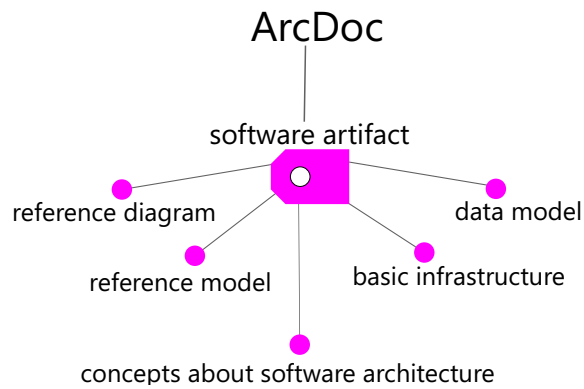


Figure 13: Graph representation from the category and codes identified for ArcDoc

codes: Reference model: “S30: A reference model would be very useful for designing the software variability tool.”; Reference diagram: “S18: I missed a reference diagram or a

use case diagram describing the steps to design an architecture.”; Data model: “S32: *A data model presenting the information flow in the main modules.*”; Concepts about software architecture: “S22: *I missed a greater knowledge about software architecture and motivation for each element*”; Basic infrastructure: “S20: *A basic structure to support the design.*”

In summary, the qualitative evaluation from *ArcDoc* provides evidence about the need for an infrastructure to better complete the design task.

5.4 Presentation and Package

We organized the package of this experiment by building a static HTML page with all instrumentation and experimental datasets, which is publicly and permanently available at <https://doi.org/10.5281/zenodo.2352150>. We followed the main principles of Open Science, such as open data and open methodology, for allowing the reproducibility and auditability of experimental studies.

5.5 Discussion of Results

We conducted a controlled experiment for analyzing the feasibility of VMTools-RA for the design of architecture for software variability tools in terms of time, accuracy, effectiveness, and efficiency. Experimental results indicate participants who used VMTools-RA achieved better accuracy in the design of the software architecture in comparison to participants who did not use it. Those who used VMTools-RA outperformed all evaluation criteria presented in Section 5.1.4. However, results confirmed participants who used VMTools-RA spent more time completing the architecture design activity. We believe the lack of training about reference architectures may have affected the time required for such an activity.

As described in Section 5.1.6, participants had different experiences in software architecture and software variability. According to the results, they benefited from using VMTools-RA for designing software variability tools architecture compared to the usage of *ArcDoc*. A comparison of results from those participants revealed graduate participants achieved significantly better design results than undergraduate ones, which implies the necessity of training on the design of architectures for inexperienced undergraduate students when no source of design support is available.

VMTools-RA provides the main functionalities and elements for the development of different software variability tools; however, we are aware some functionalities and capabilities are still missing in our reference architecture, as we need more evaluations with experts to provide more accurate results of VMTools-RA use. In addition, our research group intends to develop a software variability tool based on the VMTools-RA for providing more evidence about the effectiveness of the proposed approach in the design of a new software variability tool.

The main lessons learned from this experiment are discussed as follows.

The Controlled Experiment: the experiment conducted investigated the benefits associated with the use of VMTools-RA for the design of an architecture for software variability tools. It was performed on an academic background with students from two different universities and with different academic levels. This experiment supports our hypothesis on the advantages of using VMTools-RA with respect to the accuracy of design results. The quantitative results show VMTools-RA did not impact favorably on the time spent on the architectural design of variability management tools. However, regarding

accuracy, effectiveness, and efficiency, VMTools-RA impacted positively, although the participants were not experts in reference architecture and software variability tools. Experiments are necessary for industrial contexts, however, we believe this controlled experiment provides evidence the reference architecture can support some stakeholders during their daily work activities.

Impact of Students Skills: the experiment was conducted with undergraduate and graduate students. The graduate ones achieved better results with respect to accuracy, effectiveness, and efficiency in the design of software architecture. However, they took more time in each activity, which might indicate more commitment to the experiment in comparison to the undergraduates. We believe results from undergraduate students are also significant as stated by [Daun et al., 2015, Falessi et al., 2017].

Recommendations for Researchers and Practitioners: the design of software architecture can be very hard for students of software engineering courses. For this reason, we recommend conducting a more detailed training course with practical exercises about concepts related to software architecture design and UML concepts.

6 Phase 3: VMTools-RA Instantiation for Developing the SMarty-Modeling Tool

SMartyModeling [Silva et al., 2022, Silva and Oliveira Jr, 2021] is an environment for engineering UML-based SPLs in which variabilities are modeled as stereotypes using any UML-compliant profile. We are currently adopting the SMarty approach [Oliveira Jr et al., 2010] for variability management.

The environment supports feature, use case, class, component, sequence, and activity diagrams. It has as main features in its current version¹⁰: variability modeling and constraining, matrix-based support to traceability among SPL elements, and specific product configuration, SPL evaluation, and information export/import.

6.1 VMTools-RA Instantiation and Decisions

We selected the following VMTools-RA architectural requirements from Table 2 for developing SMartyModeling according to its planned features, as follows: **(AR.1.1)** Manage domain assets (e.g., features, models, requirements, and test cases); **(AR.1.2)** Build variability models independent from the approach or notation; **(AR.1.3)** Consider composition rules (e.g., cardinalities and dependencies); **(AR.1.5)** Document variability models in detail; **(AR.1.6)** Validate conformance among variability models by using automatic consistency check; **(AR.2.2)** Implement impact analysis to determine what changes are required; **(AR.2.5)** Provide guidance to support variability management; **(AR.2.9)** Support import/export of variability assets and relevant information; and **(AR.2.11)** Offer efficiency scalability of feature models.

We designed the SMartyModeling architecture according to the VMTools-RA description of elements and views. As VMTools-RA is widely designed for software variability, we decided to develop an environment for SPL variability. VMTools-RA does not specify a specific architectural pattern or style. Therefore, we built SMartyModeling under the Model-View-Controller (MVC) pattern.

¹⁰ Demo video at <https://youtu.be/Rt2wlrPSOI4>

The SMartyModeling architecture is composed of classes and interfaces organized in four main packages: Model, Controller, View, and File. The latter has two sub-packages: Export and Import. Figure 14 depicts the SMartyModeling high-level package diagram.

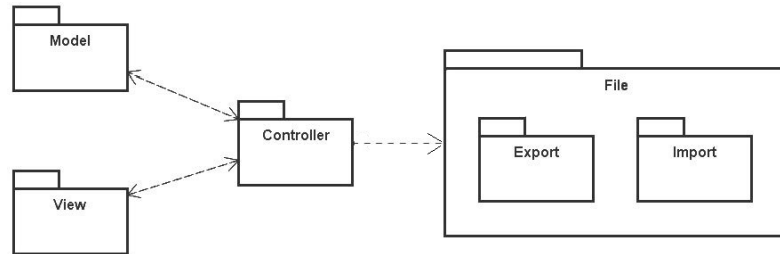


Figure 14: Logical Architecture of SMartyModeling

To build such an architecture, we adapted certain VMTools-RA views and elements for the context of SPL specifically aiming at: identifying, constraining, representing, and tracing variabilities. Figure 15 depicts VMTools-RA elements and views instantiated for SMartyModeling and a brief description of the decisions made for the SMartyModeling architecture. Such a figure is a representation of a VMTools-RA instance in the perspective of the Module View of Figure 7.

The activities described by VMTools-RA (Tables 4, 5, 6, 7, and 8) were essential to gather up information and determine the organization of concerns in the structure of the environment during the requirements phase.

Regarding the assets base (described in Table 4), the identification of domain assets is constrained to the scope of the architectural elements, in particular, the use case, class, component, activity, and sequence diagrams following the UML metamodel and offering support to SPL concepts. SMartyModeling allows asset control over traceability among elements, products, and instances, as well as SPL metrics. Despite allowing assets import and export, the environment does not provide direct integration with repositories or asset access policies.

Activities related to the integration of SMartyModeling with other tools (Table 5) are more complex due to the fact that there are multiple SPL tools, aimed at different purposes, with particular file format management. Therefore, we keep SMartyModeling project information in a file with the .smt extension, in which all the information is hierarchically organized throughout an XML file. The model with the export format is available to support the processing of such data. We already developed an integration of SMartyModeling to FeatureIDE¹¹ for both importing and exporting feature models. We are also integrating SMartyModeling to DyMMer [Bezerra et al., 2021] to evaluate the quality of feature models and their maintainability decisions throughout metrics.

Variability modeling and validation (Table 6) are significant features for SMartyModeling and have been adapted for the SPL context. The structure of the Variability class includes attributes such as name, variation point, variants, constraint, minimum, and maximum values. The framework for variability and dependency and uniqueness associations was built based on the SMarty approach, including the stereotypes described

¹¹ <https://www.featureide.de>

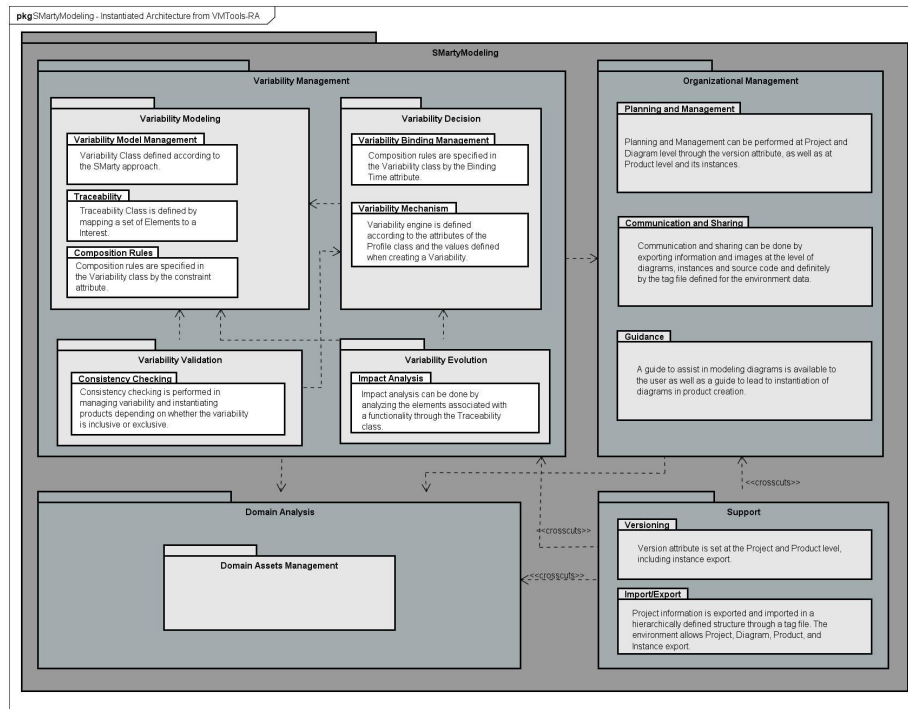


Figure 15: VMTools-RA Module View Instance for SMartyModeling

by the SMartyProfile. The variabilities are represented in the modeling panels of the UML diagrams. Consistency checking is performed at runtime as variability is created or updated. The variabilities are included in the export file as part of a diagram. SMartyModeling also allows the export of images containing the visual representation of the variability in a given diagram.

Regarding variability evolution (Table 7), the environment supports eventual changes regarding information about variability, in particular, changes in variants and variation points. The impact analysis on possible changes can be performed by defining traceability, however, this process requires the user to manually indicate which elements are related to a particular functionality or interest.

For variability decisions (Table 8), the environment supports diagram instantiation and product generation by means of a variability resolution process, thus allowing the user to complete the instantiation process respecting the constraints and the selected variants. The variability class has the binding time attribute, which holds information on the moment of the SPL life cycle that a given variability must be resolved.

6.2 SMartyModeling Design and Implementation

Figure 16 depicts the internal organization of the `model` package aiming at the separation of concerns, increasing cohesion, and decreasing coupling according to VMTools-RA modules instances.

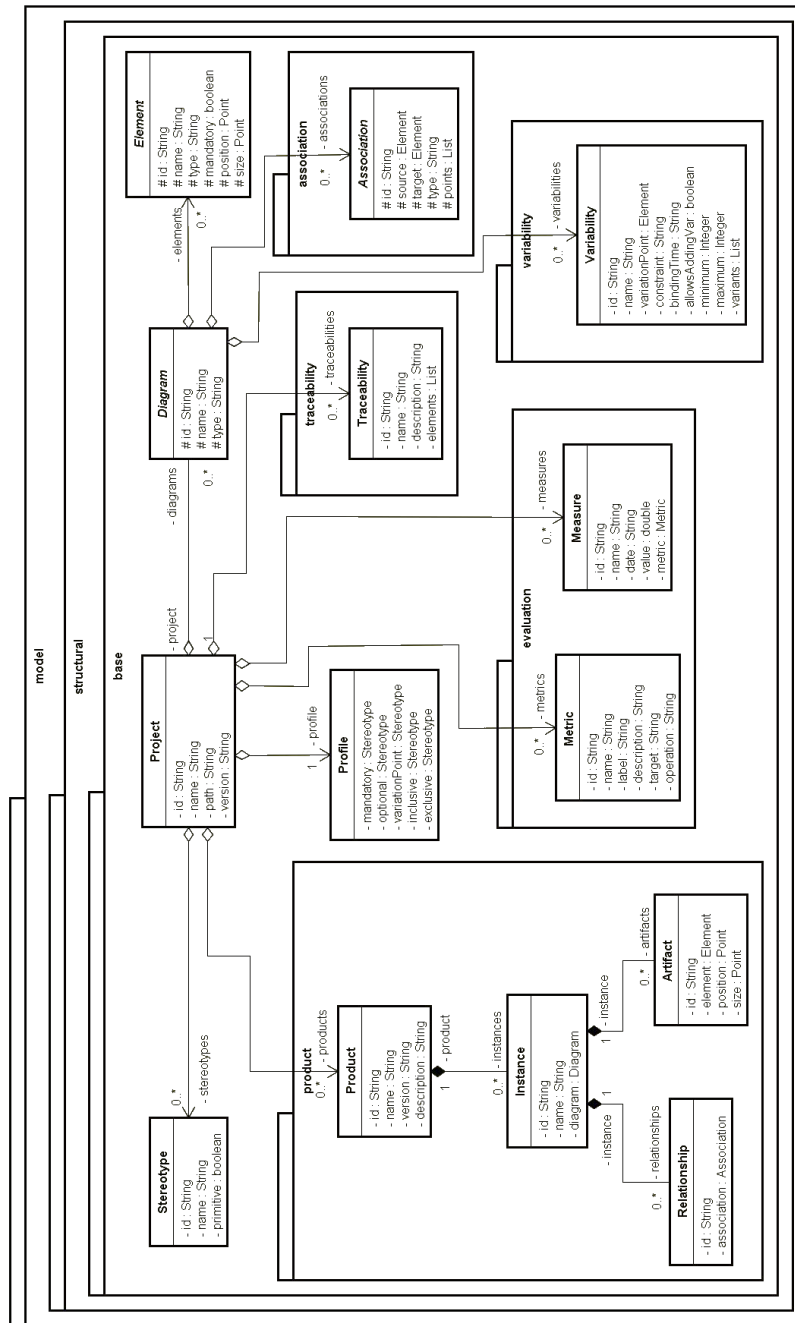


Figure 16: Classes and Interfaces of the Package model from Figure 14

The package model has as main class `Project`, which is related to several other essential classes and interfaces, such as: `Profile`, `Traceability`, `Diagram`, `Stereotype`, and `Product`.

The class `Project` contains a reference to a set of `Diagram`, which allows variability modeling in use case, class, component, sequence, and activity diagrams. Each `Diagram` is a composition of `Element`, `Association`, and `Variability`.

We defined the structure of a `Variability` taking into consideration tagged-values of a variability definition in the SMarty approach. Therefore, a `Variability` is related to a variation point and a set of variants to resolve such variation point.

The class `Project` is related to a `Profile`, which defines the role of each `Stereotype` for modeling variabilities. For instance, the user might adopt Gomaa's profile [Gomaa, 2006], Ziadi et al.'s [Ziadi et al., 2003], or the SMartyProfile [Oliveira Jr et al., 2010]. We set the latter as default.

One or more SPL-specific `Product` might be configured for a `Project`. Each `Product` is composed of `Instance`, which is a composition of `Relationship` and `Artifact`. An `Instance` class refers to a `Diagram`, thus a `Product` is composed of a set of `Diagram`.

From the VMTools-RA instantiated elements (Figure 14) and the high-level architecture of the environment (Figure 15), we established a relationship among the elements of the VMTools-RA and the architecture of the environment. Thus, we present how a described element of VMTools-RA is transformed into a solution from an architectural point of view for SMartyModeling.

The architectural elements of SMartyModeling responsible for carrying out variability management activities are present in the model package, especially the `Variability` and `Traceability` classes. The modeling panel includes the `PanelBaseVariability` and `PanelBaseVariants` view classes, responsible for defining the interface with information about the variability and its variants, and `PanelBaseTraceability` and `PanelBaseElements`, with information on traceability and its elements. In the environment, every view class has a controller class, responsible for validating and updating the attributes of the model class. Figure 16 presents the main packages and classes related to Variability Management.

For Support, versioning is restricted at the project and product level via the `version` attribute. The import and export functions are arranged in the `file` package, and the `Project` information is hierarchically organized in a standard format throughout the export method of the `Exportable` interface. The `ImportProject` class is responsible for importing the project, including reading all members, and for every diagram, the import is performed by the `ImportDiagram` class.

6.3 Running Example for Modeling Use Cases

We used the Arcade Game Maker¹² (AGM) SPL to this example. We, then, started modeling the AGM use case diagram with two actors (`Game Player` and `Game Installer`); eight use cases (`Save Score`, `Install Game`, `Exit Game`, `Uninstall Game`, `Play Selected Game`, `Play Brickles`, `Play Pong`, and `Play Bowling`) (Figure 17).

Then, we can start modeling variabilities. Optional use cases are modeled simply by selecting one of them, then unchecking the mandatory check box in the bottom left panel. The selected use case automatically changes to the annotation `<<optional>>`. For instance, use case `Save Score` is now optional (see Figure 18).

¹² <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=485941>

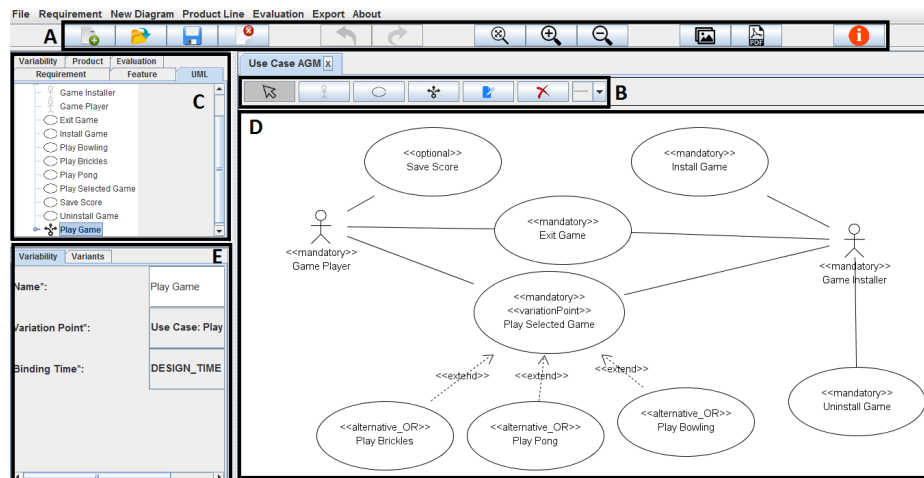


Figure 17: Modeling Variability in Use Cases with SMartyModeling

To model a variation point, we select a use case, for instance, Play Selected Game, then, select the `<<variationPoint>>` in the Stereotype window at the bottom left panel. The same is done for inclusive (`<<alternative_OR>>`) or exclusive (`alternative_XOR`) variants.

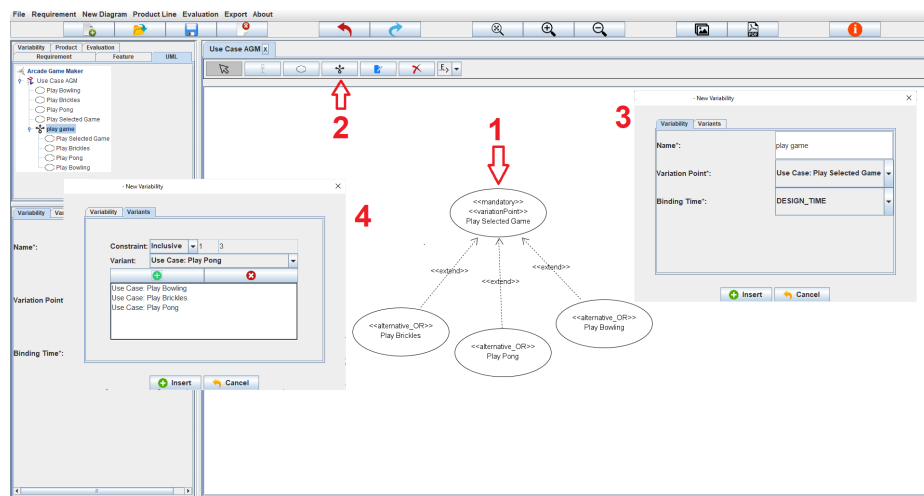


Figure 18: Creating Variation Points in Use Cases with SMartyModeling

We can use SMartyModeling to configure a product by resolving variabilities, as in Figure 19.

When saving the SMartyModeling project, one creates a “smt” file, which is an XML-format file. Tags are hierarchically organized according to the structure presented

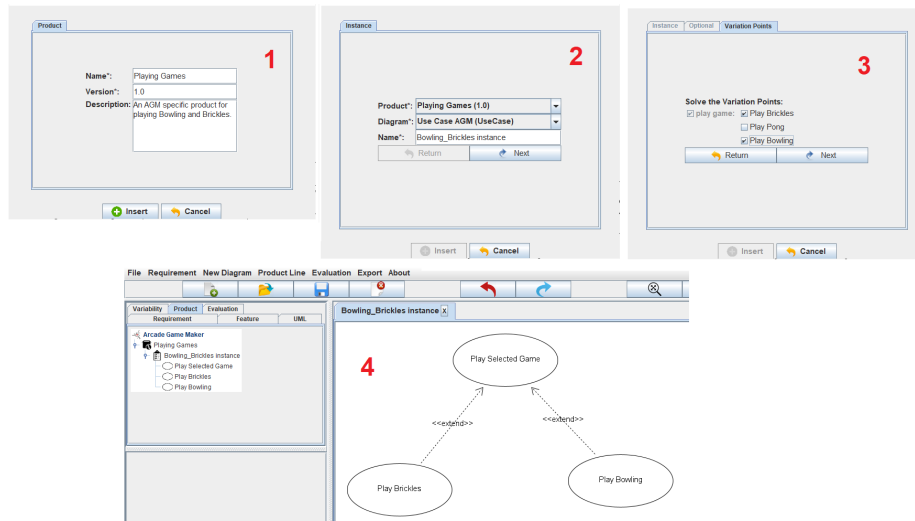


Figure 19: Configuring Products with SMartyModeling

in Figure 15. It starts with data on the project, then stereotypes, profiles, diagrams, and respective elements, relationships, variability, and products. Figure 20 depicts an excerpt of the such file created for Figure 18.

```
<diagram id="DIAGRAM#1" name="Use Case AGM" type="UseCase">
  <actor id="ACTOR#2" name="Game Installer" mandatory="true" x="790" y="180" height="50" width="75"/>
  <actor id="ACTOR#1" name="Game Player" mandatory="true" x="120" y="120" height="50" width="75"/>
  <useCase id="USECASE#7" name="Exit Game" mandatory="true" x="350" y="190" height="100" width="200"/>
  <useCase id="USECASE#5" name="Install Game" mandatory="true" x="520" y="100" height="100" width="200"/>
  <useCase id="USECASE#17" name="Play Bowling" mandatory="true" x="650" y="470" height="100" width="200"/>
  <useCase id="USECASE#15" name="Play Bricks" mandatory="true" x="80" y="440" height="100" width="200"/>
  <useCase id="USECASE#16" name="Play Pong" mandatory="true" x="370" y="540" height="100" width="200"/>
  <useCase id="USECASE#10" name="Play Selected Game" mandatory="true" x="370" y="370" height="100" width="200"/>
  <useCase id="USECASE#3" name="Save Score" mandatory="false" x="330" y="10" height="100" width="200"/>
  <useCase id="USECASE#13" name="Uninstall Game" mandatory="true" x="680" y="320" height="100" width="200"/>
  <extend id="EXTEND#18" source="USECASE#15" target="USECASE#10"/></extend>
  <extend id="EXTEND#19" source="USECASE#16" target="USECASE#10"/></extend>
  <extend id="EXTEND#20" source="USECASE#17" target="USECASE#10"/></extend>
  <realization id="REALIZATION#11" actor="ACTOR#1" useCase="USECASE#10"/></realization>
  <realization id="REALIZATION#12" actor="ACTOR#2" useCase="USECASE#10"/></realization>
  <realization id="REALIZATION#14" actor="ACTOR#2" useCase="USECASE#13"/></realization>
  <realization id="REALIZATION#4" actor="ACTOR#1" useCase="USECASE#3"/></realization>
  <realization id="REALIZATION#6" actor="ACTOR#2" useCase="USECASE#5"/></realization>
  <realization id="REALIZATION#8" actor="ACTOR#2" useCase="USECASE#7"/></realization>
  <realization id="REALIZATION#9" actor="ACTOR#1" useCase="USECASE#7"/></realization>
  <variability id="VARIABILITY#1" name="Play Game" variationPoint="USECASE#10" constraint="Inclusive" bindingTime=
    <variant id="USECASE#15"/>
    <variant id="USECASE#16"/>
    <variant id="USECASE#17"/>
  </variability>
</diagram>
```

Figure 20: SMartyModeling project data file

7 Discussion of Results and Lessons Learned

One of the challenges in this research was the lack of reference architectures for variability management tools. Two important SPL international standards, namely ISO/IEC 26550 and ISO/IEC 26555 were used. They are more abstract in comparison to reference architectures. Moreover, the functionalities and technologies used in the development of software variability tools were identified for the obtaining of variability management information. Although an SMS was performed towards gathering more data about such solutions, not enough information on the technologies used for their construction was found. Online reports from the most known proprietary solutions, namely Gears, and pure::variants, were therefore searched. They provided knowledge about APIs used for integration among tools used for specifying our VMTools-RA deployment view and information on software architecture representation, which influenced the specification of VMTools-RA as a layered architectural style. More technical details are necessary for the design of software architectures based on our reference architecture and a survey with practitioners from the industry may be a solution. However, it is outside of the scope of this research.

The identification of views that might be necessary for different stakeholders' concerns was a difficult task to be performed. The description used in FERA (*Framework for Evaluation of Reference Architectures*) [Santos et al., 2013] was followed and it guided the specification of VMTools-RA and supported us to decide which view should be presented in our reference architecture documentation. A survey with practitioners from the industry may contribute to the identification of other views that were not described in this research.

The development of SMartyModeling involved a series of decisions, starting from the understanding of the views and elements described by VMTools-RA to the planning, definition, and implementation of the environment's architecture. The complete process allows for more elaborate analysis.

Instantiating an architecture from a reference architecture is based on the principle of reusing knowledge learned from previous architectures and projects. Especially in the context of SMartyModeling, its architecture instantiation from VMTools-RA allowed us to previously understand the concepts and views on variability. Even before designing any solution, we could have a broad view of the concepts about the domain and the main activities involved, which is essential for defining the architecture of the environment.

It is also important to emphasize that the scope contemplated by VMTools-RA encompasses software variability tools in a broader context. Given this, the first decision we had was to restrict the domain of the environment architecture, to focus on the representation of variability for the SPL context. However, regardless of this restriction, the concepts and elements described mainly in terms of Variability Management were taken into account for the representation of variability in the environment, naturally adapted to the SPL domain.

The instantiation and implementation of an architecture from VMTools-RA mainly collaborate with a practical application and are part of a maturity process for the RA itself. Among the points to improve in the environment, even though it is focused on SPL, it could include more elements described by VMTools-RA. In particular, regarding the evolution of variability, we foresee a solution that encompassed a control with the information and management of the evolution of the variability during the project. Such functionality would even allow a broader view of the impact caused by changing a given variability.

The environment export module was built considering the elements described, as VMTools-RA supports such activities. It would be more complete if it allowed the persistence of packages, source code, test plans, and other artifacts, with the possibility of including integration with repositories that control changes and versioning.

8 Final Remarks

Software variability tools and reference architecture are two important research topics in the software engineering area. Reference architectures have played an important role in the development, evolution, and standardization of software systems. Variability management has been consolidated as one of the essential activities for a successful non-opportunistic reuse of software artifacts. Aiming at promoting the development of variability-intensive software systems through the support of the design of variability tool, this paper presents the reference architecture VMTools-RA. A set of architectural requirements, stakeholders, concerns, and risks identified can also be considered contributions, as they can provide knowledge for the development of other tools or the establishment of other reference architectures in the variability management context.

Results of a qualitative empirical evaluation based on checklist inspection and experts' comments enabled the improvement and design of a second version of VMTools-RA. A preliminary analysis revealed VMTools-RA supports the design of software variability tools.

Future works include: further incorporating evaluation-based changes to VMTools-RA proposed by participants; evaluation of VMTools-RA from both specification and instantiation perspectives; including new features to SMartyModeling as a way to evaluate VMTools-RA adequacy; and empirically evaluating SMartyModeling with SPL practitioners.

Acknowledgements

The authors would like to enormously thank all the participants from the State University of Maringá (UEM) and the University of São Paulo (USP) for attending the evaluation. They are also indebted to Amazon.CA and the Federal University of Bahia. Edson Oliveira Jr thanks National Council for Scientific and Technological Development CNPq (Grant # 311503/2022-5). Elisa Y. Nakagawa thanks the São Paulo Research Foundation FAPESP (Grants #2015/24144-7, #2016/05919-0, #2018/20882-1) and CNPq (Grant # 313245/2021-5).

References

- [Allian et al., 2020] Allian, A. P., Oliveira Jr, E., Capilla, R., and Nakagawa, E. Y. (2020). Have variability tools fulfilled the needs of the software industry? *J. Univ. Comp. Sci.*, 26(10):1282–1311.
- [Angelov et al., 2013] Angelov, S., Trienekens, J. J. M., and Kusters, R. J. (2013). Software reference architectures - exploring their usage and design in practice. In *7th European Conference on Software Architecture (ECSA 2013)*, pages 17–24, Montpellier, France. Springer.
- [Bashroush et al., 2017] Bashroush, R., Garba, M., Rabiser, R., Groher, I., and Botterweck, G. (2017). CASE tool support for variability management in software product lines. *ACM Computer Survey*, 50(1):14:1–14:45.

- [Bayer et al., 2004] Bayer, J., Forster, T., Ganesan, D., John, I., Knodel, J., Kolb, R., and Muthig, D. (2004). Definition of Reference Architectures based on Existing Systems. Technical Report 34, Fraunhofer Institute for Experimental Software Engineering (IESE 2004), Kaiserslautern, Germany.
- [Berger et al., 2013] Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., and Wsowski, A. (2013). A survey of variability modeling in industrial practice. In *7th VaMoS*, pages 7:1–7:8, Pisa, Italy.
- [Bezerra et al., 2021] Bezerra, C., Lima, R., and Silva, P. (2021). Dymmer 2.0: A tool for dynamic modeling and evaluation of feature model. In *Brazilian Symposium on Software Engineering*, page 121–126, New York, NY, USA. Association for Computing Machinery.
- [Bosch et al., 2015] Bosch, J., Capilla, R., and Hilliard, R. (2015). Trends in systems and software variability. *IEEE Software*, 32(3):44–51.
- [Capilla et al., 2013] Capilla, R., Bosch, J., and Kang, K. C. (2013). *Systems and Software Variability Management: Concepts, Tools and Experiences*. Springer.
- [Chen et al., 2009] Chen, L., Babar, M. A., and Ali, N. (2009). Variability management in software product lines: A systematic review. In *13th SPLC*, pages 81–90, San Francisco, California.
- [Cloutier et al., 2010] Cloutier, R. J., Muller, G., Verma, D., Nilchiani, R., Hole, E., and Bone, M. A. (2010). The concept of reference architectures. *System Engineering*, 13(1):14–27.
- [Daun et al., 2015] Daun, M., Salmon, A., Weyer, T., and Pohl, K. (2015). The impact of students’ skills and experiences on empirical results: a controlled experiment with undergraduate and graduate students. In *19th International Conference on Evaluation and Assessment in Software Engineering, EASE 2015*, pages 29:1–29:6, Nanjing, China.
- [Dobrica and Niemela, 2008] Dobrica, L. and Niemela, E. (2008). An approach to reference architecture design for different domains of embedded systems. In *SERP*, pages 287–293, Las Vegas, NV, United States.
- [Duarte, 2013] Duarte, L. S. (2013). Establishment of a reference architecture for digital television applications. Master’s thesis, University of São Paulo, São Carlos, Brazil. Master Thesis.
- [EIRA, 2018] EIRA (2018). European interoperability reference architecture (eira). URL: <https://ec.europa.eu/isa2>. Accessed in 2018.
- [Falessi et al., 2017] Falessi, D., Juristo, N., Wohlin, C., Turhan, B., Münch, J., Jedlitschka, A., and Oivo, M. (2017). Empirical software engineering experts on the use of students and professionals in experiments. *Empirical Software Engineering*.
- [Feitosa, 2013] Feitosa, D. (2013). Simus - reference architecture for service multirobotics systems. Master’s thesis, University of São Paulo, São Carlos, Brazil. Master Thesis.
- [Galster et al., 2014] Galster, M., Weyns, D., Tofan, D., Michalik, B., and Aygeriou, P. (2014). Variability in software systems - A systematic literature review. *IEEE Transactional Software Engineering*, 40(3):282–306.
- [Gomaa, 2006] Gomaa, H. (2006). *Designing software product lines with uml 2.0: From use cases to pattern-based software architectures*. Springer-Verlag.
- [Höst et al., 2000] Höst, M., Regnell, B., and Wohlin, C. (2000). Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering*, 5(3):201–214.
- [ISO/IEC26550, 2015] ISO/IEC26550 (2015). Software and systems engineering - Reference model for product line engineering and management (ISO/IEC 26550).
- [ISO/IEC26555, 2015] ISO/IEC26555 (2015). Software and systems engineering - Tools and methods for product line technical management (ISO/IEC 26555).
- [ISO/ISO42010, 2011] ISO/ISO42010 (2011). Iso/iec/ieee 42010:2011. systems and software engineering — architecture description.

- [Kang et al., 1990] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (foda): Feasibility study ;. Technical Report Technical Report CMU/SEI-90-TR-21 - ESD-90-TR-222, CMU/SEI.
- [Lisboa et al., 2010] Lisboa, L. B., Garcia, V. C., Lucrédio, D., de Almeida, E. S., de Lemos Meira, S. R., and de Mattos Fortes, R. P. (2010). A systematic review of domain analysis tools. *Information and Software Technology*, 52(1):1–13.
- [Martínez-Fernández, 2013] Martínez-Fernández, S. (2013). Towards supporting the adoption of software reference architectures: An empirically-grounded framework. In *11th IDoESE*, pages 1–8, Baltimore, Maryland USA.
- [Muller and Laar, 2008] Muller, G. and Laar, P. (2008). Right sizing reference architectures - how to provide specific guidance with limited information. In *18th INCOSE*, pages 1–8, Utrecht, Netherlands.
- [Nakagawa et al., 2007] Nakagawa, E. Y., da Silva Simão, A., Ferrari, F. C., and Maldonado, J. C. (2007). Towards a reference architecture for software testing tools. In *9th SEKE*, pages 157–162, Boston, Massachusetts.
- [Nakagawa et al., 2011] Nakagawa, E. Y., Ferrari, F. C., Sasaki, M. M. F., and Maldonado, J. C. (2011). An aspect-oriented reference architecture for software engineering environments. *Journal of Systems and Software*, 84(10):1670–1684.
- [Nakagawa et al., 2014] Nakagawa, E. Y., Guessi, M., Maldonado, J. C., Feitosa, D., and Oquendo, F. (2014). Consolidating a process for the design, representation, and evaluation of reference architectures. In *2014 IEEE/IFIP Conference on Software Architecture*, pages 143–152, Washington, DC, USA. IEEE Computer Society.
- [Oliveira and Nakagawa, 2011] Oliveira, L. B. R. and Nakagawa, E. Y. (2011). A service-oriented reference architecture for software testing tools. In *5th ECSA*, pages 405–421, Essen, Germany.
- [Oliveira Jr et al., 2010] Oliveira Jr, E., Gimenes, I. M. S., and Maldonado, J. C. (2010). Systematic management of variability in uml-based software product lines. *Journal of Universal Computer Science (JUCS)*, 16(17):2374–2393.
- [Pereira et al., 2015] Pereira, J. A., Constantino, K., and Figueiredo, E. (2015). A systematic literature review of software product line management tools. In *14th ICSR*, pages 73–89, Miami, FL, USA.
- [Pohl et al., 2005] Pohl, K., Böckle, G., and van der Linden, F. J. (2005). *Software Product Line Engineering: Foundations Principles and Techniques*, volume 26. Springer-Verlag New York Inc., Secaucus, NJ, USA.
- [Raatikainen et al., 2019] Raatikainen, M., Tiihonen, J., and Männistö, T. (2019). Software product lines and variability modeling: A tertiary study. *Journal of Systems and Software*, 149:485–510.
- [Rodriguez et al., 2015] Rodriguez, L. M. G., Ampatzoglou, A., Avgeriou, P., and Nakagawa, E. Y. (2015). A reference architecture for healthcare supportive home systems. In *28th CBMS*, pages 358–359, São Carlos, Brazil. IEEE Computer Society.
- [Salman et al., 2015] Salman, I., Misirli, A. T., and Juzgado, N. J. (2015). Are students representatives of professionals in software engineering experiments? In *37th IEEE/ACM International Conference on Software Engineering, ICSE*, pages 666–676, Florence, Italy.
- [Santos et al., 2013] Santos, J. F. M., Guessi, M., Galster, M., Feitosa, D., and Nakagawa, E. Y. (2013). A checklist for evaluation of reference architectures of embedded systems (S). In *25th SEKE*, pages 451–454, Boston, MA, USA.
- [Silva and Oliveira Jr, 2021] Silva, L. F. and Oliveira Jr, E. (2021). SMartyModeling: An environment for engineering uml-based software product lines. In *15th International Working Conference on Variability Modelling of Software-Intensive Systems*, New York, NY, USA. Association for Computing Machinery.

- [Silva et al., 2022] Silva, L. F., Oliveira Jr, E., and Santos, R. P. d. (2022). A field study on reference architectural decisions for developing a uml-based software product line tool. In *Brazilian Symposium on Software Components, Architectures, and Reuse*, page 20–29, New York, NY, USA. Association for Computing Machinery.
- [Strauss and Corbin, 1998] Strauss, A. L. and Corbin, J. M. (1998). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, second edition.
- [Wohlin et al., 2012] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., and Regnell, B. (2012). *Experimentation in Software Engineering*.
- [Wood et al., 1999] Wood, M., Daly, J., Miller, J., and Roper, M. (1999). Multi-method research: An empirical investigation of object-oriented technology. *Journal of Systems and Software*, 48(1):13–26.
- [Ziadi et al., 2003] Ziadi, T., Hélouët, L., and Jézéquel, J.-M. (2003). Towards a uml profile for software product lines. In *International Workshop on Software Product-Family Engineering*, pages 129–139. Springer.