International Journal of Advanced Research in Engineering Technology & Science
ISSN: 2349-2819
Email: editor@ijarets.org Volume-10, Issue-6 June- 2023 www.ijarets.org

FINDING PROBLEMS AND CONCERNS WITH SOFTWARE MODEL STREAMS REPLICATION

Praveer Saxena,
Research Scholar, Glocal School of Technology and Computer Science,
Glocal University, Mirzapur Pole Saharanpur (U. P.) India.

Dr. Amit Singla,
Research Supervisor, Glocal School of Technology and Computer Science,
Glocal University, Mirzapur Pole Saharanpur (U.P) India.

ABSTRACT.

Reusing common assets for multiple products reduces costs, which is one of the reasons software product lines (SPL) were introduced. Creating components that may be utilised again in other goods is frequently difficult. Software and other asset reuse presents a variety of issues, including growing complexity from the multiplicity of functions and their interactions and an increasing number of product versions. We have carried out a survey to look into how software reuse is adopted in SPL in order to provide the required level of support for engineering software product line applications and to identify some of the issues and concerns in software reuse in an effort to understand the barriers to its implementation. SPL practitioners are also asked in this study what factors affect their choice of software to repurpose inside a software product line. The survey's findings are presented in this document.

Keywords: Domain Engineering, Software Creation Line, Software Reprocess,

1. INTRODUCTION

The challenges of developing software system families, or groupings of related systems, are addressed by the study of software product lines [2, 5, 6]. By leveraging the similarities between the systems and formally controlling the differences between them, a software product line seeks to minimise the total engineering effort needed to create a group of related systems. This is a well-known issue with software reuse [8]. Reusing software has been done since the early days of computer development.

However, McIlroy's article, which suggested a software industry based on reusable components, is typically credited with establishing reuse as a separate topic of study in software engineering [1]. Software product line research has mostly focused on many challenges, including process definition, architectural modelling, domain analysis and modelling, etc. [3].

There are success stories in software product lines, according to research [9, 10, 11], but no firm guidelines for software reuse can be drawn from these tales. One of the objectives of software product lines is systematic reuse, which calls for the sharing of components throughout several 2 subprojects in addition to their mere copying. Additionally, a software product will have some components unique to it in addition to shared components for the entire software product line. This indicates that the requirements for commonality and variability determine whether software can be reused. Practitioners of Software Product Lines encounter numerous issues related to software reuse. Software reuse challenges and concerns have not yet been systematically studied, compared, and recorded.

We carried out a study to find out if domain expertise is a factor in software reuse in SPLs, to pinpoint areas that need to be considered for software reuse inside SPLs, and to ascertain whether the dependability of software assets or components is a problem for reuse in SPLs. In order to uncover problems and concerns with software reuse, this poll expands on our earlier one [4] on software reuse in the conventional software community.

The rest of the document is organised as follows: The steps, milestones, and methodology used to carry out the survey are covered in Section 2. A summary of the questions posed to researchers, managers, developers, and software engineers in SPLs is provided in Section 3. Section 4 provides a detailed description of the analysis and significant findings. The report is concluded in Section 5, where possible future research based on the survey is outlined.

2. Methodology

Midway through 2020, an email-based survey was conducted. Five distinct activities were involved in the creation, distribution, and evaluation of the survey questionnaire as well as the analysis of the findings. The following are these activities:

• **Step 1:** Gather good candidate questions;

• **Step 2:** Select questions that were of most interest;

• **Step 3:** Determine how the questions would be asked and in which order;

• **Step 4:** Select the people from SPL community whom we would like to contribute to our survey and send them the survey;

• **Step 5:** Analyze the results by putting all responses together to point out areas where the respondents identified issues and concerns.

3. Presentation

There were 51 questions in all, divided into five subsections: general type (2 questions), reuse technical issues in SPL (10 questions), reuse measurement type in SPL (16 questions), testing and the reliability of reused software in SPL (10 questions), and development environment in SPL for reuse (13 questions).

The purpose of the general type questions was to ascertain the respondents' degree of education and experience. Questions about reuse included determining the amount of reuse carried out in SPL, quantifying the amount of reuse, testing code that has been reused, the development environment that was utilised, etc. The software product line community and those involved in software product line initiatives were the survey's specified target audience.

4. Result and Analysis

4.1 Section 1: General Questions

Out of the 29 participants in the survey, 17% had fewer than five years' experience in SPL, 37% had five to ten years' experience, 37% had ten to twenty years' experience, and just 6% had more than twenty years' experience. According to the survey results, the bulk of respondents have between five and twenty years of experience. Among the responders, 65% worked as SPL researchers and managers, 20% as product line architects, 6% as core asset architects, and 9% as leaders of R&D projects. In other words, these figures may indicate that our population is more experienced in terms of working in an SPL setting. This would suggest that we have a representative sample of software product line engineers and developers, which could help us with our software reuse questions.

4.2 Section 2: Software Reuse Management and Measurement

The benefits, drawbacks, and variables affecting software product line reuse should be discussed before a company makes the decision to fully transform into a functional product line organisation. The survey's part on reuse management and assessment has collected responses regarding the benefits, drawbacks, and variables affecting reuse within the software product line community. The main advantages of reuse in SPL are well acknowledged, and the outcome of our poll confirms this. Reuse in SPL, according to 100% of respondents, will result in higher quality, planned productivity, domain knowledge capture, and cost savings. Reuse is therefore viewed as strategically significant when organisations implement SPL processes and tools, which is advantageous. According to our respondents, SPL requires efficient requirements, scope, and release management of products and services; market analysis; a shift from bespoke customer relationships and projects towards market, product, and service orientation; and effective strategic planning and product line road mapping. Furthermore, the range of reuse is expanded to include all domain artefacts rather than just software (code).

Disadvantages of Software Reuse in SPLs

On the other hand, startup and maintenance costs were the drawbacks of reuse in SPL. The "gravity" of software engineering—complexity—has been emphasised by respondents as one of the main drawbacks. Reuse can increase complexity by establishing dependencies between once independent organisational units. One of the respondents mentioned a number of issues with the dependencies, including process and technology diversity, cost of delivering integration, web of dependencies, and coordination costs.

Is Software Reuse domain based?

Software reuse "is not sufficiently domain based," according to respondents. The comments from the respondents include things like "There is no real way of translating practice into theory," "There can be widespread reusable software with inappropriate design," "People reuse software without solving architecture mismatches," and "There is too much reusable software that is large grained, such as in Service Oriented Architecture (SOA)." Other comments include "Documentation may be missing, the trace between requirements and design artefacts may be missing, and the underst Since SOA typically erodes other quality attributes like performance and scalability, developing loosely coupled software modules is challenging. Additionally, software developers are interested in developing software for reuse but are not interested in creating a product with reusable software components.

Reuse may also impede the development of original concepts and breakthroughs. Reuse and innovation should be balanced, and this should be decided by firm strategy. SPL ought to get rid of the notion that "copy and paste" is THE reuse tactic.

According to respondents, in order to preserve asset health, consistent investments should be made in order to maintain a healthy code base and a reasonable amount of variations.

The Impact of Product Line Engineering

Inquiries on how current software product line engineering practices impact reuse were also posed to the participants. The negative answers to this question ought to alter the way we now handle reuse. Reuse education, according to 87% of the respondents, will undoubtedly help them learn more about new reuse technologies that will enable reuse in their organisation. Additionally, businesses must have development rules that require reuse. Businesses that are serious about SPL need to inform all stakeholders—developers, higher level managers, and technical staff—about what SPL is, how it will be achieved, and how they will contribute to the transition's success. Nonetheless, over half of the participants (55%) believe that reusing other people's code could result in extra work. 20% of people believe that using other people's code won't make them work harder. But 26% think it will undoubtedly result in more work for them. All respondents (100%) think that reuse is becoming more recognised; yet, knowledge alone will not lead to software reuse.

Reuse Planning

According to the responders, SPL reuse requires careful planning. According to them, the key to producing helpful, reusable artefacts in the product line domain is to anticipate common and variable artefacts. To reap the rewards of SPL-based product development, the four perspectives of SPL—business, architecture, process, and organization—must be in harmony. Reuse is a component of SPL, according to the respondents, and its enablers include: high domain knowledge expertise, technical proficiency in SPL techniques, tool support, the quantity of software units produced in an SPL, driven staff, and strong management support and commitment for combining domain and technical expertise. Long-term plans and dedication are necessary. It is necessary to create standards for communicating every one of reusable components' attributes.

According to the respondents, engineering managers, quality managers, configuration managers, and/or product teams should oversee and manage the reuse policy. They should also include mechanisms for planned variation and variation control conflicts with assets, rotate product team members among various products, and/or assign them to core asset teams. According to the respondents, institutionalising SPL is extremely difficult as long as the advantages of establishing an SPL programme throughout the entire organisation are thought to outweigh the expenses. Reaching the break-even threshold between expenses and benefits as soon as feasible is crucial.

The Influence of Software Engineering Practices

Reuse is influenced by software engineering practice, according to 85% of respondents. Reuse is influenced by many different software engineering techniques, including requirements elicitation, using core assets, configuration management philosophies, and quantity and kind of documentation. Ninety percent of the participants expressed that reuse is becoming more widely recognised, with software product line success stories having an influence on businesses. It was believed that using a shared software process would encourage reuse inside a single company. While there is some benefit to having a uniform procedure across products within an SPL, there is no actual benefit to having a single process between firms.

If both the technology and the product are given equal weight, a common process will be beneficial.

A Reuse Repository

There were differing opinions on whether or not using a reuse repository would improve code reuse. There was no clear concept behind the SPL reuse repository. As long as there are appropriate procedures and guidelines around its use, a reuse repository would enhance code reuse, according to a number of respondents. Software product lines operate on the fundamental tenet that code repositories without the context that a software product line provides are inoperable.

Influence of Organization or Project Size on Reuse

Most respondents don't think that organisational reuse is correlated with a company's divisions or project size. The general consensus is that size has no bearing on reuse. This runs counter to the previously known hypothesis that states that reuse difficulty rises with project size [10]. People also believe that approaches to product lines should be customised for individual organisations. While it's not the only customisation factor, size is one of them. Although a rich (big) domain can be reused, coordination is typically more difficult. Reuse is not beneficial to a small business that creates complex, customised products. However, it is advantageous for a small business to develop products with 50–80% same components, 20–50% customised software, reuse, and SPL. Reusable software components offer a lot of potential in mass-market products. Thus, the importance of the business and target markets outweighs that of the organization's size.

Reuse and Software Quality

The majority of respondents said that reuse is not hampered by software quality. Since most of the code in each product is repeated across several products, it is already of a high calibre.

In fact, SPLs should be supported by quality considerations. However, a number of respondents believed that disputes over core assets, which typically result from divergent stakeholders' concerns about quality, can restrict the alteration and/or use of core assets within goods.

Reuse (and even modular software architecture) is discouraged, particularly for widely sold, deeply embedded systems where memory and CPU power are severely constrained owing to budgetary considerations. Reuse may also be prohibited for some real-time systems with particular performance requirements. Therefore, the biggest barriers to reuse are performance in terms of efficiency and response time. It goes without saying that every domain artefact that is intended for reuse (usually in an SPL environment) needs to undergo thorough testing to identify any defects, both functional and non-functional. When reuse is not consciously and strategically planned for, quality issues can significantly impede reuse.

Domain Knowledge and Software Reuse

It has been demonstrated that domain expertise is still essential for software reuse [10]. Regarding this question, all of our respondents were in agreement. It is impossible to predict common and variable artefacts without domain knowledge. Actually, domain knowledge—rather than code—is what is reused in SPL. However, if the reuse infrastructure sufficiently supports the production process, then not every application engineer needs to possess that knowledge. According to a number of respondents, subject expertise is important yet insufficient. Businesses that have effectively implemented an SPL strategy typically highlight their existing strengths and minimise their difficulties. It has been observed that SPL efforts produced by architects, process engineers, and managers are eventually required. Domain expertise is important, but not the primary component. More important are

management commitment and process discipline to follow through on a reuse agenda motivated by business objectives.

4.3 Section 3: Software Reuse Technical Aspects in SPL

According to a number of respondents, in order to encourage reuse, software engineers should modify the programming language they use. Classes and components, for example, are important elements of reuse. High-level programming languages facilitate the creation and use of reusable applications. There are differing views, nevertheless, on how crucial the programming language selection is. Math libraries built in FORTRAN were the first significant reuse store.

The abundance of research on reuse CASE tools and the expanding market for them demonstrate how many organisations view CASE tools as a means of enhancing reuse. Participants were asked if they agreed with the statement, "CASE tools have promoted reuse across projects in their organisation," in order to gather data for this study. According to the research, half of the respondents typically believe that CASE tools have not encouraged reuse across projects within their company, whilst the other half concur. The future is in model-driven environments and tools, not so much in classic CASE tools.

Reuse takes place at the model level in Model-Driven Development (MDD) and Model-Driven Architecture (MDA). Design artefacts can be reused with the use of CASE tools. Software attributes can be better understood by reverse engineering code with the use of tools that support round trip reengineering, allowing for reuse. We come to the conclusion that reuse is not currently being greatly aided by CASE tools. Of the respondents, half stated that the SPL community uses CASE tools, and the other half indicated they don't. Reuse is encouraged, nevertheless, by strategies like SOA and various product line tools [10].

Respondents who were asked if they would rather construct something from scratch or reuse something already existing felt that in an organisation that manages a product line, reuse was a given and was not a choice.

It is obviously problematic to reuse materials arbitrarily that have not been prepared and explicitly designed to be reused for a certain purpose, as we have learned from ad hoc reuse experiences. The consensus among the respondents is that they would rather reuse existing software than create it from scratch if it matches the design.

Thus, the primary factor that makes reuse easier is the architecture. Reuse can be accomplished efficiently if we know (1) which domain artefacts are available and (2) we can trust the domain artefacts to do the things we want them to do, so why not reuse them? was one very intriguing comment that was received. The question then becomes, "Do we know these issues, and how can we know them?" The SPL community has been presented with an extremely difficult question.

All the participants are of the opinion that domain knowledge is the key to reuse of software. One cannot decide what software to build as reusable software without understanding the domain and identifying common functionality/features that can be developed as reusable software. According to our respondents the main advantages of the SPL approach are:

• The product line's wide engineering vision can be shared among the projects easily

• Development knowledge and corporate expertise can be utilized efficiently across projects, and

• Assets can be managed systematically.

The SPL approach often requires a large upfront capital investment to create an organizational structure dedicated to implementing a reuse program and it takes time to see a return on investment [9].

4.4 Section 4: Testing and the Reliability of Reused Software

Despite the community's enthusiasm, it is almost impossible to discover the exact component in a library of components that would address the current problem. There are two aspects to the issue. First of all, the library of reusable components does not contain the ideal part. Second, is the component dependable enough to be used even if we locate the ideal one? A software architecture that is documented can help with understanding a system and demonstrate how the dependability of a system depends on the dependability of its interfaces and components. The software architecture defines the behaviour of the programme with regard to how its various modules interact with one another.

We looked into a few survey responses when looking into this matter.

In addition to being asked to score how much they agreed with the statement "Software developed elsewhere is reliable," participants were also asked if they tested any components before reusing them. The results of our study indicate a substantial association between these variables, indicating a close relationship between the amount of external reuse and quality problems. Only 40% of the respondents said they tested the code they reused, despite the fact that nearly 63% of them said their code contained some aspect of reuse. Typically, more time is spent testing features unique to a product. Furthermore, issues that are found are typically associated with reusable parts that were improperly modified for a particular product.

Respondents stated that the system design aids in the comprehension of the aforementioned characteristics when asked if they "understand system and component reliability, their interactions and the process to identify critical components." Numerous studies have been conducted that detail some of the standard practices (as well as client demands) in specific industries, such as the automotive sector. FMEA (Failure Mode and Effects Analysis), FTA (Fault Tolerance Analysis), and the soon-to-be ISO WD 26262 are some examples. Occasionally, scenario-based techniques such as the Architecture Tradeoff Analysis Method (ATAM) [5] have been employed to pinpoint

crucial system components. The majority of SPL teams use expertise to determine which components are essential. The primary sources of information on the crucial components are the source code, documentation, and real component testing. Understanding system and component reliability as well as the interconnections between components lacks a systematic or formal method. Component reliability is based on how any given component contributes to the defect reports, which are derived from (internal or field) defect reports. Systems and component interactions are analysed via informal architectural reviews. The majority of poll participants said that an architecture review is the best way to locate trustworthy components.

Respondents stated that single classes are usually not evaluated when asked if they "test a single component, class, or core component in any way." The majority of respondents stated that they don't test individual parts on their own. While tests are occasionally conducted on individual components, integrated components are tested the majority of the time after some degree of integration with other components has been completed. An assemblage of components may occasionally be the subject of unit testing, which is the most basic type of test "unit." There isn't much use of unit testing. 10% of the components, according to one of the answers, have some kind of unit test. SPL mostly uses JUnit, Window Tester, httpUnit, and Rational Test Manage as its test frameworks. Software is tested using a conventional, structured methodology that includes both unit and integration testing. Unit testing concentrates on a component's internals.

Integration testing is done on how components are used together. Both the model and the code levels undergo the same testing. Additionally, some respondents stated that they don't test components since they have faith in the Eclipse process. Some of the respondents believe that since a component is always created based on the specifications of a product, it needs to be tested to make sure it satisfies those specifications.

Reusing software components may also change how much time is allotted during the phases of software development. Time spent in the analysis and design phases can be reduced if a component is already reusable. Test cases that are correctly documented and up to date with the most recent version of the main item can be reused. Our data indicates that the analytical phase takes 5–20% of the time, the design phase takes 5–60%, the implementation phase takes 0–50%, and the testing phase takes 10–30% of the time. A few participants stated that they dedicate 50% of their time to testing and 0% to implementation.

4.5 Section 5: Development Environments for Reuse

We asked survey respondents what kind of development environment they typically work in. Most respondents said they preferred Netbeans, Java Eclipse, or a basic IDE for programming, such JCreator. A few participants employ a model-based methodology, utilising UML2 as the mainstream modelling language, supplemented by tools and models for ontology-oriented design. Eclipse is a widely used tools platform these days.

For embedded devices and microcontrollers, a specialised environment is utilised, such as Eclipse for Java projects, and Visual Studio is usually used for PC-based programmes. A project designer said that the programming language and environment vary depending on the type of Programmable Logic Controller (PLC) being used, since each PLC brand has a unique environment for the software designed for that PLC.

We obtained a mixed reaction when we asked if they thought the choice of framework (e.g.,.NET, EJB, CORBA, etc.) affected the software's potential to be easily upgraded over time. A single participant expressed unfamiliarity with the frameworks, while others expressed disagreement, and the remaining group thought that the major frameworks would likely be easier to update than the others. The majority of respondents also concurred that a component's complexity had no bearing on whether it should be developed or reused. No matter how sophisticated a component is, it gets used if it has been tested before. One of our respondents did, however, think that domain expertise and complexity might influence these kinds of choices, although this truly depended on the business plan.

Regarding the requirement for a thoroughly documented software architecture of the system in order to reuse code, we got extremely encouraging feedback. Software/system architectures with thorough documentation are essential for facilitating reuse decisions and properly integrating various components to shorten testing times.

Few responses, meanwhile, think that architectural features are not necessary as long as domain understanding is sufficient. Specialists are adept at retaining domain knowledge. Reuse benefits greatly from architecture, albeit this may not always be the case.

This raises the issue of what would happen if experts left a company. With legacy systems, there is a significant unresolved problem where expertise have departed the organisation and the documentation is out of date [7]. The majority of software product lines, according to all of our respondents, need to be updated and maintained over time. The largest issues facing SPL, according to core asset architect respondents, are integrating core assets in application engineering and maintaining and upgrading core assets.

5. CONCLUSION AND FUTURE WORK

The problems and worries regarding software reuse in SPL were acquired for this article through a community survey of SPL participants. Among the most important conclusions is that there are other keys to reuse in SPL besides domain knowledge. Even if there are a lot of reusable resources in the SPL, the product line architecture needs to come first. Since SPL has a clearly defined reuse procedure, having a properly documented architecture is not necessary for reuse. Nonetheless, the architecture should be required to be documented through an SPL process. The problem with reuse is that sufficient traceability of the artefacts within and across interacting domain and application requirements engineering life-cycle requires 10 to be maintained. The variability should be

specified in domain requirements artefacts. Furthermore, understanding system and component reliability as well as the relationships between components lacks a systematic or formal method. Software architecture could be useful in determining essential elements. Some of the most significant practical needs in software product lines are not addressed by the tools and methods that are currently in use.

We believe this paper will be of interest to software product line practitioners as well as those organizations struggling with reuse as they will gain the following from the paper:

• Software architecture can be used in a product line setting to support better identification, reuse, and integration of reliable components. Documentation of software architecture requires a substantial amount of effort.

• The list of issues and concerns given in this paper can be used to implement a new software reuse process in SPL.

• Current product engineering and derivation processes in SPL can be analyzed for systematic reuse of software in the SPL.

A Knowledge Base Software Reuse System (KBSRS) is necessary to support today's software product lines due to their potential for size and complicated diversity; otherwise, a methodical approach to software reuse might not be achievable. The creation of a KBSRS, which will be utilised to record organisational knowledge in software reuse, is a component of our next effort. Similar surveys were previously carried out to identify problems and worries around software reuse within the traditional software engineering community. In order to compare the two communities and find lessons that both communities can benefit from, we will be comparing the results of this poll with those of the prior survey.

REFERENCES

1. McIlroy D., “Mass Produced Software Components”, *Software Engineering Concepts and Techniques: Proceedings of the NATO Conferences,* Buxton, J M, Naur, P., and Randall, B. eds., Petrocelli/Charter, 1969, Pages: 88-98.

2. Hoffman D. M., and Weiss D. M., *Software Fundamentals: Collected Papers by David Parnas*, Addison-Wesley 2001.

3. Torkar R., and Mankefors S., “A Survey on Testing and Reuse”, *Proceedings of the IEEE International Conference on Software-Science, Technology & Engineering*, 0-7695-2047-2/03, 2003.

4. Jha M., O'Brien L., and Maheshwari P., "Identify Issues and Concerns in Software Reuse", Proceedings of *Second International Conference on Information Processing* (ICIP'08), Bangalore, India, 2008.

5. Clements P. and Northrop L., *Software Product Lines: Practices and Patterns.* Addison Wesley: Boston, MA, 2002.

6. Birk A., Heller G., et al., "Product Line Engineering: the State of the Practice ", *IEEE Software,* Vol. 20(6): Pages: 52-60, 2003.

7. Jha M. and Maheshwari P., "Reusing Code for Modernization of Legacy Systems", *IEEE International Workshop on Software Technology and Engineering Practice (STEP),* September 24-25 Budapest, Hungary, 2005.

8. Kruger C.W. "Software Reuse", *ACM Computing Surveys,* Vol. 24 No. 02 June, Pages 131-183, 1992.

9. Weiss D. M. and Lai C. R., *Software Product Line Engineering: A Family- Based Software Development Process.* Addison-Wesley, 1999.

10. Pohl K., Bockle G., and Linden F. v.d., *Software Product Line Engineering: Foundations, Principles, and Techniques,* 1st ed. New York, NY:Springer, 2005.

11. Linden F. v.d, "Software Product Families in Europe: The ESAPS & CAFÉ Projects", *IEEE Software,* Vol. 13, No. 3, Pages: 41-49, 2002.