An Approach for Enabling Effective and Systematic Software Reuse

In a Globally Distributed Software Engineering Team that uses a Lean Development Methodology

Roopa M. S., V. S. Mani
Siemens Technology and Services Pvt. Ltd.
Bangalore, India
roopa.ms@siemens.com, vs.mani@siemens.com

Stefan Halwas
Siemens AG
Nuernberg, Germany
stefan.halwas@siemens.com

***Abstract*—we share our experience in pursuing effective software reuse in a globally distributed software engineering team that uses a lean development methodology. The paper outlines the journey, starting from recognizing the potential for reuse, the steps taken to enable systematic reuse in lean projects, the challenges faced, and the corrective actions taken to ensure effectiveness of systematic reuse. The main lessons learned include: i) identification of relevant domains for reuse, ii) explicitly assigning responsibilities for reuse component development, iii) providing enabling infrastructure, iv) defining more rigorous software development processes for reuse components, and v) establishing a centralized team for developing reuse components. The results of our successful reuse initiative including the significant increase in quality and a 12 percent reuse of total code developed have been presented.**

Keywords—software reuse, global software engineering, lean development methodology

I. INTRODUCTION

We share our experience in pursuing effective software reuse in a globally distributed software engineering team consisting of about 1,000 engineers distributed across locations in North America, Europe, and Asia.

II. BACKGROUND

Our organization develops products in a highly regulated industry. We are involved in developing hardware, software, and firmware for multiple product families. Due to the nature of the industry, these products have long lifecycles and are expected to perform as specified for several decades. In addition, the products must fulfill regulatory requirements that entail conforming to industry standards. Hence, for our products, quality is non-negotiable.

To ensure on-time delivery of high-quality software, the team had switched to lean software development methodology [1,2]. The team had been operating for over 10 years and had addressed the issues encountered in global software development [3,4].

III. INITIAL APPROACH FOR SOFTWARE REUSE

The potential of software reuse within our organization was immediately recognized when we started software development. At that time, reuse was achieved by merely copying source parts from one project into another without any governing process. We soon observed a rapid increase in maintenance effort and growing incompatibility between products. We realized that we required a systematic approach to organize reuse [5]. Subsequently, we explored ways to harness the potential of software reuse and proposed an approach to the management with the following objectives: 1) reducing development time and costs by reusing existing components, 2) ensuring components with the same functionality have identical behavior and look and feel, and 3) increasing quality by systematic reuse of software components.

Our approach focused on four main areas: A) relevant domains for reuse, B) organizing for reuse, C) funding, and ensuring reuse, and D) enabling infrastructure and processes to facilitate reuse.

A. Relevant domains for reuse

In view of the differences of handling reuse in the different technical areas, we identified relevant reuse domains that were aligned with the development technologies and product platforms. These are listed in Table I. Our approach to identifying domains shares the objectives of domain analysis [6] while being pragmatic without the rigor of formalism.

TABLE I. RELEVANT REUSE DOMAINS

	Reuse domains	***Focus topics***
1	PC software	PC platform based software components
2	Test and diagnostics	Test tools, test scenarios, and test strategies
3	Embedded software	Software components for embedded platforms
4	Third-party software	Handling, tracking, and organizing third-party software
5	Hardware	Hardware circuit blocks
6	Documentation	Technical documentaton for different products
7	Open source software	Handling, tracking, and organizing open source software

2329-6313/16 $31.00 © 2016 IEEE
DOI 10.1109/ICGSE.2016.14

B. Organizing for reuse

For each reuse domain, we identified members of the different project teams, who were assigned dual roles – one for project related tasks and another for reuse component related tasks. We also defined two new roles: domain owner and component owner. Domain owners are typically architects or subject matter experts who are expected to drive reuse in their domain. For their domain, they are responsible for: 1) assessing proposals for both new reuse components and significant enhancements in existing components, 2) defining a reuse roadmap, and 3) the architecture of the reuse components. On the other hand, component owners are responsible for: 1) design, 2) implementation, 3) testing, 4) release, and 5) maintenance of the reuse component (see Fig. 1).

C. Funding, and ensuring reuse

All activities were funded by the projects on a need basis (see Fig. 1). Further, to ensure support for reuse, the architect team was given a target to reduce costs explicitly through reuse of software. In addition, it was an unwritten understanding that the entire development organization would support the domain owners and the component owners in achieving the goals of reuse.

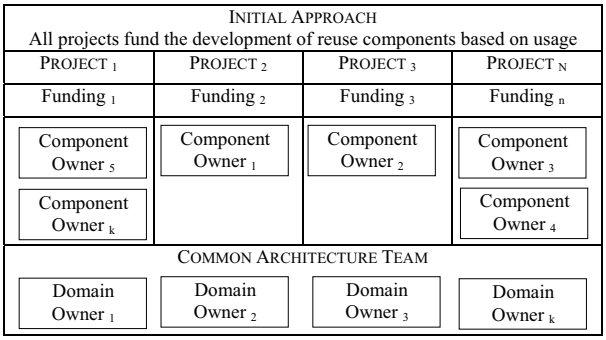

Fig. 1. Organizing for reuse in the initial approach

D. Enabling infrastructure and processes to facilitate reuse

To support systematic software reuse, we invested in enabling infrastructure: 1) configuration management and defect tracking of the reuse components, and 2) web portal to share information related to reuse components, and to highlight the benefits of the reuse initiative.

1) Configuration management, release management, and defect tracking: We aligned the configuration management, release management, and defect tracking of the reuse components with the respective systems in the projects. This allowed different projects to include the reuse components like any other project component. In addition, defect handling was adjusted for reuse purposes, so that all projects could report and track the defects in reuse components.

2) Web portal: The web portal contained information about the different reuse components, their features, documentation links, release versions, and their component owners. Thereby, all information required to decide whether or not to reuse a component was readily available. The portal reported the usage of reuse components. It also computed the savings realized. Savings were computed as *(number of projects-1) × (half the total development cost of the component)*. This computation was based on analysis of past data, which showed that the overhead costs associated with reuse were about half the development costs. See Table II.

TABLE II. COMPUTATION OF SAVINGS

No. of projects reusing component	*Savings (units)*
1	0
2	500
3	1,000
4	1,500
Total cost of development of reuse component = 1,000 units	

IV. EXPERIENCES AND ISSUES WITH THE INITIAL APPROACH

The initial approach was successful. We were able to identify about 200 reuse components. Moreover, about 100 of these were internally developed and used successfully. In addition, team members across the organization actively contributed to the software reuse initiative.

However, over time, we encountered a variety of issues due to which project and product managers began to lose trust in the reuse initiative. These are detailed below and the quantitative data is presented in Table III.

A. Prioritization

As component owners were a part of a project team, they used to assign higher priority to project-specific tasks due to pressures of project deadlines, and lower priority to the development tasks for reuse components. This impacted the ability of the component owners to deliver reuse components as scheduled to the other projects, resulting in far too many escalations and unpleasant discussions.

B. Feature bias

The component owners also demonstrated a bias towards features that were needed by their projects and by critical large projects. This had a double negative impact. One, the smaller projects did not get the functionality they needed in the reuse components. Two, the reuse components bloated as more and more project-specific functionality got included in them.

C. Quality

Since the component owners were not provided with a standardized design and test strategy for the delivery of the reuse components, no consistent design and test strategy was followed. This impacted the quality of the reuse components, especially with respect to non-functional requirements like maintainability, scalability, and the other 'ilities'. The quality issues became evident during enhancement and maintenance of the components. It was also reflected in the over 200 unresolved defects and the considerable time, about 70 percent, spent in resolving defects.

D. Communication

As there was no single point contact for multiple components, project managers had to interact with multiple component owners, which made it difficult to track the progress in the different components. Furthermore, all reuse issues had to be escalated to the management.

E. Funding

Typically, product managers were unwilling to bear the additional costs associated with the reuse of software components, such as the efforts associated with increased communication and conflict management coupled with the costs associated with more rigorous design, testing, and documentation.

V. Analyzing the Issues

We analyzed the issues in the initial approach with the aim of identifying steps to address them. The root causes for the issues were:

A. Dual Responsibility

Inability of component owners in scheduling workload while playing a dual role of being project team member and at the same time designing, developing, testing, maintaining, and enhancing a reuse component. Though the budgeted effort for tasks related to reuse components was adequate, the challenge lay in scheduling these tasks along with the typically unpredictable daily project pressures.

B. Enforcement of processes

Component owners were not enforced to follow defined standard processes while developing reuse components. Further, processes for the specific requirements of reuse components had not been defined. Hence, component owners were able to progress through the entire development lifecycle of reuse components without adequately involving other specialist teams such as architecture, usability, and test; which resulted in lower quality.

C. Single point of contact

Lack of a single point of contact for all reuse components made it difficult to resolve issues such as the delayed delivery of features, enhancements, and defect fixes. These coupled with quality problems of reuse components led to project managers questioning the value of the reuse initiative.

D. Reluctance in funding reused code base

Though product managers were willing to harvest the benefits of reuse, they were reluctant to take on the additional costs related to development and maintenance of reuse components within their existing budgets.

VI. Changed Approach

To counter the causes of the issues faced we made several changes to the initial approach. The issues we encountered have been reported in reuse literature even in co-located teams [7,8,9]. However, we observed that the issues were amplified due to the globally distributed nature of our team. We also realized that it was difficult to directly lift and drop any of the suggested practices to resolve our issues [10]. Instead, in the spirit of lean, we always sought an optimized solution to our specific issues with the overarching goal of reducing waste.

However, the original goals of the software reuse initiative remained unchanged. The changes were in the following areas: 1) Creation of a centralized reuse project, 2) Funding, 3) Processes for reuse component development.

A. Creation of a centraized reuse project

A separate reuse project was created with a dedicated reuse project manager accountable for all reuse components. The reuse project manager serves as a single point of contact for all project teams. This enabled issues related to reuse components to be resolved effectively.

The reuse project manager is supported by a dedicated software engineering team that oversees the design, implementation, testing, enhancement, and maintenance of reuse components. Most component owners are now part of the reuse project, which helped us in avoiding the problems due to dual responsibility. However, a few component owners are still members of project teams. Domain owners continued to be part of the common architecture team (see Fig. 2).

B. Funding

The funding of a reuse project was changed with the aim of shifting the extra costs and risks of reuse production and consumption out of individual projects [2]. The development (design, implementation, and test) of a reusable component is still financed by the project in which the component was originally planned. However, the additional efforts (extensive testing, documentation, build and release management, and management of information on the web portal) are covered by the reuse project. In addition, the reuse project finances future-oriented enhancements such as web-enabling all user interfaces, and minimizing dependence on third-party software components (see Fig. 2).

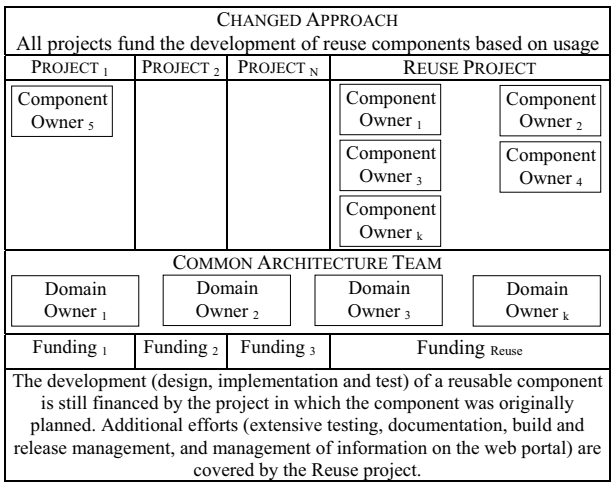

Fig. 2. Organizing for reuse in the changed approach

C. Processes for reuse component development

As part of the new approach, we introduced rigorous processes for reuse component development with an emphasis on improving quality [8]. The changes were in the areas of: 1) enabling reuse in lean software development methodology, 2) code quality and testing, 3) defect tracking, and 4) release management.

1) Enabling reuse in teams using lean software development methodology: Since detailed planning for features for products was done just before the implementation started, features related to reuse components came up very late in the project phase, which was a challenge. To overcome this we realized that estimation for features related to reuse components had to be completed at least one 'takt' (a takt is similar to a sprint in agile) ahead of scheduled work. This also enabled having the dedicated reuse project team fully available for reuse tasks in a lean way.

2) Code quality and testing: To ensure code quality, we introduced mandatory static code analysis with pre-specified goals. In addition, we made the testing process more rigorous by focusing on the potential impact of the changes to the code, mandating increased code coverage, and achieving it through test automation. We also developed smoke tests for project environments to quickly detect obvious defects. It was possible to enforce the changed processes because reuse was a separate project with its own funding.

3) Defect tracking: In the initial approach, after a defect was resolved by the component owner, there was no mandate for the project team to close the defect in the defect tracking system. Hence, the defect tracking system had too many defects in an improper state, which led to repetitive and wasteful discussions. To address this, we modified the defect tracking process. Now, defects in reuse components have to be reported in the project and cloned to the reuse component. This meant that unless the reuse defect is closed the project defect cannot be closed. This ensured timely closure of all resolved reuse component defects and enabled more effective monitoring of the unresolved defects.

4) More rigorous release management process: We introduced a checklist to confirm that the mandated steps prior to creating a release label are fulfilled. These include mandatory reviews, static code analysis, test coverage, and documentation checks. The checklist is reviewed by a team from the reuse project comprising quality manager, test manager, and component owner. Since the reuse project is separately managed, it was possible to enforce this process for all reuse components.

VII. RESULTS AND IMAPCT

The changed approach resulted in a significant shift in the culture of the organization with respect to reuse. This helped the reuse team in increasing quality, reducing development time, and reducing overall cost.

A. An evolving culture of reuse

By culture of reuse we mean the organization-wide belief that reuse is beneficial, and the resulting changes in the way the organization performed its tasks.

With the new approach, the *build-for-reuse* and *build-with-reuse* mindsets were firmly established in the organization [9].

1) Build-for-reuse: By build-for-reuse we mean the reuse team focused on generic solutions while deliberately using robust architecture and design practices combined with stringent code quality checks and reliance on test automation for more intensive testing.

2) Build-with-reuse: By build-with-reuse we mean the projects systematically explored the possibility of reuse. During the product definition phase, the architects in our organization perform a detailed architectural review where in addition to general architectural questions, the suitability of existing reuse components is evaluated, new requirements for the existing components are identified, and new reuse components are determined. In addition, the impact of specific enhancements on other products is assessed.

3) Handling conflicts in requirements of reuse components: Since we follow a lean development methodology, requirements are often identified during the 'takt' realization. In such cases, the project manager sends requirements to the component owner who in turn synchronizes the enhancement with the respective reuse architect and product architects. If a requirement is in conflict with the interest of any product, the reuse team tries to find a common solution by involving domain experts and usability experts. If a common solution is not found, the product team implements the feature independently.

4) Unbiased prioritization: Reuse requirements are prioritized based on the following factors. 1) Project release criticality, which is typically determined by the need to sustain market share or requirements of strategic customers. 2) Project release timeline, which entails giving preference to features required by projects that need to be released sooner. 3) Risk, if the impact due to inclusion of a feature is high, then the risk to the project is assessed to determine if the feature should be deferred to a future release.

5) Feature assignment: Features are assigned to the reuse team based on the following factors. 1) Cost-effectiveness, which entails assessing if it is more cost-effective to implement a feature in the project or in a reuse component. 2) Resource availability, if the resources are not available to implement the feature in the reuse team, the feature is implemented in the project. If it makes business sense, the feature is subsequently moved into a reuse component.

6) Communication: One of the challenges faced by a globally distributed organization is informing all stakeholders of the existing and planned features in the reuse components. In addition to the web portal, we send a monthly newsletter to the entire organization, which contains details of reuse components released in the current month and details of

releases planned in the next month. We also made it a practice to email the release notes for every component release to all individuals who had subscribed on the web portal for that component.

7) Collaboration: Reuse team members and project team members work together on many occasions such as 1) clarifying ambiguous requirements, 2) examining hard to reproduce defects, 3) determining the cause of defects that cannot be readily isolated, 4) understanding domain-intensive use cases of the products that demand greater domain expertise.

8) Flexibility: Since we follow a lean development methodology, it is essential to accommodate changes in requirements and their scope. Consequently, planning for reuse features and enhancements needs to be very flexible to achieve the desired lean goals. We achieve this flexibility by explicitly allocating effort for unplanned requirements and high-priority defects.

B. Achieveing quality, delivery, and cost goals

The cultural changes coupled with the more rigorous processes helped the reuse team in making improvements in several areas such as: 1) quality, 2) on-time delivery, and 3) increase in number of reuse components.

The quantitative data of the improvements are presented in Table III. The sources of this data are the component repository, configuration management system, and defect tracking system.

1) Improvement in quality: With increased focus on rigorous design and testing, there has been a significant increase in the quality which is reflected in reductions in: the number of reported defects, number of unresolved defects, and time spent on resolving defects. In addition, there has been an increase in time spent on enhancements.

2) On-time delivery: In the changed approach, project teams were expected consider dependencies during the planning phase, which is essential to ensure that the reuse team is able to deliver the component in time for the required takt. Thereby, we were able to adhere to committed schedules and ensure timely delivery of priority enhancements, critical defects, while resolving and reporting issues more effectively. It is noteworthy that we have had zero escalations in the last two years.

3) Increase in number of reuse components: The share of reuse components completely moved to the reuse project has increased nearly five-fold in five years, while at the same time the reuse project budget decreased by 33 percent. (See table III). It is important to note that in addition to developing new components, the reuse team is actively enhancing existing reuse components, within this lower budget.

TABLE III. IMPROVEMENTS IN QUALITY AND PRODCUTIVITY

	Initial (2010)	***Now (2016)***
Total amount of code reused	Not measured	12%
Average number of unresolved defects in a year	200	30
Time spent on resolving defects	70%	30%
Time spent on enhancements	30%	70%
Escalations	Too many to count	0
Share of reuse components handled in Reuse project	8%	38%
Reuse budget	-	33% less
* Percetage of all software reuse components		

VIII. CONCLUSIONS

Despite our organization having experience in global software engineering, we encountered issues while trying to enable effective software reuse within our teams, which were similar to what has been reported in earlier studies of reuse. However, we observed that the issues were amplified due the globally distributed nature of our development teams. We were able to successfully resolve the issues through systematic corrective actions that were aligned to our lean methodology and specific to our situation.

IX. REFERENCES

[1] Lean Transformation: How Lean helped to achieve Quality, Cost and Schedule, Uma V., IEEE 9th International Conference on Global Software Engineering, 2014

[2] Successfully transforming to Lean by changing the mindset in a global product development team, U. Samatha, V. S. Mani, IEEE 10th International Conference on Global Software Engineering, 2015

[3] TAPER: a generic framework for establishing an offshore development center, G. Hoefner, V. S. Mani, 2nd IEEE International Conference on Global Software Engineering, 2007

[4] 4 C: An approach for effective people management in an offshore software development center, G. Hoefner, V. S. Mani, 7th IEEE International Conference on Global Software Engineering, 2012

[5] Systematic Software Reuse: Architecture, Process and Organization are Crucial, Martin L. Griss, (http://martin.griss.com/pubs/fusion1.htm)

[6] Domain Analysis: An Introduction, R. Prieto-Diaz, ACM SIGSOFT. Software Engineering Notes Vol 15 No 2. Apt 1990 Page 47

[7] Incentive compatibility and systematic software reuse, Robert G. Fichman, Chris F. Kemerer, Journal of Systems and Software, New York; Apr 27, 2001; Vol. 57, Iss. 1; pg. 45

[8] Why software reuse has failed and how to make it work for you, Douglas C. Schmidt (http://www1.cse.wustl.edu/~schmidt/reuse-lessons.html)

[9] Experiences in Software Evolution and Reuse: Twelve Real World Projects (Research Reports Esprit), 2013 Editors Svein Hallsteinsen, Maddali Paci

[10] Why new software processes are not adopted, Stan Rifkin, Advances in Computers, vol. 59, 2003