Transitioning from plan-driven to lean in a global software engineering organization: a practice-centric view

Roopa M. S., Ratnanabh Kumar, V. S. Mani
Siemens Healthcare Pvt. Ltd. 84, Keonics Electronics City
Bangalore, India
roopa.ms@siemens-healthineers.com, ratnanabh.kumar@siemens-healthineers.com, vs.mani@siemens-healthineers.com

ABSTRACT

We share the experience of a globally distributed software development organization in transitioning from a plan-driven approach to a lean methodology with a focus on role-specific practices. We outline how the new practices supported effective working in a global setup by strengthening trust, increasing communication, fostering openness, and encouraging faster decisions.

CCS CONCEPTS

Software and its engineering → Software creation and management → Software development process management → Software development methods; Agile Software Development

KEYWORDS

Global software engineering, lean, agile, software engineering practices

1 INTRODUCTION

This paper shares experiences from a large globally distributed software development organization where software, firmware, and hardware are developed. The projects typically had about 200 team members distributed globally in Europe, the U.S. and India. We develop software products that are an integral part of medical devices used for imaging and diagnostics. Typically, these products have a lifecycle spanning 10-20 years. As the products serve the healthcare market they need to conform to regulatory requirements [1].

Despite long product lifecycles we need to continually add new functionality to meet customer expectations and competitive pressures. As the new functionality is increasingly realized through software, it results in frequent changes to the code.

ICGSE '18, May 27–29, 2018, Gothenburg, Sweden 2018
Copyright is held by the owner/author(s).
ACM ISBN 978-1-4503-5717-3/18/05.
https://doi.org/10.1145/3196369.3196395

Over the years the organization saw that plan-driven methodologies were ineffective in handling the increasing complexity of systems and delivering new features quickly. After exploring new ways of working, the organization decided to transition to lean methodology for software engineering [2,3,4]. Though we use the Scrum methodology, we follow lean principles [5]. We see lean being different from agile in the way it focuses on creating more value for customers while also minimizing waste continually. Lean software development achieves these objectives by rigorously focusing on identifying customer needs to define the minimum viable product (MVP), which is the smallest set of features you can build that delivers customer value. To minimize waste, lean relies on steadily completing tasks, which maximizes flow, through intensified communication and better prioritization realized through visual management. Waste is also minimized by empowering teams and team members to call out problems early without fear. Lean drives continuous improvement through the practice of retrospectives where team members share what worked and what did not.

The project organization is outlined in section 2. Practices that were followed by the central teams are outlined in section 3. The practices within the Scrum teams are outlined in section 4. The conclusions are presented in section 5.

2 OUTLINE OF PROJECT ORGANIZATION

Since most requirements demand expertise in areas such as user interface, business logic, database and firmware; we formed cross-functional Scrum teams with expertise in each of these areas. As the expertise was distributed across regions, it resulted in the cross-functional Scrum teams being globally distributed.

The benefits of cross-functional scrum teams include: i) reducing the need for synchronization meetings and related delays between the hardware, firmware, and software ii) planning the iterations effectively, ii) facilitating exploring optimizing the solution across all modules rather than locally optimized solutions.

To effectively coordinate the multiple Scrum teams, we setup a central organization consisting of: 1) Central Planning team (CPT), 2) Product Owner team (POT), and 3) Architectural Governance Board (AGB). Team members and responsibilities of the different teams are outlined in **Table 1**, while **Figure 1** reflects the regional spread.

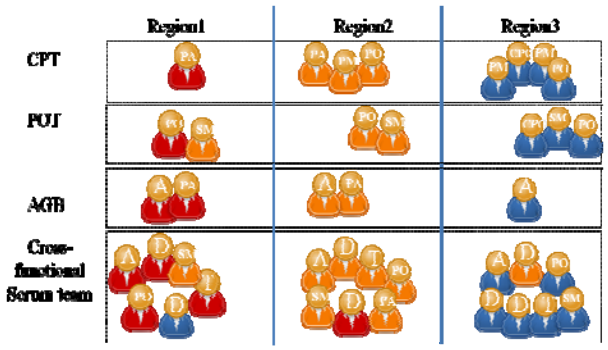

Figure. 1: Project organization

Table 1: Project Organization

Team	Members	Responsibility
•**Central Planning Team (CPT)**[1]	•Product managers •Chief product owner •Product architects •Key experts (e.g. usability, domain, test)	•Define scope of project (minimum viable product-MVP) •Define initial plan and budgetary estimate for MVP •Coaching scrum teams
Product Owner Team (POT)[1]	•Product managers •Chief product owner •Product owners •Product architects •Scrum masters	•Govern conformance to quality policy and regulations •Elaborate requirements •Rank user stories and track their fulfillment •Identify and enable resolution of dependencies and conflicts between scrum teams •Decisions on re-ranking and integration of requirements
Architectural Governance Board[1] **(AGB)**	•Product architect •Scrum team architect	•Govern conformance to architecture and design •Reduce technical debt
Cross-functional scrum team[2]	•Scrum master •Team architect •Developer engineers •Test engineers •Shared key experts in usability, configuration engineering, domain topics	•Plan iterations •Detail vertical user stories •Design, develop, and test functionality to fulfill user stories •Ensure conformance to quality policy
Central functions[3]	•Configuration engineer •Usability experts •Quality engineers	•Manage infrastructure for integration •Monitor conformance to quality policy and regulations •Provide expertise on demand

1. Single team for entire project, distributed globally
2. Multiple teams in a project, distributed globally
3. Single team for each region

3 PRACTICES OF THE CENTRAL TEAM AND THEIR BENEFITS

Two-step planning using backlogs

Planning was done in two steps: 1) a high-level plan of the project, the requirements, the resources, team composition, and budget, 2) a detailed just-in-time iteration plan.

High-level planning. All requirements are compiled in a single product backlog for the all modules of the product. The Central Planning Team (CPT) assesses the return on investment and customer value of each requirement in the single product backlog to define a release backlog. This approach significantly reduced the time for high-level planning from around six weeks to two weeks.

Just-in-time iteration-level planning. This step is completed before the start of an iteration. The POT assesses the feasibility of integrating high-priority requirements in the release backlog for the iteration backlog. Based on the impact of a requirement on different modules, a requirement is assigned to one or more Scrum teams. POT also creates user stories for these requirements.

The Scrum teams estimate the number of user stories that can be completed in the iteration. Since the estimation is done incrementally, the team can learn from previous iterations. As the trust between the distributed teams increased, the need for formal estimation reduced and was eventually done away with, which significantly empowered teams.

Planning ensures basic architecture and design is available but detailing is done while progressing with implementation. Since certain issues became clearer as we progress in project, design is more accurate, and developers have greater say in low-level design.

Co-locating to build trust across regions

As part of the transition to lean, entire teams from different locations travelled to headquarters and worked there together for two iterations. This helped the team members understand each other better. It also provided the opportunity for the entire team to grasp domain specific issues and constraints. It also helped the teams better comprehend and be conscious of the effort needed to make what appeared to be small changes. The central team coached the different scrum teams to be more open and supported them in sharing views freely. In addition, the teams had basic intercultural and soft skills trainings to facilitate better listening and appreciating different views. These measures helped the individual team members quickly respect each other and foster trust.

Co-locating component expertise

Though the globally distributed cross-functional scrum teams worked well, the large number of interactions across time zones resulted in more than expected overheads. To address this, we

came up with an approach to equip co-located teams to complete workflows independently. As this involved transfer of component ownership, it required buy-in from affected stakeholders.

The Scrum team defined a knowledge transfer plan with theory sessions followed by practical sessions using reverse shadowing and mentoring. This enabled teams to deliver faster due to reduction in waiting time. However, in several cases, the transfer of knowhow and working independently took a couple of years.

Knowhow sharing session across Scrum teams.

We introduced a monthly knowhow sharing session across all Scrum teams. In these meetings, new techniques, such as new test approaches, new tools or coding infrastructure are shared. Teams share why some approaches worked and why some did not, which helps them deploy best practices across scrum teams.

Architectural practices.

The Architecture Governance Board (AGB) governs practices to reduce technical debt across multiple scrum teams.

In addition, the AGB reviews the design and implementation to identify architectural guideline violations such as interaction between classes, public interfaces, persistent and non-persistent data. The AGB also decides which functionality will be developed as a reusable component.

The practice of assessing the risk and safety criticality of a requirement is continued from the plan-driven world, and is done by the product architect and the product manager.

4 ROLE WISE PRACTICES OF THE SCRUM TEAMS AND THEIR BENEFITS

4.1 SCRUM MASTER

The Scrum master is responsible for iteration level planning and enabling the Scrum team in achieving the iteration goals by coordinating, collaborating and resolving team impediments.

As the Scrum team aims to add value, fixing defects and meeting quality goals are prioritized over implementing requirements. If the team feels a prioritized feature is not adding value proportional to the effort required they request the product owner to re-rank the requirement.

Detailed user stories

After the requirements for an iteration are assigned, the Scrum team elaborates them into detailed user stories with the help of usability experts. The Scrum master ensures detailed user stories (smallest working features) are vertical in nature covering different layers in the system architecture (see Figure. 2). This aids in: 1) better understanding of functional and non-functional requirements, 2) early identification of unanticipated dependencies, 3) enabling rapid integration, and 4) getting regular feedback from product owners, product managers, architects and testers. Overall, this practice had a significant contribution in releasing usable software at the end of each iteration. Previously, without this practice, we found a significant number of requirements were getting integrated late in the project, which resulted in a late detection of defects.

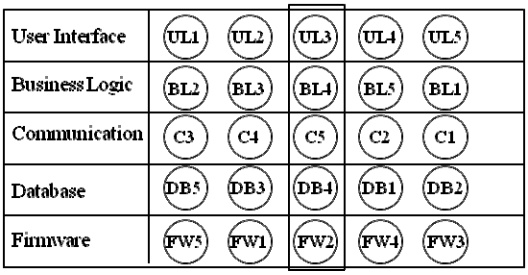

Figure 2: Vertically-sliced smaller user story covers different layers in system architecture

Intensifying communication

We started a practice of daily Scrum of Scrum meetings to identify and remove impediments such as dependencies across Scrum teams, early in an iteration. These were scheduled and conducted virtually as the team was globally distributed. As the project progressed, the number of topics to be discussed reduced these meetings were held weekly.

To achieve flow, a Scrum team needs to: 1) rank the backlog, 2) create vertically-sliced smaller user stories, 3) identify dependencies early, and 4) resolve issues. Achieving this requires intensified communication, for which we initiated several practices over and above the ritual of the daily stand up meeting.

The daily standup meeting is practiced by all teams including those that are not co-located. During the stand up meeting, team members are encouraged to spell out issues blocking progress of the iteration. The Scrum master encourages team members to resolve these issues by themselves, or if requested by the team members facilitate support from other agencies to resolve the issue. Over time, team members observed that issues were getting addressed without challenging individual capability and became more open. To further aid communication, a single team has an email distribution list to share relevant information too long to share in a standup meeting.

Coaching and feedback

The Scrum master uses proven coaching techniques like GROW [6] to enable team members uncover approaches to complete tasks.

The Scrum master facilitates a monthly review of team performance at a location based on results of the previous iteration. This ensures feedback to the team and individual team members is more objective and immediate. Hence, it is easier to have constructive conversations, especially on individual performance. Since Scrum masters are not directly responsible for appraisal of team members there is greater acceptance of the feedback, even when it is negative. This further increases openness within the team.

Iteration retrospectives, which are different from monthly reviews, are not limited to a location and include all team members. The Scrum master ensures that retrospectives only allow discussion on performance of the team as a whole, and not about specific individuals to facilitate openly sharing perspectives and feedback.

We use a visual board for the retrospectives that has four quadrants: 1) what went well, 2) what did not go well including blockers, 3) risks, and 4) wish list. The visual board enables team members to openly share their impressions of an iteration. To address identified deficiencies, action items are defined and included in the work tasks for the next iteration. At the same time, to foster working as a team and focusing on meeting team goals rather than individual targets, team achievements are recognized, rewarded and celebrated.

4.2 TEAM ARCHITECTS

The team architect is responsible for ensuring that within the Scrum team: 1) technical debt is managed effectively, 2) design of the implemented functionality is appropriate and satisfies all non-functional requirements, 3) the software conforms to regulatory and quality guidelines, and 4) knowhow is shared to enable continuous improvement.

To address the technical debt that was getting accumulated, we began a practice of resolving the identified technical debt within the iteration itself. For example, if legacy code violates the quality guidelines, it has to be modified within the iteration.

In some cases, a proposed design change impacts many components, and potentially delays the product release. For these situations, the Architectural Governance Board analyzes the impact of new features on the existing design and implementation. The AGB along with POT decides if the change can be postponed to a later version of the product.

We introduced a practice of analyzing the nightly build reports of unit tests, static code analysis. The team architect informs the team of any failures and deficiencies. Since the team uses a single code base, such an analysis enables early detection of code quality deficiencies, which reduces the cost of fixing such defects. From an architectural perspective, the nightly build analysis ensures that no new defects are introduced.

4.3 DEVELOPERS

Developers are responsible for pulling tasks from the iteration backlog which also include automation. The implemented features should conform to the quality guidelines. Developers are supported by our code quality management program [7]. To avoid delays in release due to defects, developers follow several practices that include:

1. Strict code check-in conditions. All non-legacy code should completely satisfy the code quality guidelines. For example i) all unit tests must have passed, ii) zero open defects in the module. Only if these conditions are met the code can be checked-in to the integration branch.

2. Bug Jail. If the total number of quality issues including defects and unmet NFRs exceed a specified limit, then the entire team stops all new feature development and focuses on resolving open issues.

3. Stop the line. If any team member sees issues in achieving the iteration goal, they can request the Scrum master to stop the line.

4. Workflow-centric unit tests. To create these test cases the developers need to understand the requirements better, which is facilitated by the test engineers through user-centric acceptance criteria.

5. Peer testing. Developers test features implemented by a peer developer in the scrum team. They apply their knowledge to create white-box and grey-box tests that effectively identify difficult to detect defects such as event leaks.

4.4 TEST ENGINEERS

Test engineers who are subject matter experts, are responsible for enhancing use cases and writing tests to validate that the software satisfies the acceptance criteria.Test engineers also automate integration tests and NFR tests, which are executed on the nightly build to detect integration and stability issues early. The practice resulted in stable builds, which let POT evaluate intermediate versions early. This enabled team to release usable software at end of each iteration.

Knowhow sharing within scrum teams

We started a practice of weekly knowhow sharing sessions, which are driven by the team architect. In these sessions:

1. Developers present their implementation approach for currently pulled tasks and seek feedback from the entire team. This allows the collective wisdom and prior experience to improve the implementation and testing approach. These interactions also enable more effective reviews as the team now knows the nuances upfront, and reduces rework. The practice is effective even in globally distributed teams.

2. Test engineers provide more insights about the end user workflows and scenarios for different user stories. This helps writing effective integration and system tests. Due to the closer interaction with the developers, test engineers were able to better understand the code implementing the features to create error-detecting grey box tests, which complemented the peer testing practice of the developers.

The sessions also helped new team members quickly learn what does not work. Instead of merely informing the team about different design, coding guidelines, and quality policies; the real project examples in these sessions helped the team understand the guidelines better. This helped reduce rework and improve performance. In addition, the formal review effort with experts reduced by over 50%.

4.5 CONFIGURATION ENGINEERS

Configuration engineers follow a practice of having two branches: the iteration branch and the integration (main) branch. To ensure code does not break other functionality the iteration

branch tests were run to ensure that delays due to any failure were local. It saved time in a globally distributed team.

Configuration engineers share the list of changes in the checked-in source code with the teams. The team architect uses this information to identify the impacted functionality, which needs to be tested. Team architects from other scrum teams, test engineers, and domain experts comment on the impacted functionality and NFRs to effectively identify high-priority test cases.

5. ROLE-BASED DEVELOPMENT PROGRAM

We identify role-based development programs for the team members [8]. Scrum masters and product owners also receive coaching. Teams are trained on the entire tool chain required to monitor code-quality and security. Individual development programs are monitored to ensure team members have the required capabilities to be effective in their roles. In addition, awareness sessions on our principles [5] are conducted periodically for all team members, which help teams in living them.

6. LESSONS LEARNED

Some of the key lessons learned are:

1. Vertically-sliced detailed user stories helped a lot in releasing usable software at the end of each iteration. Previously, without this practice, we found many requirements were getting integrated late in the project, which resulted in delayed releases.

2. Automated build tests that were run at night resulted in stable builds, which let POT evaluate intermediate versions sooner.

3. Closing defects in the same iteration increased the impact and contribution of test engineers in meeting the iteration goals, defects got resolved quickly and with less friction.

4. The lean approach of working helped strengthen the test engineer-developer partnership, resulting in better knowledge within the team and increased productivity.

5. Coaching and ongoing feedback are critical to enable teams in becoming self directed, which is essential for the lean way of working.

6. The awareness sessions around our principles were highly appreciated by all team members and considerably helped in living our principles. These sessions also strengthened the use of value stream mapping, agile visual management, and the practice of coaching peers.

7. CONCLUSIONS

The practices mentioned in this paper have proved to be effective in enabling high performance software engineering. There is increased trust across teams, which enabled cross-functional scrum teams in different regions to take on more global roles. The project organization helped reduce time for planning, and effectively resolved conflicts and dependency issues.

Involving test engineers across the lifecycle of a requirement, while sharing the same team goal helped strengthen their partnership with developers.

Project teams achieved shorter release cycles, while ensuring high quality reflected in a significantly lower number of field defects.

REFERENCES

[1] AAMI TIR45: Technical Information Report Guidance on the use of AGILE practices in the development of medical device software, 2012.

[2] . Agile Transition at Siemens Healthcare https://www.slideshare.net/AgileAndrea/agile-transition-at-siemens-healthcare-syngo-xp2012-presentation

[3] M. Poppendieck, T. Poppendieck Lean Software Development – An Agile Toolkit for Software Development Managers, 0-321-15078-3, Addison-Wesley, Boston (2003).

[4] M. Poppendieck and M. A. Cusumano, "Lean Software Development: A Tutorial," in IEEE Software, vol. 29, no. 5, pp. 26-32, Sept.-Oct. 2012.

[5] . Principles of Healthineers (can be accessed at https://www.healthcare.siemens.de/careers#).

[6] Alexander, Graham. "Behavioural coaching—the GROW model". In Passmore, Jonathan. Excellence in coaching: the industry guide (2nd ed.). London; Philadelphia: Kogan Page. pp. 83–93, 2010.

[7] H. K. Singh, U. Uppili and U. Pavuluri, "Making Continual Code Quality Monitoring and Control Processes Work in a Global Delivery Organization: COSMOS," 2013 IEEE 8th International Conference on Global Software Engineering, Bari, 2013, pp. 173-177.

[8] G. Samarthyam, G. Suryanarayana, A. K. Gupta and R. Nambiar, "FOCUS: An adaptation of a SWEBOK-based curriculum for industry requirements," 2012 34th International Conference on Software Engineering (ICSE), Zurich, 2012