Towards Agile Yet Regulatory-Compliant Development of Medical Software

Vlad Stirbu
CompliancePal
Tampere, Finland
vlad.stirbu@compliancepal.eu

Tommi Mikkonen
University of Helsinki
Helsinki, Finland
tommi.mikkonen@helsinki.fi

***Abstract*—As digital transformation is taking place in more and more industries, the role of software increases and the skills required to develop software trigger a ripple effect. Entire industries, where regulations and government standards play an important role, like health care, have used long development cycles that relied on detailed planning. Agile software development proved that it can deliver results that satisfy customers needs faster than traditional waterfall methodologies. The apparent conflict between fast delivery cycles and lack of detailed planning have lead to situations where the use of agile became synonymous with lack of documentation and poor quality. In this paper we propose new approaches that demonstrate that appropriate use of agile practices brings benefits to compliance activities too.**

1. Introduction

Digital transformation is a phenomenon in which digital technology is integrated in all areas of a business. The result is that a business engaged on this path becomes a software business, which fundamentally changes how it operates and delivers values to its customers. The transformation is typically accompanied by a cultural change that requires the organization to challenge status quo through experimentation and getting familiar with failure.

Digital transformation poses a significant challenge for businesses operating in heavy regulated environments such as health care. The agility enabled by modern software development methodologies, open source software, and the increasing use of cloud infrastructure capabilities is at an apparent conflict with the practices that are used to implement the regulatory frameworks, optimized for the old world.

This paper explores new practices and cultural shifts that enable organizations developing medical device software to reconcile the high velocity of agile methodologies with the mandatory regulations in the field. The background of the paper is two-fold. On one hand it is based on review of regulations and earlier work partially reported in [1], and in the other hand, the paper contains realizations from practical software engineering in industrial setting.

The paper is structured as follows. Section 2 introduces modern software developments and practices that enable high velocity delivery of features. Section 3 provides an overview of regulatory landscape relevant for medical device software development. Section 4 describes the proposals, followed by the experimental results in Section 5. Concluding remarks are presented in Section 6.

2. Modern software development practices

In this section we explore agile software development and development operations (DevOps), the two main practices that enable software development organizations to deliver functionalities for their customers at high speed.

Agile software development [2] is a lightweight approach of developing software, where the requirements and the solution evolve through collaboration between the development team and the beneficiary of the solution. The development team is typically cross-functional and self-organizing. The development team starts with an initial plan and design that evolves through a rapid cycle of releases and continuous improvement. The approach promotes flexible response to change over strictly following plans, and it has been sometimes misinterpreted as a series of ad-hoc decisions rather than a disciplined engineering methodology. In this paper, we build on the disciplined interpretation of agile software development.

Scrum [3] is one of the most commonly used agile framework for software development. The methodology defines two special roles within the development team: the *product owner* that represents the voice of the stakeholders and customers, and the *scrum master* that facilitates the scrum and is responsible with removing impediments that can hamper the ability of the team to deliver on their goals. The development team works in small time-boxed increments called *sprints* and takes work items from an ordered list of product requirements called *product backlog*. Each sprint is bounded by a *planning* and *review* session. During the sprint, the team holds *daily scrum* sessions to evaluate progress and identify impediments.

DevOps [4] combines practices and tools with cultural philosophies that increases the ability of an organization to deliver applications at high velocity. The DevOps practices and culture are aligned with and complement agile software development practices by integrating, testing and deploying applications at a rapid pace. For example, monitoring in real-time the behavior of the applications in production

and acting if not performing within the desired quality parameters, creates a fast feedback loop.

The capabilities of modern infrastructures exposed via application programming interfaces (APIs) are leveraged into a set of tools that allow a high level of automation throughout the life-cycle of the application. The use of software development practices for handling the infrastructure, such as version control, allows the changes to be handled in a standard and controlled way, resulting in repeatable and consistent deployments that can be easily rolled back.

Agile software development and DevOps create an environment in which *experimentation* can thrive [5]. By having the mundane tasks of testing, integrating, packaging and deploying automated, the teams can focus their attention on bringing new features to the customers faster. Anyone in the team can perform these operations as needed without required additional expertise.

3. Medical device software regulations

The regulations covering medical device software fall into two broad categories: information handling and safety. The regulations related to information handling are Health Insurance Portability and Accountability Act (HIPAA) [7] in Unites States and General Data Protection Regulation (GDPR) [6] in the European Union. HIPAA is a medical sector regulation that defines what constitutes *protected health information*, its use and disclosure when several health providers are involved in the care process. GDPR is a generic data privacy framework that defines how personal information is collected and used. To comply with HIPAA and GDPR regulatory frameworks, a service provider that implements part of the functionality using software must establish procedures for handling the relevant information. These procedures typically get materialized into technical requirements, which have to be implemented in software, or standard operating procedures, which have to be followed by the stuff that interacts with the protected information. The regulations extend to business associates that process or handle protected information, which have to comply themselves.

The safety of medical devices or services is regulated in Unites States by the Food and Drug Administration (FDA) and Medical Device Directive in European Union. International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) have developed harmonized standards that contain guidance on the processes and requirements that must be followed when developing software so that relevant regulatory authorities accept medical products in the respective markets.

For example, ISO 13485 [9] specifies the requirements for a quality management system that allows an organization to demonstrate its ability to provide medical devices and related services that consistently meet customer and applicable regulatory requirements. Further, ISO 14971 [10] specifies the processes that a manufacturer has to follow to identify the hazards associated with medical devices, to estimate and evaluate the associated risks, to control these risks, and to monitor the effectiveness of these controls. Lastly, IEC 62304 [8] defines the life cycle requirements that must be followed by an organization where software is embedded or is an integral part of the final medical device. The requirements are envisioned as a set of processes, activities and tasks that establish a common framework for medical device software life cycle processes.

4. Medical device software goes agile

Although the processes, activities and tasks defined in the IEC 62304 are specific in what they have to accomplish, the standard is not opinionated on how they are actually implemented. This situation leaves a high degree of freedom for affected organizations to select the software development methodologies best suited for their needs, such as waterfall, agile, or hybrid. The implementation is relatively straight forward for traditional waterfall software development as the upfront detailed planning can incorporate all the steps required. The implementation is more complicated for modern software development practices that incorporate agile and DevOps methodologies. There, the objective is on delivering customer features at high velocity while maintaining the organization ability to react to changes at reduced costs. As the plans are detailed and revised regularly based on the learnings found during the implementation and releases to the customers, there is an increased possibility for rifts to appear between the compliance activities and the actual realities of the software implementation.

We propose an approach in which the risk management, requirements traceability and verification is embedded in the development team and becomes integral part of the software development process. Their needs become essentially first class needs, and appropriate tools must be developed for handling compliance tasks.

4.1. Risk management

IEC 62304 defines three threat classes that convey information about the level of harm that the recipient of the medical device that includes software can experience if used according to the intended use specified by the manufacturer: A does not result in any injuries, B can lead to non-serious injury and C can lead to serious injury or death.

To evaluate in which category the software fits, the product is decomposed into *software systems* that are composed of integrated collections of *software items*. The items can be decomposed further into *software units*. The manufacturer decides on granularity of items and units. The risk classification is by default inherited from the parent component and can be changed by performing risk analysis at component level. If following the risk analysis a child component is classified with a higher risk than its parent, the classification of the parent components are elevated recursively up to the software system.

In agile teams that do not have dedicated compliance expertise, the risk classification steps are performed traditionally by compliance consultants before a major product

release or a certification process starts. The risk analysis is conducted on a snapshot of the code base, a situation that leads to a feature freeze as no new functionality can be merged till the assessment is completed. This leads to friction with the development team.

To mitigate these situations, we propose an approach in which the risk assessment is performed continuously as the software is developed. The development team should include compliance expertise and the risk assessments are performed in small increments as soon as new risks are identified. To avoid overloading the compliance aware team members, the process is facilitated by DevOps tooling that identifies common risks (a process further detailed in Section 4.3). The tools are typically run on every code commit and deviations are brought to the attention of responsible parties.

Another aspect emergent from having the development team and compliance separated is that the two teams use separate tooling which reduces the visibility of the compliance activities to the development team. We propose to handle the risk management files using the same requirements management tool used by the development team, each file being a assigned a corresponding issue. The approach has further benefits as the risks can be linked with the risks controls implemented in software from the product backlog, which aids traceability.

4.2. Traceability and verification

Traceability refers to the ability of tracking requirements implementation as they are decomposed into product requirements, system requirements, and finally into software requirements. Verification refers to the ability to verify the actual implementation state of the mentioned requirements.

JIRA[1], a tool commonly used to manage requirements, has the ability to manage links between requirements. Similarly, Jenkins[2] has the ability to identify corresponding JIRA issues with jobs results. Together with Git, these tools commonly used by organizations practicing agile software development, are the enablers for realizing traceability and verification. However, although these tools can be connected together via web-based application programming interfaces (APIs), setting them for this purpose is not straightforward nor obvious, particularly for users not familiar with the regulatory frameworks.

The situation can be improved with specialized tools that configure the issue trackers, source version control and continuous integration and deployment to implement standardized workflows for traceability and verification. When DevOps practices are in use, the source version control covers also the software configuration management and infrastructure changes, either in public clouds or on premises.

4.3. Compliance checks on code commits

During typical software development, the number of code changes merged into the common code base is several orders of magnitude larger than the number of software requirements. This situation makes harder for the compliance designated personnel to verify that no new risks have been introduced during the software development.

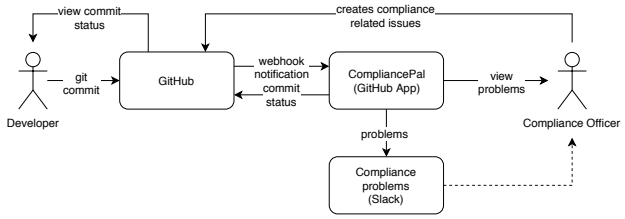

Figure 1. Experimental environment workflows.

One common risk is that developers use 3rd party software that is not developed according to IEC 62304 practices, known in this context as *software of unknown provenance* (SOUP). These kind of risks can be mitigated by having compliance checks performed automatically on each code commit or on merge requests. If new risks are identified, corresponding issues are opened automatically and the incidents are investigated as soon as possible by relevant people. Performing as many checks as possible immediately reduces the friction between the developers and compliance managers, eliminating the need to perform compliance investigations before major milestones can be achieved; all deviations and corrective actions are performed close to the time they occur.

5. Experimental results

Bridging the medical device software development and agile practices is a long term endeavor that should be accomplished, in truly agile spirit, in a sequence of small increments. For this reason, among the three points described in the previous section, we have decided to explore how the compliance checks on code commits is a valuable practice that enhances the visibility of compliance among the development team, and enable a feedback loop that allows the compliance officers to perform risk analysis as potential issues occur.

The experimental setup, depicted in Figure 1, mirrors the daily routine of an agile development team practicing the scrum methodology. Among the agile team we emphasize two roles relevant for the experiment: the developer and the compliance officer. The team works on a project that develops software using JavaScript programing language. The code produced by the team is managed using a Git repository hosted on GitHub[3]. GitHub built-in issues functionality is used for requirements tracking. The compliance checks are performed by our service that extends the standard GitHub workflows using the *Apps*[4] integration methodology. Possible compliance problems are brought to the attention of the team via dedicated chat room hosted in Slack[5].

1. https://www.atlassian.com/software/jira
2. https://jenkins.io
3. https://github.com
4. https://developer.github.com/apps/
5. https://slack.com

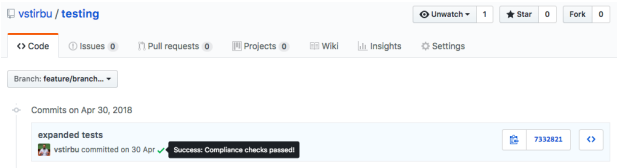

Figure 2. Check progress and state viewed in GitHub user interface.

The software developer workflow consists of developing new functionality, test it locally, then commit changes to the local git repository. Following the team practices to merge completed new functionality to the common code base, the developer pushes the changes to the remote GitHub shared repository. GitHub notifies its service integrations upon receiving new changes. Our service receives the change notification and performs code analysis to identify new SOUP components in the new change set. The service updates the progress of the check using the GitHub Statuses[6], an API that allows third parties to convey progress information via a harmonized *state* (e.g. error, failure, pending, or success), a short *description* of the status, and a *web link* where further information can be obtained. The developer is able to follow the progress using the familiar GitHub user interface. For example, Figure 2 contains the depiction of a successful SOUP check.

If the compliance checks detects new SOUP components, the service updated the status on GitHub to *failure* and post a new notification in the compliance problems channel hosted in Slack. The designated compliance officer receives the notification and starts the investigation by inspecting the detailed view of the problem presented by the CompliancePal service. For example, in Figure 3 we can observe the result of the analysis presented in two sections. The first section represents as a UML package diagram that contains highlighted packages that correspond to the newly identified SOUP components. The second section guides the compliance officers with the most common actions that can be taken to address the identified problems. The compliance officer may decide to open a GitHub issue and assigns the issue to the relevant GitHub project board from within the problem detailed page. Once the identified problems have issues, the compliance officers can proceed with conducting risk management activities according to agreed team practices.

6. Conclusions

The sample workflow presented in the experiment proves that properly designed tooling can bring together agile software development and compliance practices. The seamless integration into development team tools reduces friction, enabling the developers to focus on software development activities. The use of familiar DevOps patterns gives them the confidence that compliance problems are detected and handled by the responsible team members fast. Similarly,

6. https://developer.github.com/v3/repos/statuses/

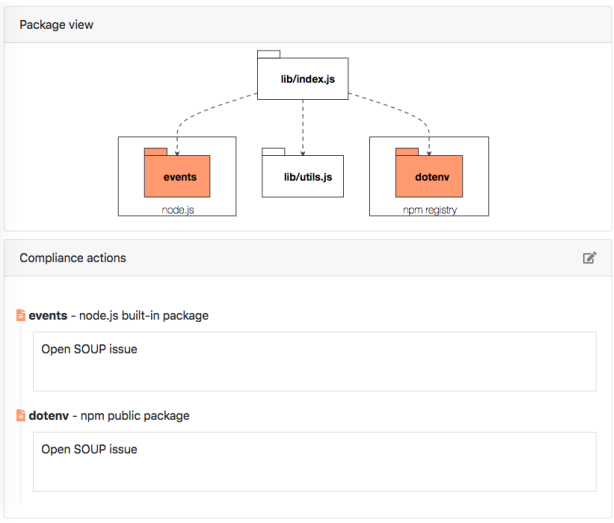

Figure 3. Detailed view of SOUP compliance check.

compliance officers are confident that they have up to date information about the software implementation in their area of interest. The high level of automation and transparency builds trust within the team. The tooling assists the team members to perform to compliance activities only when needed, making them efficient while maintaining high velocity. As a result, compliance becomes an organization's shared goal instead of an impediment.

References

[1] Laukkarinen, T., Kuusinen, K., and Mikkonen, T. DevOps in regulated software development: case medical devices. In Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track, pp. 15-18. IEEE Press, 2017.

[2] Cockburn, A. *Agile Software Development* Addison-Wesley, Boston, 2002.

[3] Schwaber, K., and Beedle, M. *Agile software development with Scrum*. Vol. 1. Upper Saddle River: Prentice Hall, 2002.

[4] Debois, P. DevOps: A software revolution in the making. Journal of Information Technology Management 24, pages 3-39, no. 8, 2011.

[5] Fagerholm, F., Guinea, A.S., Mäenpää, H., and Münch., J. Building blocks for continuous experimentation. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, pp. 26-35. ACM, 2014.

[6] Official Journal of the European Union, Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), Vol 119, 2016-05-04

[7] Centers for Medicare & Medicaid Services (1996), The Health Insurance Portability and Accountability Act of 1996 (HIPAA), Online http://www.cms.hhs.gov/hipaa/. Referred 15.7.2018.

[8] International Electrotechnical Commission, Medical device software - Software life cycle processes, IEC 62304, 2015-06

[9] International Standards Organization, Medical devices - Quality management systems - Requirements for regulatory purposes, ISO 13485, 2016-03-01

[10] International Standards Organization, Medical devices - Application of risk management to medical devices, ISO 14971, 2007-10-01