Medical Software Needs Calm Compliance

Tuomas Granlund, Vlad Stirbu, and Tommi Mikkonen

Abstract—Traditionally, compliance activities in association with medical software have been considered document-oriented. However, with automated workflows it is possible to pinpoint the decisions that are taken, and document them so that comprehensive documentation can be generated as needed. Accomplishing this will embed compliance in the fabric of the development activities of a company to the extent that very few people in the organization are thinking about compliance besides the designated regulatory compliance professionals. In this article, we present such *calm compliance*, where compliance activities are carried out in a non-invasive fashion, without the concerns over delayed bug fixing, extended turnover time, stress, and drop in productivity that are commonly associated with them. The paper is based on several years of hands-on engineering of standalone software medical devices in industry, covering both in-house development as well as a subcontracting role.

Index Terms—Regulated software, compliance, agile development, change management, calm computing, calm compliance

1 Introduction

The promise of *calm computing* [14], a concept introduced by Mark Weiser over 20 years ago, has been mostly realised today. For instance, as an end user, you expect to interact during your daily routine with a multitude of sophisticated services and applications, without being aware of their technical implementation. All you need is a smartphone and an always-on internet connection.

However, application providers and software developers perceive the calm technology reality differently than end-users. They need a deep understanding of the specialized hardware and software used to deliver their applications. In addition, the involvement of other stakeholders can add complexity to the software development process. In the medical devices industry, in particular, that is subject to comprehensive compliance requirements, the regulatory authorities aim to ensure that the technology produced by the developers is fit for purpose and safe for end-users. Furthermore, there is a considerable disconnect between the medical domain, where empirical data is crucial, and software engineering, where evidence-based attitude is less common [8].

In this article, we introduce *calm compliance*, where compliance activities are carried out in a calm fashion, without the hassle that is commonly associated with them. Calm compliance – or low-ceremony compliance, which we also considered as the name – is the end result of applying agile, continuous software development activities in a regulated domain. The required regulatory activities are fed into the process in a way that does not lead to the big-bang type of integration, and, like with end-user applications, the software is used to carry out the tedious, time consuming, and humanly error-prone tasks in the process.

• T. Granlund is with Solita, Tampere, Finland.
E-mail: tuomas.granlund@solita.fi
• Vlad Stirbu is with CompliancePal, Tampere, Finland. E-mail: vlad.stirbu@compliancepal.eu
• Tommi Mikkonen is with University of Helsinki, Helsinki, Finland. E-mail: tommi.mikkonen@helsinki.fi

Manuscript received April 19, 2005; revised August 26, 2015.

To make calm compliance concrete, we use Software as Medical Device (SaMD) definitions [5] to observe how agile development methodologies can be augmented with regulatory activities required by the Federal Food, Drug, and Cosmetic Act (FD&C Act [10]) and forthcoming Medical Devices Regulation (MDR [2]). The viewpoint we assume is that of the developer, building on the applicable standards (see sidebar). To simplify, we leave other, possibly related compliance requirements outside the scope of the article, apart from possible inspiration we can take from them. The article is based on several years of experience in engineering standalone software medical devices in industry, covering both in-house development and consulting roles.

2 Software as Medical Device

Medical device software development has unique needs. Its design, development, and manufacturing processes are regulated. Hence, there must be proper control mechanisms in place to ensure the end product's safety, reliability, and ability to meet user requirements. These control mechanisms originate from the regulations' requirements, corresponding guidance documents, international standards, and national legislation. However, their plentiful existence is one of the reasons medical software is often considered a complex domain.

Plan-driven methodologies have been the preferred way to develop products in regulated industries. Their cultural affinity with the language and format used by standards referred to above have made them the natural choice. However, the long feedback loops that characterize these methodologies are even longer in the high ceremony process required to comply with regulations.

The applicability and rigor of the medical device regulatory requirements are based on the risk level of the product. It is not enough to have just a compliant end-product as the regulatory framework affects the operations of the entire manufacturing organization. The design and development of the device are done within a formal process that is part of a robust quality management system. The controlled product development process aims to produce high-quality software while simultaneously producing objective evidence to demonstrate compliance upon regulatory approval. The purpose of the evidence is to demonstrate that the development was done according to predefined procedures and, at the same time, to prove that the software meets its predetermined specifications and fulfils its intended use and user needs. In practice, the evidence includes test results, review records and automatically generated traceability records with an audit trail. It is essential that full traceability for all design and development elements can be demonstrated.

The overall goal of design and development controls is to manage the end product's risk level. Therefore, in addition to the quality management system, a regulatory-compliant risk management system is needed. In the case of medical software, risk management activities are implemented according to the requirements of ISO 14971 (see sidebar), where the focus is on safety-related risks. These can be divided into risks that emerge from the intended use of the medical software, usually directly affecting the patient (e.g., providing an overdose of medicine), and those related to the technology selection (e.g., operating system failure). Both types of risks are treated as product-related risks during and after development. The process includes risk identification, mitigation, and verification of implemented mitigation actions. Furthermore, the process assesses the acceptability of the residual risk when mitigation actions have been implemented.

As the product development process of the medical device is multidisciplinary in its nature, also the risk management process requires a multidisciplinary team to be genuinely effective, covering both clinical and technical risks. Clinical risks, i.e., risks that can compromise the clinical condition or the safety of the patient, have most often been identified well before the implementation process. However, in the implementation phase, the developers are responsible for identifying and documenting technical risks associated with the design, which are then connected to clinical risks. Streamlining the connection between clinical and technical risks requires teamwork between software developers, clinical experts, product owners, security experts and compliance officers, and forms the core of calm compliance.

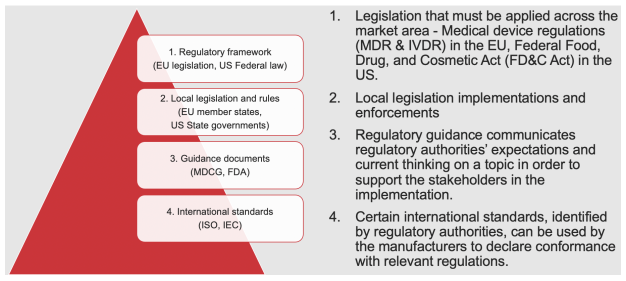
3 Pain Points Today

In addition to the usual complications related to software projects, the developers in the medical domain face certain unique challenges. The key pain points can be associated with regulations, guidance documents, and a selected set of standards that establish an interrelated set of requirements that must be interwoven into the development process. In the following, we list the most serious ones and provide a short rationale for their emergence.

Compliance-related activities frustrate developers. The benefits of agile methods and continuous software engineering also apply to medical software. Still, using them in medical software development introduces the same concerns as with any technology – how to deal with legal and regulatory bindings in a new context [13]. This culminates in the context of continuous software development, where

SIDEBAR: The regulatory landscape in the medical domain.

The medical domain's regulatory landscape can be interpreted to consist of several layers (Figure 1). For every phase within the product lifecycle – design, development, manufacturing, risk management, maintenance, and post-market processes – certain standards must be followed for regulatory compliance. The set of applicable standards for software include: general requirements for health software product safety (IEC 82304-1) [IEC82304], software life cycle process (IEC 62304 [IEC62304]), risk management process (ISO 14971 [ISO14971]), and usability engineering (IEC 62366-1 [IEC62366]). Furthermore, the manufacturers are expected to have a quality management system that must comply with further associated regulations – requirements of the Medical Device Quality Systems standard ISO 13485 [ISO13485] or its US counterpart, US FDA 21 CFR part 820. These standards form a minimum set of regulations to consider when developing medical devices with software. Although their total page count is not extensive as such, the requirements included can feel overwhelming as the standards come with rich, sometimes heavy information.



1. Legislation that must be applied across the market area - Medical device regulations (MDR & IVDR) in the EU, Federal Food, Drug, and Cosmetic Act (FD&C Act) in the US.
2. Local legislation implementations and enforcements
3. Regulatory guidance communicates regulatory authorities' expectations and current thinking on a topic in order to support the stakeholders in the implementation.
4. Certain international standards, identified by regulatory authorities, can be used by the manufacturers to declare conformance with relevant regulations.

Fig. 1. The regulatory landscape in the medical domain illustrated.

References

[IEC62304] International Electrotechnical Commission. IEC 62304:2006/A1:2015. Medical device software – Software lifecycle processes, 2015.

[IEC62366] International Electrotechnical Commission. IEC 62366-1:2015. Medical devices – Part 1: Application of usability engineering to medical devices, 2015.

[IEC82304] International Electrotechnical Commission. IEC 82304-1:2016.Health software – Part 1: General requirements for product safety,2016.

[ISO13485] International Organization for Standardization. ISO 13485:2016. Medical devices – Quality management systems – Requirements for regulatory purposes, 2016.

[ISO14971] International Organization for Standardization. ISO 14971:2019. Medical devices – Application of risk management to medical devices, 2019.

new releases can be made several times a day, but this is not benefited from because of regulatory constraints. Instead, the developers are stopped from deploying things until all the compliance and regulatory related processes are complete, forming a contradiction to Agile manifesto statement *Working software over comprehensive documentation* and breaking the natural flow of the development team. It is not uncommon to discover that the developers feel that regulatory activities do not improve the quality of the software and are, therefore, a waste of time. When the compliance controls are seen as an impediment to effective product development, the developers may start looking for ways to circumvent quality system requirements. For example, common development practices like refactoring can be hard to grasp from compliance perspective, due to their potential to change things to such a degree that previous completed compliance activity are invalidated, and have to be performed again. To complicate matters further, regulatory affairs professionals have often practiced in environments where the medical devices always include hardware, and where they typically follow a linear development model. Hence they might not have the skills and experience to operate in an agile software development environment, in particular when dealing with medical devices that consist of only software, are highly distributed, contain a large number of open source components developed by third parties, or rely on external software systems maintained by public cloud providers.

Gap between regulations and development activities. The legally binding legislation texts and international standards describe the expected results, but do not describe how to achieve those results. Therefore, practical expertise is required to define the steps required to achieve the objectives [6]. To complicate matters, many of the available project management and application life cycle management tools for software development require that the developers invest time and effort to keep them in sync instead of relying on automation. Software developers are professionals in software development, not regulation. They often resort to compliance over-engineering or adding extra effort to compliance-related activities to play it safe – even if those activities do not truly contribute to improved quality. Hence, the developers may view compliance as the necessary evil that must be considered but has little practical relevance. Consequently, the compliance activities are often put aside while creating software and resurrected only when the development task is completed. This resurrection often needs support from dedicated compliance personnel.

Regulations are written in a non-familiar language. Regulations, guidance documents, and industry standards have been written with careful consideration and precisely defined terminology to communicate the text's purpose without ambiguity. However, these governing documents' regulatory language and terminology might not be familiar to an organization. The regulatory framework, which consists of several different documents, is a complex totality that must be well understood to create an effective and compliant regulatory implementation. In addition, as the concrete requirements in the regulatory documents are often written vaguely, they can be subject to interpretation, implying that the practitioners may interpret even the key terminology in various ways. Unfortunately, regulatory approval processes are designed in such a way that they tend to verify interpretations only afterwards when the effort is already invested in the development. One practical way to overcome related problems is to educate the personnel regarding regulations and applying the learnings in software development. Moreover, endorsing commonly shared and understood terminology with Domain-Driven Design's Ubiquitous Language practice [3] might offer help.

Tooling silos. Traditionally, quality processes, and regulatory activities in particular, have been implemented using Application Lifecycle Management (ALM) tools that have a different philosophy than version control systems managing the software development lifecycle (SDLC). Typically, proficiency in using one tool does not translate into mastery of the other. While the ALMs are effective at recording high-ceremony document-centric activities, like change management boards decisions, the version control systems and associated tools excel at automating tasks, serving as integration points for various DevOps tools. As such, the attempts to align the two tool ecosystems is brittle and requires significant maintenance effort.

Due to the above pain points, much of the regulatory actions in software development are at risk to be rendered as illusion of control. Their effect is largely an add-on to daily development activities that the developers are familiar with, and hence compliance-related activities are only triggered when fundamentally unavoidable. To improve, we rely on calm compliance where compliance activities are part of daily routines and do not require elaborate, high-level ceremonies.

4 Towards Calm Compliance

The proposed calm compliance for software development aim at alleviating the above pain points. In general, the approach we propose is the full integration of compliance activities to the daily software-related activities, following the trend for lean thinking [11], rapid feedback cycles [1], and extensive tool support across all the development activities.

Use streamlined regulatory processes. The core of compliance should be carefully determining the intended purpose of the medical software, and its transformation to user needs and system requirements. In general, it is helpful to minimise the footprint of the requirements that need to be traced, rather than considering the superset of possibly related factors – the latter tends to generate considerably more tasks for the developers. For instance, the architectures of some medical software products should be segregated into a number of modules where some modules do not have a medical purpose. The non-medical device modules are not subject to the medical device regulations [7] and, as a result, can be addressed with more light-weight development processes. Understanding the balance between regulated and other requirements is critical for streamlined regulatory processes.

Perform compliance activities when change happens. When working with regulatory software, compliance is everyone's business. Hence, all team members should be prepared to work on compliance related activities, to the extreme that whenever code is committed to a source code repository, compliance related activities are invoked. In essence, this means that when committing code, there is a procedure concerning compliance checks, as proposed in [12].

To a large degree, automation can help to perform the compliance checks. However, the developer must be aware of them and their intended consequences, which must be documented. Furthermore, it is often a good practice to collect the change approval from the most qualified team members when change happens, as this may help in problematic cases.

Document architecture evolution. One of the fundamental artifacts in software development is the underlying architecture. Well-documented software architecture serves as a communication tool that facilitates the interaction between the development team and the other stakeholders, and a map that describes how the user needs are implemented into software components. In addition, IEC 62304 (see sidebar) requires the manufacturer to create software architecture and more detailed design documentation for certain architectural decomposition items. Due to the evolutionary and iterative nature of the development process, the architecture has to reconcile and capture both the intentional and the emergent design decisions resulting from individual changes made by the developers. The decisions can be captured using a markdown file structured according to MADR[1]. Further, the tooling for architecture management should be lightweight and accessible. For example, the C4[2] model for visualizing software architecture is serialized using a human but also machine readable domain specific language[3], an approach that provides a low entry barrier so that performing architecture activities is not limited to specific individuals in the team. To meet compliance requirements, the supporting tools should support versioning, so that the evolution of the architecture can be documented effectively and traced to the requirements that triggered the respective change.

Use *documentation as code* practices. Regulatory documentation is the "sum of all docs" created during product development. Today, a common perception is that these documents and their traceability form the core of the work, at the expense of technical development. However, in true calm computing, the tools become the documentation. For example, Git can act as a ledger, where changes are tied to commit hashes rather than archival tools such as SharePoint. In general, rich text formats like Markdown or AsciiDoc, combined with domain-specific languages for capturing technical diagrams like PlantUML[4] or Mermaid[5], provide the right affordances that allow the documentation to be maintained and versioned in Git but also to build pipelines that generate end user friendly content for online (e.g., a website) or offline (e.g., a PDF document) consumption.

1. https://adr.github.io/madr/
2. https://c4model.com
3. https://github.com/structurizr/dsl
4. https://plantuml.com
5. https://mermaid-js.github.io/mermaid

5 CompliancePal Tool Chain for Calm Compliance

CompliancePal[6] is a commercially available software product that aims at streamlining compliance activities with software development. As of today, there are two pilot customers, who have provided support for externally validating the system. The experiences presented in this article largely stem from the pilot customers as well as from discussions with other manufacturers of medical software.

The CompliancePal system takes into account the elements of calm compliance introduced in Section 4, assisting the daily routine of an agile development team practicing the Scrum methodology [12]. Among the agile team we emphasize four relevant roles that are needed in a medical-oriented Scrum team – the product owner, the software developer, the architect and the compliance officer. Like with any Scrum team, they are jointly responsible for solving the emerging regulatory problems. Additionally, the product owner ensures that the appropriate quality management system is in use. A quote from CompliancePal users, "*We like opinionated compliance tools, because we do not have to come up with a solution ourselves*", suggests that this is well appreciated by the developers.

The code produced by the team is managed using a Git repository hosted on GitHub[7]. The compliance checks are performed by our service that extends the standard GitHub workflows using the Apps[8] integration method. The check results are exposed using the GitHub's Statuses functionality associated with each commit. Possible compliance problems are brought to the attention of the team via dedicated chat room hosted in Slack[9].

The architecture related documentation describing how the software components implement the product requirements, their hierarchy, and how they interact with each other is managed in a repository in GitHub, in a similar fashion as the rest of the code produced by the team.

The compliance officers are able to react to the detected problems that require their attention via the team communication channel. Following the link, the officer can handle the problem using CompliancePal's user interface, a web application dedicated to compliance activities, but linked to GitHub at the backend to automate as much as possible. This way, the compliance officer does not have to be familiar with the GitHub service as used by the developers.

The compliance workflows can be configured to include also the functionality provided by third parties, such as Dependabot[10] or WhiteSource[11] for managing the open source dependencies and their vulnerabilities, or Fossa[12] for managing and enforcing license policies for open source dependencies.

By using directly git and the GitHub provided APIs, CompliancePal is able to leverage the existing capabilities of the ecosystem described in Fig. 2, but also contribute to the creation of new tooling.

6. https://compliancepal.eu
7. https://github.com/
8. https://developer.github.com/apps/
9. https://slack.com/
10. https://dependabot.com
11. https://www.whitesourcesoftware.com
12. https://fossa.com

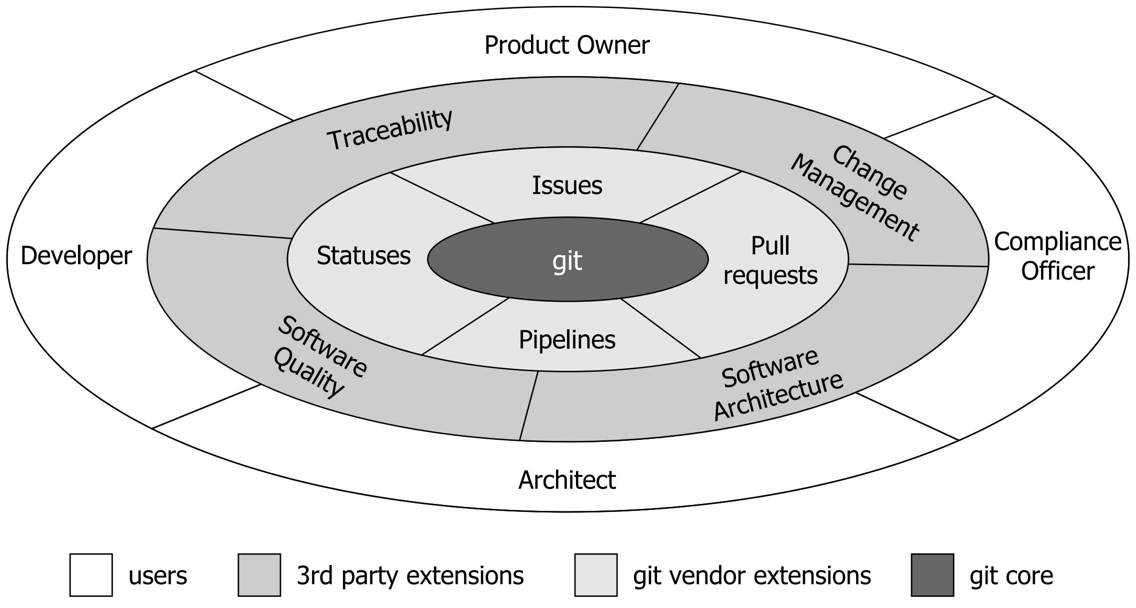


Fig. 2. Calm compliance empowers users with diverse skills to perform their activities using a common tool ecosystem. Built around git, leveraging the functionality provided by the git vendors and other services provided by third parties like CompliancePal, the ecosystem forms a continuum that presents specialised views suited for each user needs. Working together, at the speed determined by the software development, the team ensures that compliance becomes a mundane activity, completed at the end of each increment.

TABLE 1
CompliancePal solution for calming the pain points of compliance.

Impediment	CompliancePal solution
Compliance-related activities frustrate developers.	CompliancePal hooks into GitHub, and seamlessly embeds compliance related actions alongside mainstream software development, streamlining the regulatory processes.
Gap between regulations and development activities.	Everything takes place in GitHub; if necessary, separate interface can be created for compliance officers to simplify their operations. Furthermore, by relying on pull requests, compliance activities are executed when a change happens.
Regulations are written in a non-familiar language.	Regulatory workflows formalised as GitHub actions and CompliancePal extensions. Developer input is collected using a familiar user interface and friendly language.
Tooling silos.	Single tool ecosystem build around GitHub. Team rhythm is determined by pull requests.

To summarize, Table 1 presents a mapping from the main pain points to CompliancePal features.

6 Recommendations

To truly reach the calm compliance, related activities should be embedded in the fabric of the development approach of a company, to the extent that very few people in the organization are thinking about compliance besides the designated regulatory compliance professionals. The rest of the personnel, in contrast, spend their days solving problems, delivering products, and managing processes, and compliance is built-in in all of them.

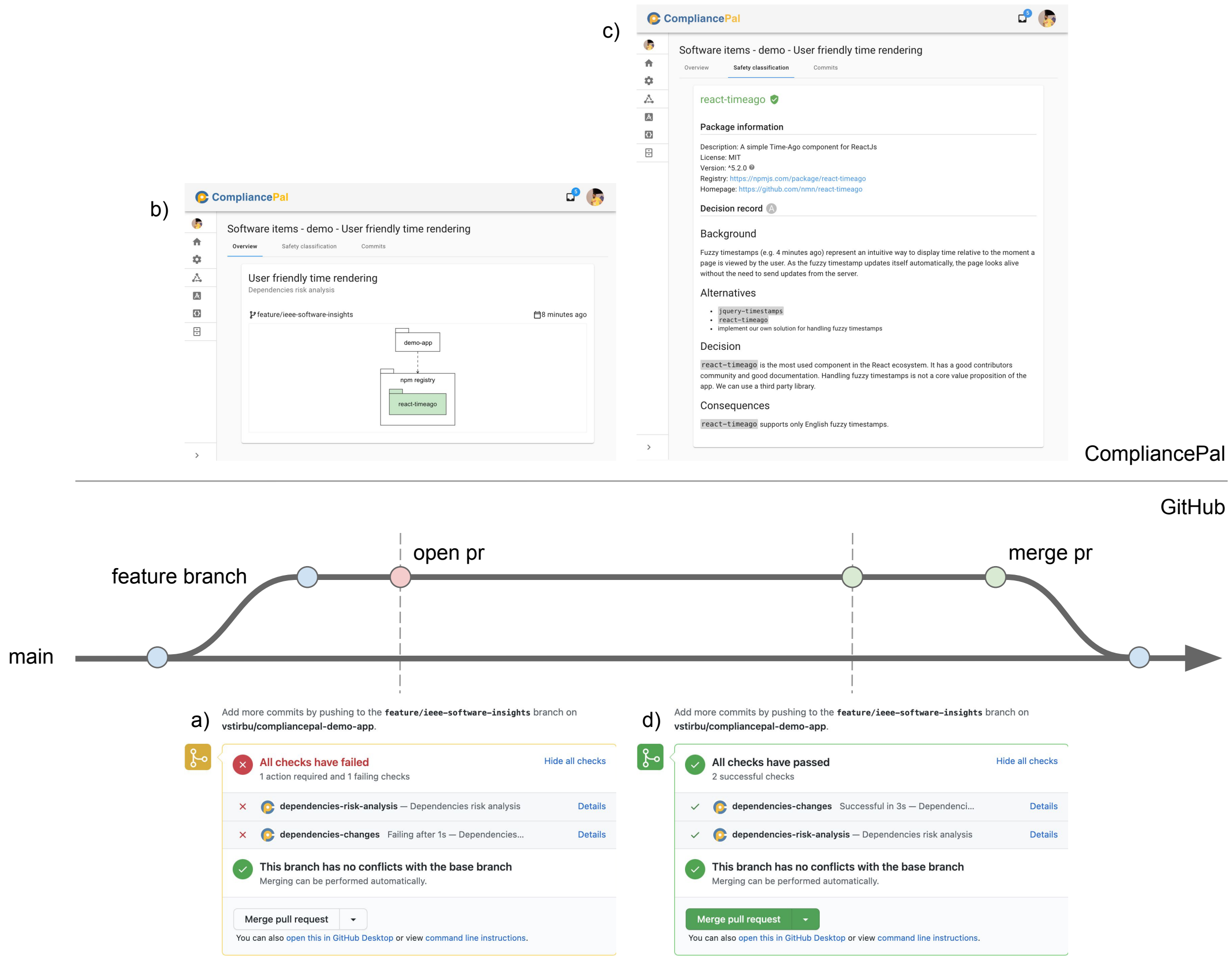
As demonstrated above, steps towards reaching the vision are related to every-day tools and practices. Integrated regulatory activities in tools used by the developers are the first step in this process. This in particular concerns pull requests, which are the way to introduce changes to software, but which can also be used as means to manage compliance with respect to changes in code. Furthermore, tooling to present artifacts stored in version control system or produced by CI/CD pipelines will facilitate the participation of all team members – including also compliance officers – in the development activities, as advocated in agile methods.

While tooling can integrate compliance related actions in continuous development, cultural differences may prevent calm compliance, even if tooling was in place. In our experience, companies with established medical development are rather looking for a separate process and application to consider the regulatory issues, thus keeping the compliance officers distant from the development team. This is in line with their established organizational structures, which are common in large companies.

In contrast, organisations that are familiar with continuous software development, relying on teams that are self-organizing, but are new to regulatory aspects have found the CompliancePal tool and the related approach fit for their use cases. Here, compliance officers align to the development activities, not the other way around. Achieving this requires reconsidering traditional distribution of tasks using well-considered, yet relatively simple tooling. So far, with the presented implementation, there has been minimal number of change requests emerging from pipeline users. We take this as evidence of the viability of its feature set for calm compliance.

Finally, it is important to notice that some elements of calm compliance cannot be achieved with development tools alone, but require a change in mindset and behavior. One example is constituted by security threat analysis

SIDEBAR: Managing 3rd party software dependencies with CompliancePal



Software items - demo - User friendly time rendering

react-timeago

Package information

Description: A simple Time-Ago component for ReactJs
License: MIT
Version: ^5.2.0
Registry: https://npmjs.com/package/react-timeago
Homepage: https://github.com/nmn/react-timeago

Decision record

Background

Fuzzy timestamps (e.g. 4 minutes ago) represent an intuitive way to display time relative to the moment a page is viewed by the user. As the fuzzy timestamp updates itself automatically, the page looks alive without the need to send updates from the server.

Alternatives

- `jquery-timestamps`
- `react-timeago`
- implement our own solution for handling fuzzy timestamps

Decision

`react-timeago` is the most used component in the React ecosystem. It has a good contributors community and good documentation. Handling fuzzy timestamps is not a core value proposition of the app. We can use a third party library.

Consequences

`react-timeago` supports only English fuzzy timestamps.

Software items - demo - User friendly time rendering

User friendly time rendering
Dependencies risk analysis

feature/ieee-software-insights 8 minutes ago

Add more commits by pushing to the **`feature/ieee-software-insights`** branch on **vstirbu/compliancepal-demo-app**.

All checks have failed
1 action required and 1 failing checks

✗ **dependencies-risk-analysis** — Dependencies risk analysis
✗ **dependencies-changes** Failing after 1s — Dependencies...

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

Add more commits by pushing to the **`feature/ieee-software-insights`** branch on **vstirbu/compliancepal-demo-app**.

All checks have passed
2 successful checks

✓ **dependencies-changes** Successful in 3s — Dependenci...
✓ **dependencies-risk-analysis** — Dependencies risk analysis

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

Fig. 3. Workflow for handling 3rd party software components

Handling 3rd party software components in medical software devices is a chore. While these components allow the developers to rapidly extend its functionality by reusing code developed by others, the drawback is that the 3rd party code becomes part of the medical product, and hence it is governed by the same software development lifecycle rules as the own code.

The procedure behind the workflow is that every 3rd party software component is accompanied by a change decision that describes the rationale for using the component. The change decision is structured following the architecture decision record (ADR) format [9]. To enforce the workflow, CompliancePal analyses the code committed by developers in pull requests to identify if new 3rd party software components have been added. If new components are added, the commit check is marked as failed (a). When the developer or architect that originated the change adds the change decision, the compliance officer can open the commit check details to view the changes to the 3rd party components using a unified view, representing a component diagram diff that is common for all programming languages (b). The change decision view (c), augmented with data fetched from the package registry, enables the compliance officer to effectively performing the required risk analysis activities. When these activities are performed, and the component is assigned a safety classification, the commit check changes to pass (d). The change set can be merged now into the common code baseline.
Comparing with a traditional plan-based development methodology, where the compliance officer would have to ask the development team what changes related to 3rd party components have happened during an increment, this approach brings the stakeholders into the process at the time of change. The risk management activities happen during the pull request review, involving only the relevant team members, which makes compliance a low-ceremony team activity.

and modelling, which are widely used practices during application development in many fields, including also the medical domain [4]. In this context, tools like OWASP Threat Dragon[13] facilitates the identification of security threats early on and mitigate the risks, helped by methodologies like STRIDE[14]. However, despite similarities with the risk analysis in the area of medical devices, practitioners have not introduced unified threat models, relying on spreadsheets in their work instead. As a result the risk analysis practice is bound to the own experience of the practitioner that analyses the risks. Then, without common conventions, beyond those that ISO 14971 prescribes, these threat assessments are not portable, essentially preventing the development of uniform tooling.

In summary, we believe that we are at the brink of a transition towards more agile development in medical domain, enabling the delivery of new features at software speed, as well as lowering the barrier of entry for new companies and innovative products. A critical element in the transition is calm compliance, where compliance no longer is a high-ceremony operation but business as usual.

Acknowledgments

The authors would like to thank Business Finland and the members of AHMED (Agile and Holistic MEdical software Development) consortium for supporting this work.

References

[1] Jan Bosch. *Speed, data, and ecosystems: Excelling in a software-driven world.* CRC press, 2017.

[2] European Parliament and the Council. Regulation (EU) 2017/745 on medical devices, 2017. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02017R0745-20200424#tocId168 Accessed Feb 16, 2021.

[3] Eric Evans and Eric J Evans. *Domain-driven design: tackling complexity in the heart of software.* Addison-Wesley Professional, 2004.

[4] Tuomas Granlund, Juha Vedenpää, Vlad Stirbu, and Tommi Mikkonen. On medical device cybersecurity compliance in eu. *arXiv preprint arXiv:2103.06809*, 2021.

[5] International Medical Device Regulators Forum IMDRF. Software as a medical device (samd): Key definitions, 2013.

[6] Teemu Laukkarinen, Kati Kuusinen, and Tommi Mikkonen. DevOps in regulated software development: case medical devices. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*, pages 15–18. IEEE, 2017.

[7] Medical Device Coordination Group (MDCG). MDCG 2019-11. Guidance on Qualification and Classification of Software in Regulation (EU) 2017/745 – MDR and Regulation (EU) 2017/746 – IVDR, 2019.

[8] Artur Nowak and Holger J Schünemann. Toward evidence-based software engineering: lessons learned in healthcare application development. *IEEE Software*, 34(5):67–71, 2017.

[9] Michael Nygard. Documenting architecture decisions, 2011. https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions, accessed Mar 28, 2021.

[10] U.S. Department of Health and Human Services. Federal food, drug, and cosmetic act. https://www.fda.gov/regulatory-information/laws-enforced-fda/federal-food-drug-and-cosmetic-act-fdc-act Accessed Feb 24, 2021.

[11] Mary Poppendieck et al. Principles of lean thinking. *IT Management Select*, 18(2011):1–7, 2011.

[12] Vlad Stirbu and Tommi Mikkonen. CompliancePal: A tool for supporting practical agile and regulatory-compliant development of medical software. In *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 151–158. IEEE, 2020.

[13] Dana R Wagner. The keepers of the gates: Intellectual property, antitrust, and the regulatory implications of systems technology. *Hastings LJ*, 51:1073, 1999.

[14] Mark Weiser. The computer for the 21 st century. *Scientific american*, 265(3):94–105, 1991.

13. https://owasp.org/www-project-threat-dragon/
14. https://en.wikipedia.org/wiki/STRIDE_(security)

Tuomas Granlund is a quality manager and a regulatory compliance specialist at Solita Ltd., Finland and a doctoral student at Tampere University, Finland. Contact him at tuomas.granlund@solita.fi.

Vlad Stirbu is the founder of CompliancePal, Finland. Contact him at vlad.stirbu@compliancepal.eu.

Tommi Mikkonen is a professor of software engineering at University of Helsinki and University of Jyväskylä, both located in Finland. Contact him at tommi.j.mikkonen@jyu.fi.