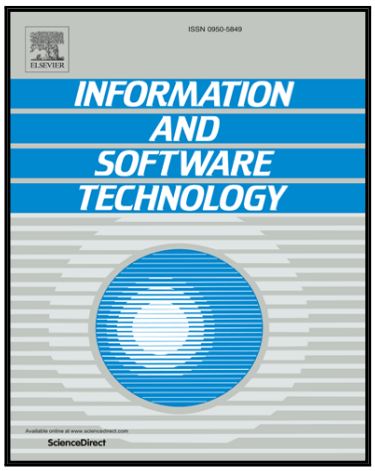


Regulated Software Meets DevOps

Teemu Laukkarinen, Kati Kuusinen, Tommi Mikkonen

PII: S0950-5849(18)30014-4
DOI: 10.1016/j.infsof.2018.01.011
Reference: INFSOF 5949

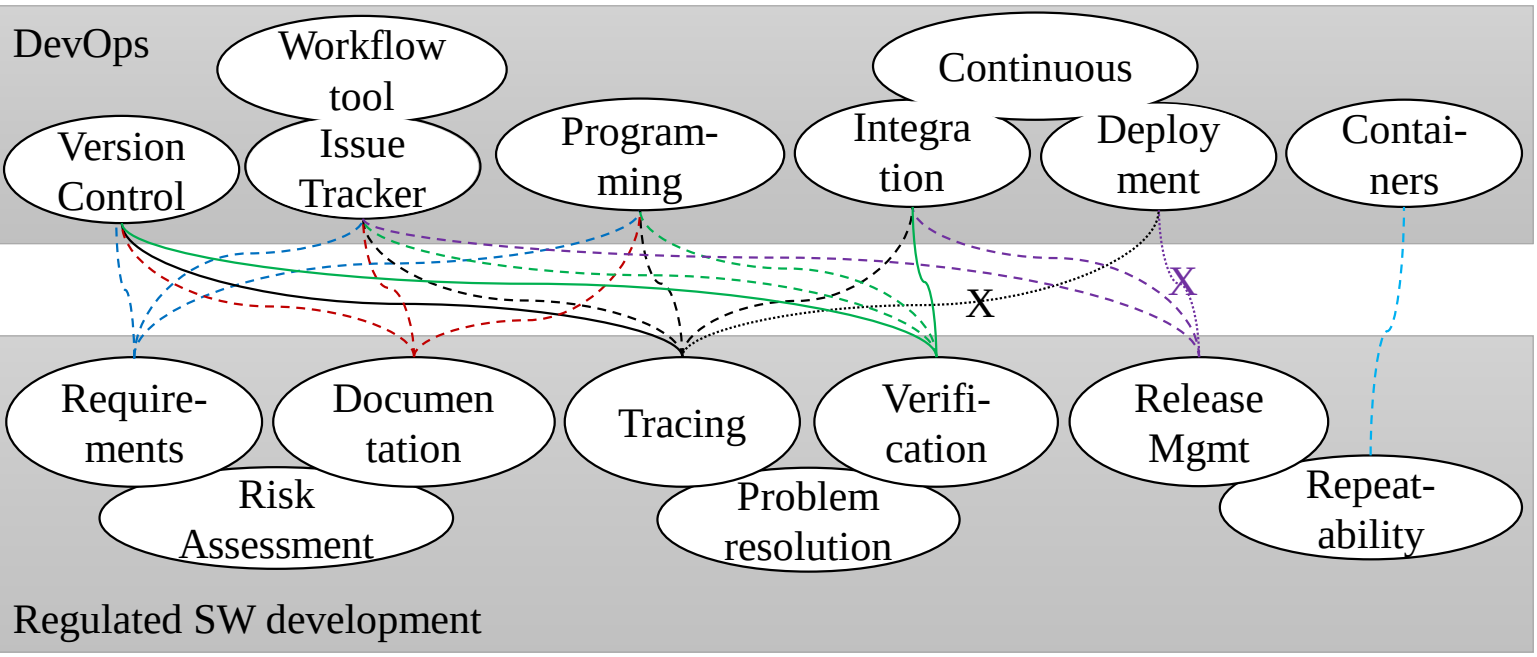
To appear in: *Information and Software Technology*

Received date: 26 September 2017
Revised date: 20 December 2017
Accepted date: 25 January 2018

Please cite this article as: Teemu Laukkarinen, Kati Kuusinen, Tommi Mikkonen, Regulated Software Meets DevOps, *Information and Software Technology* (2018), doi: 10.1016/j.infsof.2018.01.011

Highlights

- Research on DevOps and regulated software development is scarce.
- Standards for regulated development limit the use of DevOps practices.
- DevOps tools could help with strict tracing requirements if further developed.
- DevOps should automate documentation and template generation where applicable.
- Standards should provide templates that the tools can implement.



Current workflow:

- Requires manual work
- Automated / inherent relation
- X Blocked by standards

Suggested actions:

- Improve tool integration for tracing
- Provide standard obeying templates
- Update standards where applicable

Regulated Software Meets DevOps

Teemu Laukkarinen

Tampere University of Technology, Tampere, Finland

Kati Kuusinen

University of Southern Denmark, Odense, Denmark

Tommi Mikkonen

University of Helsinki, Helsinki, Finland

Abstract

Context: Regulatory authorities require proofs from critical systems manufacturers that the software in their products is developed in accordance to prescribed development practices before accepting the product to the markets. This is challenging when using DevOps, where continuous integration and deployment are the default practices, which are not a good match with the regulatory software development standards.

Objective: We aim to bring DevOps and regulated software development closer to each other. First, we want to make it easier for developers to develop regulated software with tools and practices they are familiar with. Second, we want to allow regulatory authorities to build confidence on solutions provided by manufacturers by defining a mapping between DevOps and regulatory software development.

Method: We performed a literature survey and created research suggestions using exploratory research.

Results: Tighter integration between development tools, requirements management, version control and deployment pipeline would simplify the creation of regulatory compliant development practices.

Conclusions: Regulations could be improved for more agile and incremental method in quality approval, the final step before the actual deployment of the software. Improved development practices and tool integration, created in cooperation by tool vendors, system providers, and regulatory authorities, could support developers who are not comfortable with fixed, and rigid practices of regulated software development.

Keywords: Regulated software, DevOps, Standards

1. Introduction

Numerous industries require reliability, visibility, and traceability of the software project to ensure safety and trustworthiness of the result. These industries have regulatory authorities, which require proofs from manufacturers that the software running in their products is developed in accordance to prescribed practices before accepting the product to the markets.

Regulatory authorities have requirements for the product development process that must be fulfilled, such as proofs of the software verification and traceability of the software development process. The main challenge in plan-driven methods for critical systems software development is in requirements management, particularly in the inability to accommodate changes once the development has begun [1].

So far, agile and lean methods have been tailored for regulated development by enriching them with planning practices [2]. However, there is not yet evidence how DevOps [3] fits to these regulatory standards. As the term DevOps has ambiguity in software industry [4] and our work focuses on practices and tools, we define DevOps as "practices that reduce and join repetitive tasks with automation in development, integration, and deployment".

This paper proposes ways to improve tools and practices used in DevOps context so that they would better meet regulatory requirements and thus simplify regulatory software development. The paper is a sequel to [5], where we identified key inhibitors of DevOps in the domain of medical software based on a literature survey and a number of designs.

Our previous work [5] could not identify related research on DevOps for medical device software development, and, in fact, research on DevOps in embedded software domain – which medical device software essentially is – is also scarce [6]. Ebert et al. [7] add that continuous deployment to the customer environment is not a feasible goal for safety-critical software: failure is not an option, and rapid customer feedback becomes largely inapplicable. However, they state that quick and reliable

Email addresses: teemu.laukkarinen@tut.fi (Teemu Laukkarinen), kaku@mmmi.sdu.dk (Kati Kuusinen), tommi.mikkonen@helsinki.fi (Tommi Mikkonen)

delivery is possible also in regulated environments, although accomplishing this might not be as straightforward as with simpler contexts.

2. DevOps and Regulations: Sample of two Standards

Previously [5], we examined two medical device and health software IEC/ISO standards, which were suggested by an industrial partner working on medical device software. The standards are *IEC 62304 – Medical Device Software – Software Life Cycle Processes* [8] and *IEC 82304-1 – Health Software – Part 1: General Requirements for Product Safety* [9]. These standards are related; IEC 82304-1 uses the software life-cycle model of IEC 62304 while giving eases in verification activities, as it is not applicable for life threatening medical devices. The intended medical use of the end product determines the applicable standard: IEC 62304 is for any medical device that contains hardware provided by the device manufacturer whereas IEC 82304-1 is for health software that runs on general purpose hardware that may be acquired and controlled by the customer. IEC 62304 is also applicable for medical software used for medical treatment or diagnosis, even if the software runs on general purpose hardware.

IEC 62304 defines a software life-cycle model that consists of planning, requirement gathering, design, implementation, verification, integration testing, system testing, and releasing activities as separate phases. Moreover, risk management, configuration management, and problem tracing are expanded over the whole life-cycle. In IEC 62304, the configuration management means tracing all the phases of all the identifiable deliverable of the software, including software units, documentations, test cases, and test reports to the initial software requirements. Risk management should follow ISO 14971 risk management for medical devices standard. Furthermore, ISO 13485 – an extended version of ISO 9001 quality management standard for medical devices – should be used for quality management.

While adopting the software life-cycle model of IEC 62304, IEC 82304-1 adds extra processes and activities before and after the life-cycle model to satisfy all the regulatory requirements. Such additional parts include user instructions' contents and post-market activities.

Neither of the standards specify the practical execution of their processes and activities. Thus, they allow the use of any tools and methods as long as the required activities are completed in accordance to the standards.

Regulatory software development may demand that the development tools, frameworks, libraries and operating system software are regulated too. IEC 62304 defines Software of Unknown Provenance (SOUP) for any included software that is not implemented according to the standard. All the SOUP items must be considered in the risk assessment. The implemented software must then contain risk control measures to reduce possible consequences of any risk realization to acceptable level. IEC 62304 does not require development tools to be regulated, but it does require documentation and tracing of them and their versions to ensure repeatability of the life-cycle afterwards.

3. A Call to Action

Today, software development has split on two major tracks. One is based on lean and agile methods, where fast development and deployment are the key focus through practices such as DevOps. The other track that relies on documentation-heavy development for regulated software to ensure the correctness, reliability, and safety of the software before the deployment. The lean and agile methods may be more prone to errors that end up into the final product, but it is also faster to find and fix those errors. Thus, there is clear motivation to fuse the best practices of these tracks together. As result, the lean and agile methods might produce higher quality, and the documentation heavy regulated software development might become easier and faster. In following, we discuss fusing these tracks in point-of-views of DevOps and regulated software development practices.

3.1. Improving Tools

Put briefly, DevOps is about automated tool chain, and regulated software is about ensuring the correctness of the software. There is no inherent conflict between these two, but at the moment the DevOps tool chains do not realize the precision of the tracing requirements of regulated software development. We argue that the value of using DevOps could be increased for the regulated software development with practices introduced in the following list, which was gathered by studying JIRA, GIT, Jenkins and Docker in relation to IEC 62304:

- Item tracking across tools should be a standard practice. Traces of items are needed, starting from the requirements and ending with the final product. Today, items must often be connected together manually. As an example, Figure 1 presents simplified traceability requirements for software items of IEC 63204. Any item related to Requirement B must be traceable over whole version history and at every point of the life-cycle inside the workflow tool. We argue that the workflow, version history, and continuous integration tools should form connections automatically to reduce non-productive development work. To the best knowledge of the authors, presently there is no de-facto tool chain to support that.

- Tools should include standard templates that comply with regulatory requirements. These templates should be available for all the traced items and the workflows that the items must follow.

- The tools should work hierarchically. In the best scenario, the developer can create, link, and version control requirements, consequential software items, test items, test reports, and so forth starting from the workflow tool and moving down to the corresponding implementation and testing tools, without connecting any items manually. On the other direction, any changes to the items should propagate back upwards so that a change can invalidate accepted test runs, for instance. Figure 2 illustrates this.

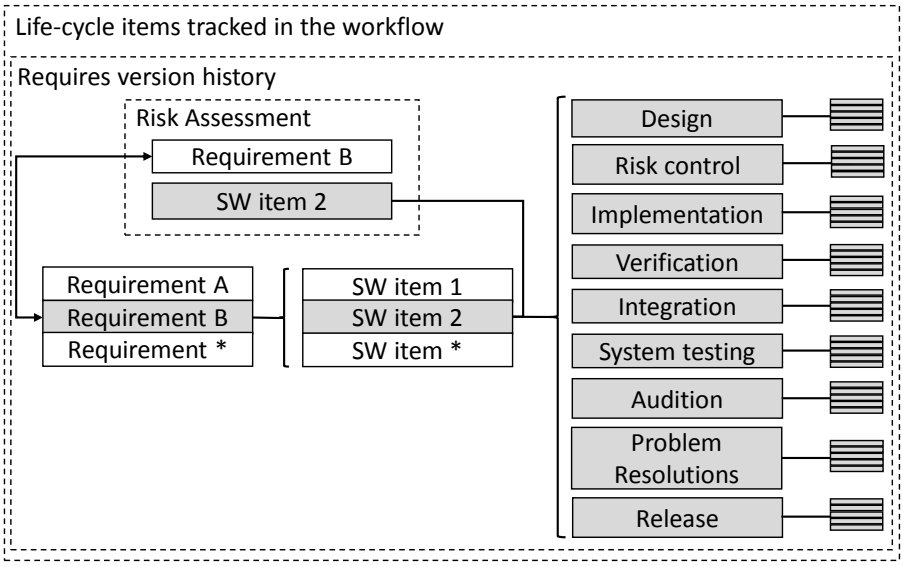


Figure 1: An illustration of the traceability requirements of IEC 63204 life-cycle between Requirement B and software (SW) Item 2. The sub-processes and their items are omitted for presentation clarity.

- The tools should guide the developer to follow the workflow in a fashion that complies with the regulatory requirements. This would make it easier for new developers to work on regulated software development, which in turn would reduce training costs and eliminate errors in following the process.

3.2. Pushing the Envelope

We also argue for further usage of DevOps in the regulated software development. The development team could use continuous delivery internally, and the workflow tool could visualize how close the development is to a release and what is yet to be done. Documentation and its generation could be integrated in the delivery pipeline to increase regulatory authorities' confidence to the software and help developers to create all the required documentation items. Staging environments could use real-life, potentially live data in testing, and compare it to the previous version of the system. Finally, with virtualization, the whole development process and deployment could be archived so that it can be brought back to alive, replayed, and examined later to investigate what went wrong if the software fails after deployment.

3.3. Redefining Regulatory Requirements

Regulations and accompanied standards could be improved to better relate the regulated software development with DevOps practices. In particular, standardized documentation, test, and audition reports that tools generate for the authorities would be a low-hanging fruit. Furthermore, the tool chain could prevent generating such before the required steps are completed, and the uncertainty of the developers towards "what is enough documentation" would definitely be reduced.

We completed this study without considering requirements for regulated development tools, as IEC 62304 does not require such. Consequently, such requirement would limit the available tools considerably. Related to this, we find two suggestions. Firstly, regulatory authorities should consider adopting the way of IEC 62304, where possible flaws of the tools are considered in the risk assessment. After all, the tool chain at its finest starts from the correctness of the CPU. Secondly, the suggested improvements to the tools would also make it easier to implement regulated development tools.

4. Summary and Conclusions

In this paper, we have discussed adopting DevOps practices and tools in regulated software development; more subjective aspects like developer and company cultures are overlooked here, but we plan to study them in the future work. We propose developing DevOps inspired methods that help developers deal with requirements of tracing, documentation, repeatability, and deployment, and automating tedious but necessary activities. Creating practices that are applicable across multiple standards may be difficult, but close discussion should be established between the regulatory authorities, standardization, and developers over the needs of software development for future editions of the standards.

References

[1] M. Mc Hugh, O. Cawley, F. McCaffcry, I. Richardson, X. Wang, An agile v-model for medical device software development to overcome the challenges with plan-driven software development lifecycles, in: Software Engineering in Health Care (SEHC), 2013 5th International Workshop on, IEEE, 2013, pp. 12–19.

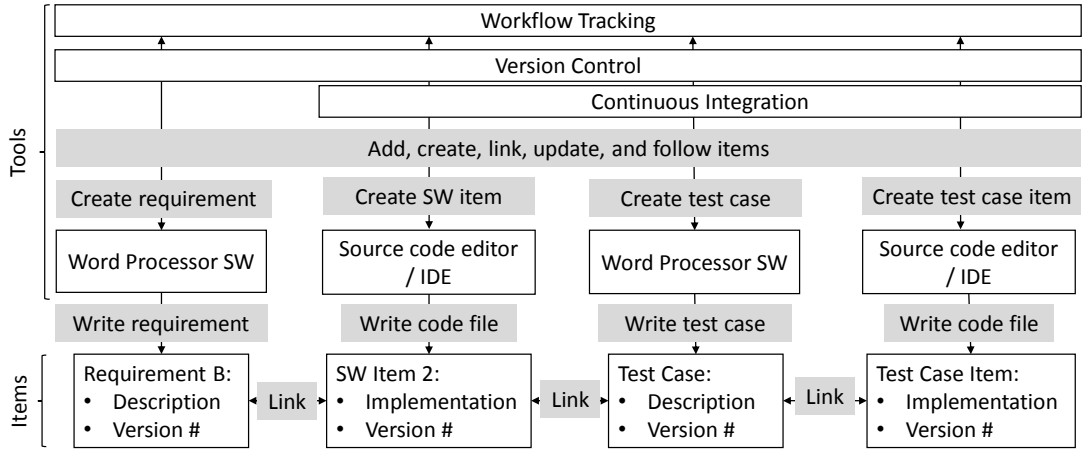


Figure 2: An example of hierarchical use of tools. The shaded boxes indicate actions that the tools could help perform through automation during the workflow. For example, creating a software item would link it to a requirement, create a stub file, version control it, and open the preferred editor for writing the code. The workflow steps are reduced and simplified for presentation clarity.

[2] O. Cawley, X. Wang, I. Richardson, Lean/Agile Software Development Methodologies in Regulated Environments – State of the Art, Springer Berlin Heidelberg, 2010, Ch. Proc. Lean Enterprise Software and Systems: First International Conference, LESS 2010., pp. 31–36. doi:10.1007/978-3-642-16416-3_4.

[3] P. Debois, DevOps: A software revolution in the making, Journal of Information Technology Management 24 (8) (2011) 3–39.

[4] J. Smeds, K. Nybom, I. Porres, DevOps: a definition and perceived adoption impediments, in: International Conference on Agile Software Development, Springer, 2015, pp. 166–177.

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

[6] L. E. Lwakatare, T. Karvonen, T. Sauvola, P. Kuvaja, H. H. Olsson, J. Bosch, M. Oivo, Towards DevOps in the embedded systems domain: Why is it so hard?, in: System Sciences (HICSS), 2016 49th Hawaii International Conference on, IEEE, 2016, pp. 5437–5446.

[7] C. Ebert, G. Gallardo, J. Hernantes, N. Serrano, Devops, IEEE Software 33 (3) (2016) 94–100.

[8] IEC, 62304: 2006 medical device software – software life cycle processes, International Electrotechnical Commission, Geneva, Switzerland.

[9] IEC, 82304-1: Health software – part 1: General requirements for product safety, International Electrotechnical Commission, Geneva, Switzerland. pp. 30.