

Agility in the avionics software world

Andrew Wils, Stefan Van Baelen, Tom Holvoet and Karel De Vlamincx

K.U.Leuven DistriNet
Department of computer science
Celestijnenlaan 200 A, 3001 Leuven
andrew.wils — stefan.vanbaelen —
tom.holvoet — karel.devlamincx @ cs.kuleuven.be

Abstract. This paper¹ takes a look at how XP and other agile practices can improve a software process for the development of avionics software. Developers of mission critical airborne software are heavily constrained by the RTCA DO-178B regulations [7]. These regulations impose strict rules regarding traceability and documentation that make it extremely hard to employ an iterative software development process. In particular, the extra validation overhead increases the time spent on small iteration cycles (for example, a bug-fix) to several weeks.

Currently, this sector is also pressed to switch to a more agile, customer driven approach. In this paper we investigate how to speed up development and cope with changing requirements using agile techniques. The research was carried out in cooperation with Barco, a major Belgian avionics equipment supplier. We explain why certain agile techniques have less effect as the project progresses. We point out the stadia in which each XP practice is beneficial and where XP practices might cause a slowdown.

1 Introduction

The upcoming popularity of agile software development is creating a pressure for application domains where less flexible software development processes are currently used. The avionics software industry is experiencing demands for a more customer oriented, agile software development approach. More specifically, this industry is confronted with late requirements changes and asked to shorten release cycles. While eXtreme Programming (XP) and other agile practices seem the obvious solution to deal with these demands, at the same time people are cautioned and advised to consider a more disciplinary approach for the development of mission-critical software. For example, Boehm and Turner [2] advise a more plan-driven approach when the software could involve loss of lives. Alistair Cockburn's crystal methodology [4] states that increasing criticality level means increasing the hardness of the method, resulting in more rigor, tighter control

¹ The described work is part of the EUREKA-ITEA AGILE project, and partly funded by the Flemish government institution IWT (Institute for the Promotion of Innovation by Science and Technology in Flanders)

and less tolerance. Unfortunately, due to a lack of experience with life-critical software development, the crystal level L (Life critical) is not discussed in more detail. The fact that people suggest a plan-driven approach does not necessarily indicate a lack of trust in agile methods, but more an observation that certain plan-driven methods have been proven to provide software that passes certification.

Indeed, the mission critical nature of this software has led to stringent procedures and plans that could specifically exclude the use of agile methods. In this paper, we will show that for the avionics software world, agile improvements can be made while still respecting the RTCA DO-178B certification guidelines. To verify this, we worked together with the Belgian avionics equipment supplier Barco for a thorough analysis of a DO-178B compliant software process. The company was assessed with the following goals in mind:

- show how to optimize the software development process and still have full documentation and traceability at the end;
- enable late integration of requirements changes with minimal re-verification efforts.

While this assessment was focused on the DO-178B standard, our findings may be useful in general for mission-critical software development.

The paper itself is organized as follows. The next section takes a closer look at the DO-178B standard. This document imposes the most important software development constraints for the avionics sector. We explain in Section 3 how we came to the results of our study. We looked at team activities, team communication, the software process structure, project artifacts and project management. We found that a modified XP based process can shorten iteration cycles, provided that a number of technical obstacles can be solved. The availability of the right tools will be even more important than in traditional agile software development. The results are broken down in an analysis of the agile principles in Section 4 and a discussion of agile opportunities throughout the entire development process in Section 5.

2 Avionics Software Development

Avionics software development is heavily constrained by a simple, yet inflexible goal: *to prevent the loss of human lives*. This mantra rightfully adds suspicion to anything that may compromise the safety and security of aircraft personnel and passengers. For software, this resulted in the establishment of some strict guidelines for the development processes. Produced by Radio Technical Commission for Aeronautics, Inc. (RTCA), the DO-178B document has become the de facto standard of such guidelines. The USA's Federal Aviation Administration and many other national certification authorities regard this document as a necessary means to certify avionics software; this is specified in FAA Advisory Circular 20-115B.

The DO-178B document dates from 1992². Fortunately, it does not impose a specific software development life-cycle process. The document specifies (a) objectives for software life-cycle processes, (b) descriptions of activities and design considerations for achieving those objectives and (c) descriptions of the evidence that indicates that the objectives have been satisfied. In practice, this requires the delivery of multiple documents and records to verify traceability and testing of all requirements. These documents include:

- plans for verification, quality assurance and development;
- all requirements, software and the source code tree;
- problem reports, verification cases, procedures and standards.

The objectives are grouped according to levels of potential danger if the developed software should fail: A (catastrophic), B (hazardous-severe), C (major), D (minor), or E (no-effect). The most stringent levels (A and B) demand amongst others:

- independent reviews of tests and of requirements compliance;
- traceability of system requirements to the source code.

In addition, the DO-178B standard includes strict guidelines concerning tool use and reuse of software. If software artifacts are reused between projects, the certification evidence of these artifacts should be integrated in the certification evidence of the new project. It should also be of the correct rigor required for the targeted safety level. If a tool is used that in one way or another eliminates or automates compliance to certain objectives, certification evidence for such a tool is also required. A distinction is further made between verification tools and a development tools. Certification evidence for a development tool should be of the same rigor as required for the targeted safety-level as such a tool can directly introduce a bug into the airborne-system. A verification tool may be developed to a somewhat lower standard as it can only fail to detect a bug in the airborne system.

3 Industry Assessment

The avionics division of Barco develops man machine interface solutions for the avionics domain. Barco desired to improve the time-to-market and wanted to respond more quickly to customer requirements changes. However, it turned out that an agile methodology such as XP did not offer the necessary improvements, mainly because it addressed problems that were entangled with other aspects of avionics software development, and the DO-178B standard. We then performed an assessment consisting of the following activities:

- seminars and workshops about the DO-178B standard and the internal software process;

² A newer version is being prepared and will be called DO-178C.

- discussions and interviews with developers, other team members and reviewers;
- visits to development labs and plant.

While looking for bottlenecks in software development, we found that the software development is dependent on hard to control external factors, such as hardware co-development. While software development could be made on a simulated hardware platform, testing for certification always needs to be done on the final product. For example, automated environmental tests can still take up to several days. We will see that this has a major impact on the overall agility of the project.

4 Agile principles

The agile principles lie at the heart of most agile methodologies. They are defined alongside the *Agile manifesto* [1]. Before trying to bring agility into a software process, we first check whether the agile principles support avionics software development. Also, they must not contradict or interfere with the DO-178B standard. It turns out that most principles can be applied in a certification driven process without any changes. We needed to reinterpret 3 principles. However subtle these changes are, they will still have an effect on how agility can be applied to an avionics software process. These are the subtle yet important comments on the principles:

Principle: Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

This principle applies, but valuable avionics software means software suitable for flight operation, which needs much more work than the ordinary, “tested” software that was targeted by this principle.

Principle: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

This principle applies, but much information that is exchanged needs to be logged and documented. Face-to-face, informal communication is hard to capture in documents, and could in fact contradict the produced documents.

Principle: Working software is the primary measure of progress.

This principle only partly applies: working software means nothing without certification.

5 Agility analysis and opportunities

To see what bottlenecks we can alleviate with agile techniques, consider Figure 1. Since a traditional software process suffers from its complexity, the effort needed to add functionality increases as the project progresses. Agile processes such as XP aim for an ideal, flattened curve, allowing a constant development pace.

At the beginning of a project, certification driven software development follows these curves. We call this the *software phase* of the project. A first divergence can be seen in the figure when deployment tests begin: the software is ready to get tested in the field. Here, the process slows down because of hardware dependencies and (partly automated) acceptance testing. These issues are common in embedded software development (e.g., see [6]). Hence, we call this the *embedded phase*. An even more significant slowdown is encountered when the software is ready to be certified. In this stadium, that we call *certification phase*, the software is presumed bug-free, but much documentation and manual testing is needed to provide the artifacts that are necessary for certification.

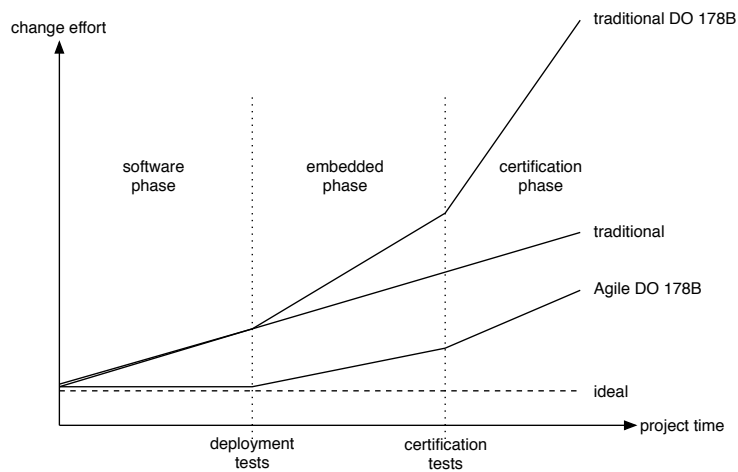


Fig. 1. Software processes compared.

For simplicity, the figure does not indicate the certification itself, after which every modification needs to be recorded in a change request. Even more, the impact on all artifacts needs to be analyzed and documented.

Table 1 presents for every phase what XP practices can be applied. In addition, we discuss the most important agile opportunities for every phase. Together with the risks and weaknesses they define a new curve for an agile DO-178B driven process.

5.1 Software phase

Software development in this phase is not yet affected by other issues or constraints - the software is developed independently. In this phase, all agile practices may be used. Requirements changes - even hardware changes - are welcomed. User and acceptance testing can be fully automated within the context of the software.

5.2 Embedded phase

In this phase, there are no fundamental reasons to abandon agility. However, there may be some repercussions on activities that depend on the developed software as input. Example activities include continuously installing the software on the target hardware, retesting the hardware and generating documentation. The opportunities in this phase mainly consist of automating these tasks. Also, feedback and communication become more important in this phase, in order to cope with the dependencies between and coordination of software and other activities. Agile practices to consider for facilitating this are post-iteration workshops and project retrospectives.

5.3 Certification phase

This phase brings with it many additional tasks that need to be executed upon each software change. Traceability needs to be established and manual testing and reviewing is required.

A logical measure here is to limit the amount of changes. First, to keep the requirements changes to a minimum, the customer can write their own acceptance tests. Regarding traceability, there is an opportunity to handle and manage documents more as source code, so that agile code-centric practices can also be applied to them. In particular, one can apply the following practices:

- auto-generate not only code, but as much documents as possible;
- include all documents in a version control system;
- manage their dependencies, so that it is immediately clear what document parts are affected by an artifact change.

For documents that cannot be auto-generated, an agile document preparation practice may be useful, such as RaPiD7 [5]. In RaPiD7, documents are made in workshops where multiple stake-holders are present. Reportedly, it speeds up the document development process significantly (with speedups varying between 15 and 96%).

For independent reviewing, pair programming may offer a solution. In a way, a paired programmer continuously reviews the other person's work. Frequent changes in pairs should guarantee independence, as people get to see a lot of other people's mistakes and gradually become expert reviewers.

The development of automatic test suites is an intensive task. Auto-generation of test code will speed-up the testing process considerably. For manual acceptance testing, there is an opportunity to automate some tests, although this may need special hardware. One would have to operate inputs (such as the control panel of a flight display) and capture the output of the system (such as the pixel values of a display).

6 Weaknesses and risks

With the agile opportunities of the previous section, we considerably flatten the steep curve of a regular DO-178B driven process, as Figure 1 shows. However, a software process is as slow as its weakest link.

Agility relies on coping with complexity, and most agile practices focus on software complexity. For the software itself, this benefits the project up to the certification phase, because once software gets installed in production type aircraft, it does not need to be updated that often.

As a project progresses, software changes create complexity that is not handled by agile software practices. Managing traceability, even with requirements tools (such as Telelogic DOORS) may remain difficult. It may not be possible to automatically generate certain written documents. The earlier mentioned automation using agile tools is has much less value if the results of uncertified tools (for example, test suites) need to be manually verified.

At a certain point in time, reducing complexity of the software may even cause greater complexity, because of the ramifications on traceability, documentation and testing. That is why practices such as refactoring are discouraged in the certification phase.

To summarize, although we expect significant speedups by applying the agile opportunities, a daily integrated system build process will most likely be unfeasible once these external factors come into the picture. Hence, the principle “Requirements changes are welcomed” will be hard to maintain in the certification phase, if at all .

This defines the Agile DO-178B curve in Figure 1. It is not flat, since a sustainable development pace will remain a hard to reach ideal for the avionics domain. Still, great improvements can be made compared to the traditional way of handling a DO-178B driven process.

7 Conclusions and future work

This analysis confirms that while a process such as XP can be applied to many domains, it is targeted at software development processes that are not hindered by or dependent on factors external to the software. While general statements cannot be made based on this single assessment, it seems that most agile principles are still valid in the avionics world. In addition, although avionics software development is clearly dominated by the plans and documents that go with it, there is room to apply agile practices.

However, because of the large certification overhead, it will not be possible to “flatten” the Boehm curve [3] as XP claims.

This said, our most important observations for improvement are these:

Customer level: communicate regularly and early in the development process and deliver incrementally functional flight-worthy prototypes. This will reduce the requirements changes later in the project, when they are more difficult to apply.

Team and project level: add more transparency and feedback to the project by applying project feedback based practices, such as post-iteration workshops and project retrospectives.

Technical level: treat documents like source code and apply continuous integration, ultimately enabling shorter iterations.

In the future, this analysis will be further discussed with a number of stakeholders. Together, we hope to further concretize the risks and utility of the agile practices, select the best practices, and apply them in (sub)projects.

To conclude, we state that as the pressure for iterative and customer driven software development will further increase, the industry has no choice but to adapt their processes accordingly. Not only the customer has to accept new responsibilities for an agile approach to work. Certification authorities will need to acknowledge that agile software development can yield software that is at least as safe as before. However, it remains the developer's task to convince the authorities of this. We can only guess the timeframe of these changes. As it took some time for the certification authorities in order to accept certain object-oriented development techniques for avionics software, we expect that agile practices will soon also be recognized by the certification authorities as useful practices within an avionics software development process.

Acknowledgments

We would like to thank the people from Barco for their time and feedback, especially Stijn Rammeloo for his active participation in this assessment and for his insights on the avionics software domain.

References

1. K. Beck and et al. Manifesto for agile software development, 2001.
2. B. Boehm and R. Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, 2003.
3. B. W. Boehm. *Software Engineering Economics*. Prentice-Hall Advances in Computing Science & Technology Series, 1981.
4. A. Cockburn. *Agile Software Development*. Addison-Wesley Professional, 2001.
5. R. Kymälkoski. Efficient authoring of software documentation using RaPiD7. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 255–261, Washington, DC, USA, 2003. IEEE Computer Society.
6. P. Manhart and K. Schneider. Breaking the ice for agile development of embedded software. In *Proceedings of the 26th international conference on software engineering (ICSE)*, 2004.
7. RTCA. DO-178B: Software considerations in airborne systems and equipment certification, 1992.

Practice	Description	Software phase	Embedded phase	Certification phase	Comments and risks
On-site customer	Customer is present and available for the team to make questions etc.	×	×	×	Commonly, avionics development starts off with a detailed requirements document from the customer. This practice could reduce the customer start-up effort and move back the start date for a project to create more room for development and certification.
Metaphor	Simple story of the purpose of the application.	-	-	-	A metaphor is not really necessary as the domain applications are very similar.
Short releases	The product is done in iterative style and new versions are “published” rapidly.	×	×	-	This is a necessary practice, but difficult to maintain towards the end of the project.
Planning game	The way for customer and the team to plan and communicate which tasks are to be implemented in each iterations.	×	×	×	As the project advances, this practice becomes essential to limit change.
Pair programming	Coding is done in pairs using one computer.	×	×	×	This practice could help comply with the DO-178B standard, because the latter mandates that all code should be proof-read by a separate person.
Collective code ownership	No one owns the code and everybody is allowed to change any parts of the code.	×	×	×	
Unit testing	Unit tests are written before the actual code.	×	×	×	
Acceptance testing	Customer writes the acceptance tests	×	×	×	These tests could seriously reduce further requirements changes. Problems arise when testing high level requirements, as the DO standard states the necessity for these to be verified by a human being.

Table 1. XP practices (continued on next page)

Practice	Description	Software phase	Embedded phase	Certification phase	Comments and risks
Refactoring	Remove duplication and add simplicity.	×	×	-	This practice is not recommended late in the development process: refactoring after certification procedures would add weeks to the certification cycle.
Simple design	Tasks are solved with the simplest possible way to avoid unnecessary complexity.	×	×	×	Simple design could improve the testing cycle and reduce low level requirements.
Continuous integration	New code is integrated as soon as it is ready.	×	×	×	Recommended, as this finds bugs early.
Coding standards	Coding rules that everybody follows.	×	×	×	This is necessary for DO-178B certification.
40-hour-week	Avoiding working overtime.	×	×	×	This is mainly useful in conjunction with pair programming.