SEVENTH FRAMEWORK PROGRAMME

Collaborative Large-scale Integrating Project

OPENCOSS

Open Platform for EvolutioNary Certification Of Safety-critical Systems

# Methodology: Agile development of safety critical systems
# Annex D1.1.d to deliverable D1.1

| | |
|---|---|
| **Work Package:** | WP1: Use case Specification and Benchmark |
| **Dissemination level:** | PU = Public |
| **Status:** | FINAL |
| **Date:** | 28 March 2012 |
| **Responsible partner:** | J. Lambourg (AdaCore) |
| **Contact information:** | lambourg@adacore.com |

## Contributors

| Names | Organisation |
|---|---|
| Jérôme Lambourg, Cyrille Comar | AdaCore |

## Document History

| Version | Date | Remarks |
|---|---|---|
| V0.1 | 2012-03-19 | First emission (AdaCore) |
| V0.2 | 2012-03-20 | Ready for PB Approval |
| V1.0 | 2012-03-28 | Approved by PB |

# TABLE OF CONTENTS

**1 ABOUT AGILE DEVELOPMENT ..... 4**

**2 AGILE DEVELOPMENT OF SAFETY CRITYCAL SOFTWARE, DO-178B CONTEXT ..... 5**

2.1 THE DO-178B ..... 5
2.2 APPLYING AGILITY TO DO-178B CERTIFICATION ..... 5
2.3 ADDITIONS TO AGILITY FOR CERTIFICATION CONSTRAINTS: ..... 6

**3 EXAMPLES OF ADDITIONAL REQUIREMENTS FOR AGILE METHODS ..... 7**

# 1 About Agile development

Agile development is a group of development methodologies based on iterative and incremental development. They can be applied to many kinds of projects, but are currently mainly applied to Information technology.

The notion of “agile methodology” has been officially described in 2001 in the Agile Manifesto (http://agilemanifesto.org), presenting the common Values of already existing methodologies. Those methods usually cover two development aspects:

- The development life cycle aspect: iterative, incremental and adaptive.
- Some common development techniques.

Some examples of those methods are:

- RAD (*Rapid Application Development*, James Martin, 1991)
- Scrum (*The New Product Development Game*, Takeuchi and Nonaka, 1996)
- Extreme programming or XP (*Extreme Programming Explained*, Kent Beck, 1999)

Extreme programming defines simple, yet interdependent organizational and engineering best practices, taken to an extreme.

For example, code reviews are a best practice, so XP recommends Pair Programming: all production code is written by a pair of programmers working together at the same workstation. One writes the code while the other reviews it as it’s typed in. The two programmers switch role frequently.

Another example concerns testing, where XP recommends Test Driven Development. It consists of short cycles based on the following steps: from the requirement to implement, write automated self-checking tests that define the desired changes or new functionality; write just enough code to make the test succeed, and then refactor the code to improve the design. This is used for both Unit testing and customer acceptance testing.

XP’s main principles are:

- Code review
- Test Driven development
- Refactoring: without changing the behavior of the software, its design is improved continuously and the code refactored.
- Simple Design: as a general principle, the simplest working design is always implemented.
- Continuous Integration: to lower integration risks, the team should constantly have a shippable version of the product including all the latest changes. Programmers always work on the latest version of the software. Therefore they check in their code and integrate several times per day.
- Small Releases (short release life cycle)
- Whole Teams, improving collaboration and communication between the users and the developers.

# 2 Agile development of Safety Critycal software, DO-178B context

## 2.1 The DO-178B

In the avionics domain, software development is usually performed by following the DO-178B/C standards. As input to this standard is the DAL (Design Assurance Level). The DAL defines the level of criticality given to avionics components according to the impact of a failure of that component on the flight's safety. The level range from "A. Catastrophic: may cause a crash" to "E. No Effect".

The DO-178B document, published in 1992, as well as its successor the DO-178C document plus its annexes, published in 2011, specifies the objectives to achieve during development in order to certify software for flight. The certification of the software is obtained after a series of audits, inspecting the proofs of the activities carried out to support these objectives.

As the level of criticality increases, so does the number of objectives to satisfy.
The DO-178 documents define the objectives to perform according to the DAL, but not the process to achieve them. However the objectives concern the software life cycle, the activities that need to be carried out, their entry and exit criteria, and the expected verifications.

A certification audit checks that the product exclusively contains code satisfying operational requirements. The implementation of the operational requirements and all the code have to be verified. Additionally, all the products (plans, requirements, architecture, code, tests, traceability) have to be validated in a review process.

## 2.2 Applying Agility to DO-178B certification

The DO-178B or C documents do not specify any process to follow to reach the certification objectives. This makes it possible to apply Agile methodologies to a certification process. Most of the objectives of the standard can successfully by reached using such methodology, bringing the benefits of Agility to the avionics domain:

- Improve the visibility of the development process, by performing the full software development life cycle at each iteration, and by providing many advancement metrics;
- Lower the risk by performing the heavy tasks such as low level testing at each iteration;
- Improve the development efficiency, by constantly improving the design, architecture, and the process itself.
- Improve the overall quality via continuous integration, giving quick feedback to the developer allowing him to take immediate corrective actions when needed, and removing the human error factor inherent to repetitive tasks (which is typically the case for test suties).

In addition to those project management benefits, Emmanuel Chenu, in his article "Agility and Lean for Avionics" (2009, http://manu40k.free.fr/AgilityAndLeanForAvionics1.pdf) even states that such use of Agile methods contributes to achieve safety objectives:

The incremental construction driven by operational scenarios enforces that the product will exclusively contain code implementing operational requirements, which is mandated by the requirement-based testing strategy of the DO-178, and the objective to fully cover the source code by tests (chapter 6.4.4 of the DO-178B).

Systematic acceptance testing will also guarantee that the implementation of all operational requirements is successful, resolving issues as they appear (chapter 6.4.2 and 6.4.3 of the DO-178B).

Systematic developer testing will ensure that all the code is checked by tests, leading to full code coverage by tests at all time. Achieving full coverage testing at the MC/DC level is one of the most difficult result to achieve when developing a Level A software.

The continuous software integration allows to automatically run the full acceptance and developer tests, ensuring at a minimum cost that the latest version of the program is always fully and repeatedly tested. Integrating the documentation into this continuous software integration

## 2.3 Additions to Agility for certification constraints:

However, the agile methods do not address all certification constraints, and thus need to be adapted. For example, the documentation is generally not seen as mandatory in the context of Agile methods, while it has to be considered part of the product in the case of avionics certification. Therefore, it has to be incrementally written to be potentially shippable after each iteration.

Also, traceability is not addressed by Agile methods : as the product under development is constantly evolving, the architecture being modified through iterative and incremental development with continuous refactoring, the traceability between requirements and the software is difficult to address and requires also a constant update.

# 3 Examples of additional requirements for Agile methods

The Agile methods requirements will have an impact mainly on the process one project needs to follow. From this process some additional constraints may be derived on the tools used (such as a continuous integration tool), but they are still related at their core to the Agile processes.

Following are proposed requirements on the process to apply agile methods to a safety critical product. They're meant to be example, but are not meant to be complete.

| **<ADA> 0001: <Iterations>** | |
|---|---|
| **Alias** | Process iteration |
| **Status** | Proposed |
| **App. Domain** | All |
| **Type** | Process |
| **Priority** | High |
| **Description** | The process followed during the product development should be iterative. An iteration lasts typically a few weeks and concerns a particular feature to add to the product. A complete life cycle is performed during each iteration |
| **Derived from** | |

| **<ADA> 0002: <Whole Team>** | |
|---|---|
| **Alias** | Whole team principle |
| **Status** | Proposed |
| **App. Domain** | All |
| **Type** | Process |
| **Priority** | High |
| **Description** | The Whole team principle places the customer as part of the team. In safety-critical systems, this implies that all external stakeholders (system level team, certification body, customer) should be deeply involved in the development process, on the features priorities, and have a strong visibility on the product under development. |
| **Derived from** | |

| **<ADA> 0003: <Certifiable state>** | |
|---|---|
| **Alias** | Certifiable state |
| **Status** | Proposed |
| **App. Domain** | All |
| **Type** | Process |
| **Priority** | High |
| **Description** | In order to increase the visibility on the product under development, the project needs to be in a 'certifiable' state after each iteration: the product should be able to run, the documentation updated, and all verifications are performed. |
| **Derived from** | |

| **<ADA> 0004: <Testing>** | |
|---|---|
| **Alias** | Test driven development |
| **Status** | Proposed |
| **App. Domain** | All |
| **Type** | Process |
| **Priority** | High |

| | |
|---|---|
| **Description** | At each iteration, a test should be written first, directly derived from the operational requirement(s) to implement. This test should be automated. |
| **Derived from** | |

| **<ADA> 0005: <Refactoring>** | |
|---|---|
| **Alias** | Refactoring |
| **Status** | Proposed |
| **App. Domain** | All |
| **Type** | Process |
| **Priority** | High |
| **Description** | At each iteration, after the code is written to implement the functionality, some refactoring should be performed to simplify the design of the software. |
| **Derived from** | |

| **<ADA> 0006: <Automation>** | |
|---|---|
| **Alias** | Automation |
| **Status** | Proposed |
| **App. Domain** | All |
| **Type** | Process |
| **Priority** | High |
| **Description** | To ease the iterations, every activity that can be automated should be automated. This increases the developers efficiency, and increases the overall product quality. |
| **Derived from** | |

| **<ADA> 0007: <Documentation>** | |
|---|---|
| **Alias** | Documentation Automation |
| **Status** | Proposed |
| **App. Domain** | All |
| **Type** | Process |
| **Priority** | High |
| **Description** | In Safety critical systems, the documentation is part of the product to deliver. It thus should evolve along with the project (iteratively), and be built automatically with it when needed (test results for example). |
| **Derived from** | |

| **<ADA> 0008: <Traceability>** | |
|---|---|
| **Alias** | Traceability |
| **Status** | Proposed |
| **App. Domain** | All |
| **Type** | Process |
| **Priority** | High |
| **Description** | In Safety Critical Systems, traceability between artefacts needs to be demonstrated. It is particularly important to maintain traceability matrix when using Agile methods, where refactoring happen very often. |
| **Derived from** | |