



BSI Standards Publication

Space engineering — Agile software development handbook

National foreword

This Published Document is the UK implementation of CEN/TR 17603-40-01:2022.

The UK participation in its preparation was entrusted to Technical Committee ACE/68, Space systems and operations.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

This publication is not to be regarded as a British Standard.

© The British Standards Institution 2022
Published by BSI Standards Limited 2022

ISBN 978 0 539 20510 7

ICS 35.080; 49.140

Compliance with a Published Document cannot confer immunity from legal obligations.

This Published Document was published under the authority of the Standards Policy and Strategy Committee on 31 August 2022.

Amendments/corrigenda issued since publication

Date	Text affected

English version

Space engineering - Agile software development handbook

Ingénierie spatiale - Guide de développement logiciel
en mode agile

Raumfahrttechnik - Handbuch zur agilen
Softwareentwicklung

This Technical Report was approved by CEN on 20 April 2022. It has been drawn up by the Technical Committee CEN/CLC/JTC 5.

CEN and CENELEC members are the national standards bodies and national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and United Kingdom.



CEN-CENELEC Management Centre:
Rue de la Science 23, B-1040 Brussels

Table of contents

European Foreword.....	7
Introduction.....	8
1 Scope.....	9
2 References	10
3 Terms, definitions and abbreviated terms.....	11
3.1 Terms from other documents.....	11
3.2 Terms specific to the present document	11
3.3 Abbreviated terms.....	15
4 Introduction to the Agile software development approach	17
4.1 Introduction to Agile.....	17
4.1.1 General	17
4.1.2 Agile characteristics (as derived from the manifesto)	18
4.1.3 Lean management	20
4.2 General issues implementing Agile.....	21
5 Guidelines for Agile life cycle selection	24
5.1 Selecting Agile.....	24
5.2 Analysis of key factors for Agile selection	24
5.2.1 General	24
5.2.2 Customer context.....	26
5.2.3 Supplier context	27
5.2.4 Project context	27
5.2.5 Team context	29
5.2.6 Key Factors Summary.....	30
5.3 Agile assessment process	31
5.4 Selecting agile or waterfall	32
6 Reference models for Scrum-like Agile software life cycle	34
6.1 Introduction.....	34
6.2 Roles and competences	34

6.2.1	Overview.....	34
6.2.2	Scrum master.....	34
6.2.3	Product owner.....	35
6.2.4	Development team.....	35
6.2.5	SCRUM team.....	36
6.2.6	Agile coach.....	36
6.2.7	Training and competencies.....	36
6.3	Exemplary Agile activities.....	37
6.3.1	Distinction between meeting or activity.....	37
6.3.2	Planning I – What will be delivered.....	38
6.3.3	Planning II – How will it be delivered.....	38
6.3.4	Sprint backlog management.....	39
6.3.5	Product backlog refinement.....	39
6.3.6	Progress tracking.....	40
6.3.7	Product backlog update.....	40
6.3.8	Coding, testing and documenting.....	40
6.3.9	User feedback.....	41
6.3.10	Review preparation.....	41
6.3.11	Sprint review.....	41
6.4	Meetings.....	42
6.4.1	Daily meeting.....	42
6.4.2	Management meeting.....	42
6.4.3	Retrospective.....	42
6.5	Organising the Agile activities and meetings in a project to create a life-cycle compliant to ECSS-E-ST-E-40.....	43
6.5.1	Preliminaries.....	43
6.5.2	Product releases.....	44
6.5.3	Start of the project: Sprint#0.....	44
6.5.4	Development phase: Sprints #1 - #N.....	45
6.5.5	Acceptance phase.....	45
6.6	Software lifecycle definition.....	46
6.6.1	ECSS-E-ST-40 reviews.....	46
6.6.2	Organising the ECSS-E-ST-40 reviews in an Agile software approach.....	47
6.6.3	Selecting the right model.....	56
7	Guidelines for software project management.....	57
7.1	Introduction.....	57
7.2	Software Project Management approach.....	57

7.2.1	Overview	57
7.2.2	Management objectives and priorities	57
7.2.3	Schedule management	61
7.2.4	Assumptions, dependencies and constraints.....	63
7.2.5	Work breakdown structure	64
7.2.6	Roles.....	64
7.2.7	Risk management	65
7.2.8	Monitoring and controlling mechanisms	66
7.2.9	Staffing Plan.....	70
7.2.10	Software procurement process.....	72
7.2.11	Supplier management	72
7.3	Software development approach	73
7.3.1	Strategy to the software development	73
7.3.2	Software project development lifecycle	73
7.3.3	Relationship with the system development lifecycle	73
7.3.4	Reviews and milestones identification and associated documentation	73
7.4	Software engineering standards and techniques	73
7.5	Software development and software testing environment	73
7.6	Software documentation plan	74
8	Guidelines for software engineering processes	75
8.1	Overview	75
8.2	Software related system requirements process	75
8.3	requirements and architectural engineering	75
8.3.1	Software requirements analysis	75
8.3.2	Software architectural design	81
8.4	Software design and implementation engineering.....	83
8.5	Software validation	85
8.6	Software delivery and acceptance	86
8.7	Software verification	88
8.8	Software operations.....	91
8.9	Software maintenance	91
8.9.1	Overview.....	91
8.9.2	Agile maintenance challenges.....	91
8.9.3	Tailoring Agile to Maintenance	92
8.10	Independent software verification and validation.....	95
9	Guidelines for software product assurance and configuration management	96

9.1	Software product assurance	96
9.1.1	Introduction	96
9.1.2	Planning of software product assurance activities	97
9.1.3	Software product assurance reporting	97
9.1.4	Technical Debt and noncompliance of Quality Requirements	98
9.1.5	Software criticality	99
9.1.6	Software problem management	99
9.1.7	Control of non-conformances	100
9.1.8	Software development environment aspects	100
9.1.9	Summary of software product assurance activities in Agile	100
9.2	Software configuration management	102
9.2.1	Introduction	102
9.2.2	Agile software configuration management challenges	102
9.2.3	Agile methods for configuration management	104
9.2.4	Summary of software configuration activities in Agile	105

Figures

Figure 4-1: From Plan-driven approach to Value-driven approach	19
Figure 4-2: The Lean Thinking House (for details see LEAN-PRIMER)	21
Figure 5-1: Factors for adopting Agile process	25
Figure 5-2: Agile selection factors scale	26
Figure 6-1: Organisation of activities during a sprint	37
Figure 6-2: Exemplar Agile lifecycle	43
Figure 6-3: Model 1: Review driven lifecycle	51
Figure 6-4 Model 2: More flexible review driven lifecycle	53
Figure 6-5: Review driven lifecycle with full flexibility	54
Figure 6-6: Sprint driven lifecycle with formalisation	55
Figure 7-1: Project Management Triangle	58
Figure 7-2: Cost Management: change for free	59
Figure 7-3: Sample Burndown Chart for a Sprint	62
Figure 7-4: Example for an Agile work breakdown	64
Figure 7-5: Agile model supports risk management	65
Figure 7-6: Success of continuous integration tests	68
Figure 7-7: A team metric dashboard	68
Figure 7-8: Summary of performed work	69
Figure 7-9: Delivered business value in a project	70
Figure 8-1: Example of User Story and Tasks	77

Figure 8-2: Kano model showing means to ensure customer satisfaction.....	79
---	----

Tables

Table 5-1: Supplier context.....	27
Table 5-2: Project context.....	28
Table 5-3: Team context.....	29
Table 5-4: Key factors for selection of classical or agile lifecycle	30
Table 5-5: Aspects for the selection of agile or waterfall approach	32
Table 6-1 – Overview of the different models	49
Table 6-2 – Examples of selection of models based on project characteristics	56
Table 8-1: Mapping ECSS-E-ST-40 to Agile activities. Software Requirements Analysis	80
Table 8-2: Mapping ECSS-E-ST-40 to Agile activities. Software architectural design.....	82
Table 8-3: Mapping ECSS-E-ST-40 to Agile activities. Software Detailed Design, Coding and testing, and Integration.....	83
Table 8-4: Mapping ECSS-E-ST-40 to Agile activities. Software validation.....	85
Table 8-5: Mapping ECSS-E-ST-40 to Agile activities. Software Delivery and Acceptance	87
Table 8-6 Mapping ECSS-E-ST-40 to Agile activities. Software Verification	89
Table 8-7: Mapping ECSS-E-ST-40 to Agile activities. Software Maintenance	94
Table 9-1: Mapping ECSS-Q-ST-80 to Agile activities. Software product assurance	100
Table 9-2: Mapping ECSS-M-ST-40 to Agile activities. Software Configuration Management	105

European Foreword

This document (CEN/TR 17603-40-01:2022) has been prepared by Technical Committee CEN/CLC/JTC 5 "Space", the secretariat of which is held by DIN.

It is highlighted that this technical report does not contain any requirement but only collection of data or descriptions and guidelines about how to organize and perform the work in support of EN 16603-40.

This Technical report (CEN/TR 17603-40-01:2022) originates from ECSS-E-HB-40-01A.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN [and/or CENELEC] shall not be held responsible for identifying any or all such patent rights.

This document has been prepared under a mandate given to CEN by the European Commission and the European Free Trade Association.

This document has been developed to cover specifically space systems and has therefore precedence over any TR covering the same scope but with a wider domain of applicability (e.g.: aerospace).

Introduction

EN 16603-40 (ECSS-E-ST-40) Space Engineering Software Standard defines the principles and requirements applicable to space software engineering. ECSS-E-ST-40 is always complemented by the EN 16602-80 (ECSS-Q-ST-80) Space Product Assurance Standard, which specifies the product assurance aspects. This ECSS-E-HB-40-01 handbook provides more detailed guidelines and advice for adopting an Agile software development approach in space projects where ECSS-E-ST-40 and ECSS-Q-ST-80 are applicable.

1

Scope

This Handbook provides recommendations for the implementation of an Agile approach in space software projects complying with EN 16603-40 (ECSS-E-ST-40) and EN 16602-80 (ECSS-Q-ST-80).

This handbook is not an Agile development book, though it provides an Agile reference model based on Scrum and also covers other major Agile methods and techniques. Scrum has been selected as reference because of its widespread application in industry and its flexibility as a development framework to introduce or merge with other Agile methods and techniques. In relation to the ECSS-E-ST-40 and ECSS-Q-ST-80, this handbook does not provide any tailoring of their requirements due to the use of the Agile approach, but demonstrates how compliance towards ECSS can be achieved. This handbook does not cover contractual aspects for this particular engineering approach, although it recognises that considering the approach of fixing cost and schedule and making the scope of functionalities variable, the customer and supplier need to establish specific contractual arrangements. Furthermore, it does not impose a particular finality for the use of Agile, either as a set of team values, project management process, specific techniques or supporting exploration by prototypes.

This handbook, covers, in particular, the following:

- In clause 4, the fundamentals and principles of Agile. It also describes major Agile methods and general issues of implementing an Agile approach.
- In clause 5, the criteria for selecting an Agile lifecycle.
- In clause 6, a reference process model based on Scrum to be used to map its elements to relevant clauses of ECSS-E-ST-40.
- In clause 7, guidelines for software project management, providing advice for ECSS-E-ST-40 clause 5.3 considering the reference process model based on Scrum.
- In clause 8, guidelines for software engineering processes, providing advice for ECSS-E-ST-40 clauses 5.2, and 5.4 to 5.10, considering the reference process model based on Scrum.
- In clause 9, guidelines for software product assurance and software configuration management, providing general advice for the implementation of ECSS-Q-ST-80 and ECSS-M-ST-40 with an Agile approach.

Individual agile practices, introduced in this HB, can also be taken on-board in other software development life-cycles.

2 References

EN Reference	Reference in text	Title
EN 16601-00-01	ECSS-S-ST-00-01	ECSS system - Glossary of terms
EN 16603-40	ECSS-E-ST-40	Space engineering - Software
EN 17603-40	ECSS-E-HB-40	Space engineering - Software engineering handbook
EN 16601-10	ECSS-M-ST-10	Space project management - Project planning and implementation
EN 16601-40	ECSS-M-ST-40	Space project management - Configuration and information management
EN 16601-80	ECSS-M-ST-80	Space project management - Risk management
EN 16602-80	ECSS-Q-ST-80	Space product assurance - Software product assurance
EN 16601-80-04	ECSS-Q-HB-80-04	Space product assurance - Software metrication programme definition and implementation handbook
	Agile Manifesto	Beck, K., et al.: Agile Manifesto and Twelve Principles of Agile Software (2001). http://agilemanifesto.org
	ISO/IEC 26515:2011	Systems and software engineering - Developing user documentation in an Agile environment
	LEAN-PRIMER	Craig Larman and Bas Vodde. 2009. Lean Primer. Available at: http://www.leanprimer.com/downloads/lean_primer.pdf
	Agilealliance	https://www.agilealliance.org
	SCRUM	https://www.scrum.org

Terms, definitions and abbreviated terms

3.1 Terms from other documents

- a. For the purpose of this document, the terms and definitions from ECSS-S-ST-00-01 apply, in particular the following terms:
 1. **process**
 2. **product**
- b. For the purpose of this document, the terms and definitions from ECSS-E-ST-40 apply, in particular the following term:
 1. **critical software**

3.2 Terms specific to the present document

3.2.1 Burndown chart

A chart that records project status, usually showing tasks completed versus time and against total number of tasks

[ISO/IEC 26515:2011]

3.2.2 Capacity

Total number of available hours for a sprint. The capacity is the available hours calculated based on resources planned holiday and company holiday if any.

3.2.3 Continuous integration

development practice according to which the source code in development is uploaded into a shared repository regularly to allow automated build, tests and quality control tools to detect and raise issues early to the development team

3.2.4 Cycle time

time elapsed between the start of the work on a particular task and its completion.

3.2.5 Daily stand-up

short daily team meeting where the team members share their project activities, synchronize themselves and identify and solve impediments that hamper them from being productive

NOTE 1 It is also known as daily meeting, daily Scrum or roll-call.

NOTE 2 Duration of a “Daily stand-up” is about 15 minutes.

3.2.6 Definition of done (DoD)

list of activities and criteria to be achieved to declare an element of the backlog as complete

NOTE 1 This element can be defined at user story, epic, feature, sprint and product release levels

NOTE 2 Elements of the DoD can be, for example: writing source code, update specification, design and user documentation, execution of unit testing, achieving a certain level of test coverage, establish compliance to a certain level of coding rules.

[adapted from the Agile Alliance]

3.2.7 Definition of ready

list of criteria that a user story has to meet to be accepted into the upcoming sprint or iteration

[adapted from the Agile Alliance]

3.2.8 Epic

big user story that is too big to be estimated. It is usually too big for an iteration and will be broken down into smaller user stories.

NOTE It is usually too big for an iteration and will be broken down into smaller user stories.

3.2.9 Feature

functional or non-functional distinguishing characteristic of a system, usually an enhancement to an existing system

[ISO/IEC 26515:2011]

3.2.10 Increment

sum of all the Product Backlog items completed during a sprint and the value of the increments of all previous sprints

3.2.11 Iteration

See “sprint”[3.2.22].

3.2.12 Lead time

time elapsed between the customer request for an increment and its delivery

3.2.13 Load factor

KPI measuring how many person days it takes to complete a story point

NOTE A story point is a relative measurement unit used to compare relatively user stories, epics (e.g. one ideal working day).

3.2.14 Pair programming

practice where two developers share the same development environment, where the first developer writes the code and the second reviews the code simultaneously and thinks ahead

NOTE Both change roles often so that the strong points of both developers can be utilised and knowledge is shared in the team.

3.2.15 Peer review

practice where code written by a developer is reviewed by another developer of the same team

3.2.16 Product backlog

ordered list of everything that is known to be needed in the product

- NOTE 1 It is the single source of requirements for any changes to be made to the product.
- NOTE 2 A Product Backlog is never complete. The earliest development of it lays out the initially known and best-understood requirements. The Product Backlog evolves as the product and the environment in which it will be used evolves. The Product Backlog is dynamic; it constantly changes to identify what the product needs to be appropriate, competitive, and useful. If a product exists, its Product Backlog also exists.
- NOTE 3 The items can be, for examples, features, requirements, software problem reports or technical tasks.
- NOTE 4 The backlog is the primary point of entry for knowledge about requirements, and the single authoritative source defining the work to be done.

[adapted from the Agile Alliance]

3.2.17 Product Owner

entity as close as possible to the end customer and users and as knowledgeable as possible to the business and solution context

- NOTE See also clause 6.2.3.

3.2.18 Product Backlog Refinement

formal or informal meeting or activity to refine the backlog

- NOTE 1 Examples of refinement are:
- removing user stories that no longer appear relevant,
 - creating new user stories in response to newly discovered needs,
 - re-assessing the relative priority of user stories, assigning estimates to user stories which have yet to receive one,
 - correcting estimates in light of newly discovered information, splitting user stories which are high priority but too coarse grained to fit in an upcoming iteration.
- NOTE 2 For detailed information about “Product backlog refinement” see clause 6.3.5.

[adapted from the Agile Alliance]

3.2.19 Sprint backlog

set of product backlog items selected for the sprint, plus a plan for delivering the product increment and realizing the sprint goal

3.2.20 Release plan

plan to deliver an increment to the product with a release estimated date and providing for each related sprint, an outline that specifies its content

3.2.21 Retrospective

meeting that is held at the end of an iteration with the objective of improving the process by reviewing what happened in the last iteration and identifying the good things done in it and what could have been done better in order to identify improvement actions for next iterations

NOTE The sprint retrospective occurs after the sprint review and prior to the next sprint planning.

3.2.22 Sprint

short time frame, in which a set of software features is developed, leading to a working product that can be demonstrated to stakeholders

NOTE Also known as iteration.

[ISO/IEC 26515:2011]

3.2.23 Sprint review

review that is held at the end of the sprint to inspect the increment and adapt the product backlog if needed

3.2.24 Stakeholder

entity who have a vested interest in the success of the project, but do not belong to the core team

3.2.25 Technical debt

metaphor for the efforts and costs that built up when code is developed without applying the optimal solution, therefore being more difficult to maintain over time

3.2.26 Test driven development

programming practice where test cases are specified before the software code is written

3.2.27 User story

story that consists of one or more sentences in the language of the end user that captures what a user does or needs to do as part of his or her job function.

NOTE It captures the 'who', 'what' and 'why' of a requirement in a simple, concise way, often limited in detail by what can be hand-written on a small paper notecard.

A good scheme for creating user stories and other backlog items is INVEST:

I Independent: The user story is self-contained, in a way that there is no inherent dependency on another user story

N Negotiable: User stories are not explicit contracts and should leave space for discussion.

V Valuable: A user story must deliver value to the stakeholders.

E Estimable: Possibility to estimate the size of a user story.

S Small: The size of User stories is such that it does not become impossible to plan/task/prioritize with a certain level of accuracy.

T Testable: The user story or its related description provides the necessary information to make test development possible.

INVEST is often used as part of the Definition of Ready for a user story.

3.2.28 Velocity

number of story points delivered/demo in a sprint. The velocity is the measure of the amount of work a team can tackle during a single sprint

The velocity is the measure of the amount of work a team can tackle during a single sprint. The velocity is calculated at the end of the sprint by adding up the points for all fully completed user stories

[adapted from SCRUM]

3.3 Abbreviated terms

For the purpose of this document, the abbreviated terms from ECSS-S-ST-00-01, ECSS-E-ST-40 and the following apply:

Abbreviation	Meaning
AR	acceptance review
CDR	critical design review
CM	configuration management
DDR	detailed design review
DoD	definition of done
DRD	document requirements definition
ECSS	European Cooperation for Space Standardization
FDD	feature driven development
HMI	human machine interface
ICD	Interface control document
IDD	ideal developer day
ISVV	independent software verification and validation
KPI	Key performance indicator
MoM	minutes of meeting
MoSCoW	Must Should Could Won't
N/A	not applicable
NCR	Non-conformance report
OBS	organisation breakdown structure
OBSW	on-board software
PBI	product backlog item
PBS	product breakdown structure
PDR	preliminary design review
QA	Quality assurance
QR	qualification review

Abbreviation	Meaning
RB	requirement baseline
SDD	Software design document
SDLC	software development lifecycle
SDP	software development plan
SLA	service level agreement
SPA	Software product assurance
SPR	Software problem report
SRS	software requirements specification
SW	software
SWRR	software requirements review
TDD	test driven development
TS	technical specification
WBS	work breakdown structure
XP	extreme programming

Introduction to the Agile software development approach

4.1 Introduction to Agile

4.1.1 General

Agile is an iterative, time-boxed approach to software development in which a software product is built through an evolutionary process characterised by early and frequent deliveries of product increments, intensive customer collaboration and adaptive planning and goals, with the aim of responding to change in a rapid and flexible manner.

Agile development is inspired by the Agile Manifesto, which identifies four basic values:

- **“Individuals and interactions** *over* Processes and tools
 - **Working software** *over* Comprehensive documentation
 - **Customer collaboration** *over* Contract negotiation
 - **Responding to change** *over* Following a plan”
- [Agile Manifesto]

Each statement asserts that what is in the sentence on the left side (in bold) is valued more than what is on the right side.

Behind the Agile Manifesto lie the twelve principles, which teams working in an Agile environment follow

1. *“Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
2. *Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.*
3. *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
4. *Business people and developers work together daily throughout the project.*
5. *Build projects around motivated individuals. Give them the environment and support their need, and trust them to get the job done.*
6. *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*

7. *Working software is the primary measure of progress.*
8. *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
9. *Continuous attention to technical excellence and good design enhances agility.*
10. *Simplicity--the art of maximizing the amount of work not done--is essential.*
11. *The best architectures, requirements, and designs emerge from self-organizing teams.*
12. *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly."*

[Agile Manifesto]

NOTE Clause 5 provides concrete models for different approaches towards adopting the change of requirements late in the lifecycle in compliance with ECSS-E-ST-40.

At the core of Agile software development lies the high quality of deliverables, as a result of continuous product and process improvement. Agile teams develop a working system where they optimise the way they communicate and organise their work in order to achieve the highest value. There is a strong link with Lean management as detailed in clause 4.1.3.

4.1.2 Agile characteristics (as derived from the manifesto)

Agile development methodologies heavily rely on customer collaboration and feedback and therefore offer the chance to examine the direction in which a project is heading while it is ongoing. Compared to waterfall methodologies, where product development depends on the initial requirements and testing is done at the end of the implementation, in an Agile context teams continuously review and improve requirements, design, implementation and testing throughout the life cycle. In waterfall models, by the time the project is finished it can occur that the needs of the business have changed – this fact is embraced in Agile, by providing a framework where the scope can vary and planning is adaptive.

There are different Agile methods or frameworks that can be applied, such as Scrum, Kanban, Extreme Programming, or Feature Driven Development. Irrespective of which method is chosen, all of them are *Agile*, hence highly adaptable to support the need of the project and company, and they have the following common characteristics:

- They are iterative and evolutionary. The software product is developed through repeated cycles in a rapid feedback loop between customer and technologists. The customer adaptively specifies the requirements for the next iteration based on the observations on the evolving software product.
- Analysis, design, coding and testing are continuous activities.
- The iteration length can be pre-determined (time-boxed) and the scope is fixed for each iteration. One of the main differences of Agile compared to other iterative methods is that the length of each iteration is much shorter: typically two to four weeks.
- They support adaptive planning. Agile methods are intended for situations that foresee changes during the development time and the final delivery. It can be possible that requirements change, technology changes and staff changes.
- They recognise the importance of people as the most influential factor of the project. The self-organized team plays an important role delivering working software as a primary measure of

success. Agile methods focus on the competence and motivation of the software engineers as well as the organisation, communication and management.

- They attempt to reduce resource-intensive intermediate artefacts. Resources are prioritised to produce working software from the very beginning, in some cases it can be deployed directly into operation.

Agile emphasizes the value of better dealing with justified or valuable change. Addressing the changes (e.g. to the requirements, design or implementation) is in effect a change to the scope of the project. Having three parameters: Scope, Schedule and Cost, the impact of the change can be either handled by impact on the cost and schedule or by adapting the scope accordingly. Agile lifecycle provides the methodology for handling the changes to the scope in a fixed cost and schedule frame (see Figure 4-1). The details of how changes to the schedule, cost and scope can be handled contractually are out of the scope of this Handbook.

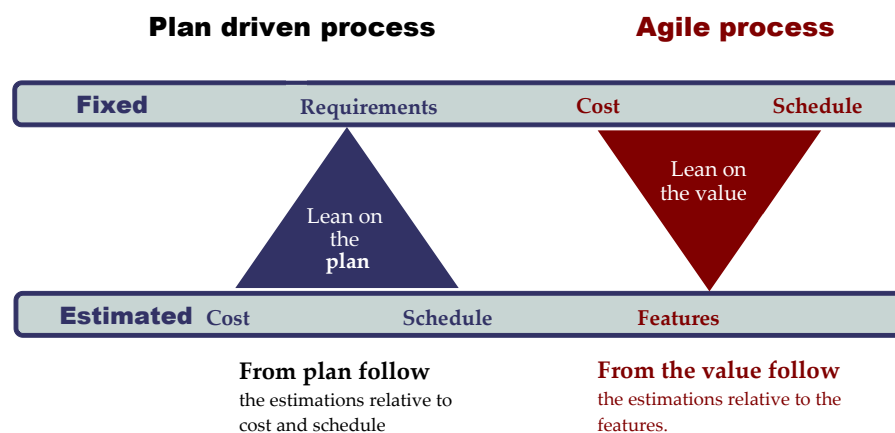


Figure 4-1: From Plan-driven approach to Value-driven approach

While defining what Agile is, it is equally important to emphasize what it is *not*. Agile is not:

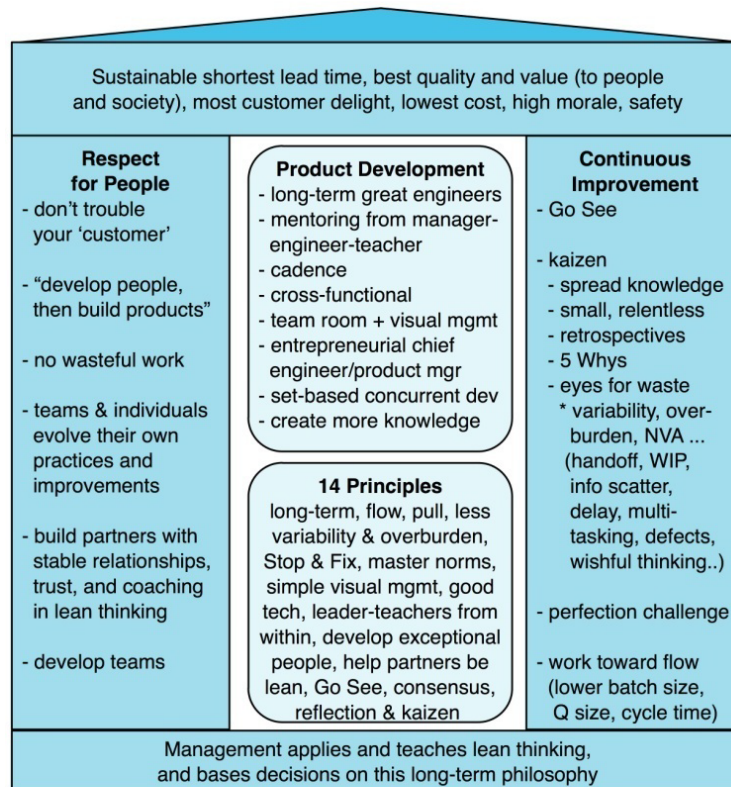
- A silver bullet. Agile cannot solve all development problems and the methods are adapted to the needs of the organisation.
- Anti-documentation. Documentation is just another valuable deliverable that can be estimated and prioritised like other user functionalities. Compliance to standards is also possible. An important note is that in an Agile context classical documents are converted into “living documents” (evolving documents) that are updated on an ongoing basis.
- Anti-planning. Agile methods involve a lot of planning (daily meetings, iteration and release planning meetings); therefore it is not advised to roll out an Agile project solely on the basis that it cannot be planned.
- Anti-architecture. An architecture needs to exist before starting the project, otherwise the lack of such an artefact indicates that the user needs are still unknown and further clarification is needed (i.e. the user needs are not clear). The architecture needs to be updated regularly as it evolves while the project is running.

4.1.3 Lean management

The core idea of Lean Management is to maximise customer value while minimising waste, where waste is any unprofitable action. In simple terms, a lean approach aims at creating more value for customers with fewer resources. An organisation that follows the lean principles understands customer value and focuses its key processes to continuously increase it. The ultimate goal is to provide perfect value to the customer through a perfect value creation process that has zero waste. The idea of providing customer value relates directly to the Agile principles.

Figure 4-2 shows the principles of Lean, also applicable to Agile. The layout follows the major structure of a house and is as follows:

- The roof defines the goal of a project, i.e. to quickly deliver things of value, while still achieving the highest quality. This can include focusing on low cost approaches, safety issues, and delighting the customer.
- The foundation (bottom) relies on management that thinks, applies, and teaches lean thinking. This includes implementing long-term philosophies and basing decisions on this.
- The left pillar is on respecting people by showing lean leadership. This includes being open by providing information and showing transparency, being fair, speaking feelings, telling the truth, showing consistency, and keeping commitments.
- The right pillar is on continuous improvement, articulated by applying retrospectives for teams, visual management, and working towards flow. The figure contains more techniques like Go See, where the responsible directly go where a problem occurred to see the problem and potential corrective actions. Kaizen is also shown and describes continuously improving development processes and projects. This can be done with several means like asking 5 times why, looking for non-value adding (NVA) activities, limiting work in progress (WIP), removing multitasking, and eliminating wishful thinking.
- The middle consists of product development and 14 lean principles. Product development lists that great engineers and mentoring between junior and senior engineers are needed. Teams shall be cross-functional and use visual management techniques. Set-based concurrent engineering is a means towards agile ways of working. The 14 principles include Agile-like principles like deciding as late as possible, the creation of flow, the idea of pull, and knowledge management.



*Summary of the Toyota Way (Lean Thinking) House
by Craig Larman and Bas Vodde. 2009*

Figure 4-2: The Lean Thinking House (for details see LEAN-PRIMER)

4.2 General issues implementing Agile

One of the objectives of this handbook is to highlight the challenging points of implementing a project that follows an Agile software development process and to provide in this context best practices to common questions that can arise.

The following list comprises a number of risks, issues, perceptions and concerns with regards to the application of Agile which are addressed in this handbook.

- a. Control over budget and schedule:
 - Customer concerns:
 - How do we ensure that at the end of the project we do not get less than originally expected due to the changes introduced as part of the Agile process (avoid running out of effort before all required features are implemented)?
 - How do we ensure that certain functionality is available at pre-defined milestones and the schedule does not slip due to the frequent changes as part of the Agile process?
 - Supplier concerns:
 - Does Agile mean that we, as suppliers, have to implement much more than originally planned within the same fixed effort and schedule?
 - Agile does not mean that the overall project schedule is flexible and extendable.
 - When team members, who are each an expert of a particular aspect of the system, work in parallel on multiple projects, how can a long term planning be achieved if the team members are reassigned to different Agile projects due to the introduced changes?

- b. Scope of documentation:
 - Customer Concerns:
 - When the value of the documentation is as high as the value of the working software, adopting Agile processes does not automatically mean “less documentation”.
 - Supplier Concerns:
 - How do we avoid that we are asked to follow both sequential and iterative Agile processes at the same time and produce exhaustive documentation according to a sequential process logic (at the corresponding milestones) while at the same time following an iterative Agile life cycle for the development?
- c. Approach towards Quality, Validation and Verification:
 - Customer Concerns:
 - There is a common perception that Agile works fine for prototype software but not for operational software such as critical on-board software. Can our suppliers use Agile and still deliver critical software?
 - There is a concern that in the end, adopting Agile can mean getting software, which has not been fully qualified (validation and verification) due to lack of time.
 - Supplier concerns:
 - If we follow an Agile approach while adhering to ECSS-E-ST-40 and ECSS-Q-ST-80, do we expect that each sprint is treated as a formal review, where each software release goes through the heavy validation and qualification process?
 - How do we produce the artefacts and evidences required for verification and validation of the full software system at each major joint review milestone, while in an Agile process the focus of each iteration is on producing working software for the features of that iteration?
- d. Process Management Concerns:
 - Customer Concerns:
 - There is a perception that doing Agile is time intensive for customers. In a waterfall project, we, as customers, can dedicate our time only at certain slots around the major joint reviews, while in Agile processes we have to participate much more actively to a lot of meetings and invest much more time throughout the project.
 - In a waterfall process, decisions are taken at formal reviews with many reviewers, panels and boards. In an Agile process with frequent interactions between supplier and customer, many decisions are taken by the whole team at each iteration. How to avoid a change in the role and a shift in the ownership of risks and responsibility from supplier to customer?
 - Supplier Concerns:
 - Agile Process can only be successful when there is a close collaboration with the customer. How do I get the required input and the feedback from the customer if they are not participating to the Agile process as they should?

Furthermore, the adoption of an Agile development method can introduce other challenges for the supplier and customer organisations, such as:

- The value of adopting an Agile process in a project is disseminated among all stakeholders.
- Possible additional effort due to new artefacts brought in by Agile methods (e.g. co-existence and maintenance of product backlog and requirements specification).
- How to align the system and sub-system development life cycles, particularly when one is not following an Agile process while the other is.

The rest of the handbook is intended to provide best practices and recommendations on how to address these concerns.

Guidelines for Agile life cycle selection

5.1 Selecting Agile

As Agile is a largely deployed software development method that has been pervasive across all types of businesses and organisations for many years, several suppliers have changed their usual software development practices to address various new challenges and Agile is one of the key enablers. Agile can bring in significant value over traditional practices; however experience indicates that it is not a silver bullet. Agile practices are developed based on several fundamental assumptions about the environment within which the project is executed. In the right context, all promises of Agile become realistic; but in the wrong context Agile can add significant risks to the successful delivery of the product. Please refer to clause 4.2 for Agile promises and risk related to its adoption.

This clause 5 aims at providing guidelines to:

- Help in deciding whether or not a project is suitable for Agile development based on criteria related to the environment and the product itself.
- Help in ensuring that the process that sustains the adoption is as well described.

5.2 Analysis of key factors for Agile selection

5.2.1 General

This clause provides a set of criteria or factors that play a role in the software development process, and that can help to guide the decision either for agile or against agile, but that can also help with the selection of the model in clause 6.

Concerning the Agile mind set, it is not very adaptive or agile to mandate one single approach with mandatory factors. A selection of factors that work best in the relevant environment is key for success. It is worth noting that theoretical fit and practical fit are often quite different.

There are some groups of people or organizations with interests in the result of the software development process, which in the most common case are the customer and the supplier as organisations. We also have to remember that these factors are just a support and they are not a replacement for thought and dialogue with the project stakeholders. Rather than using them in isolation, use them to start a conversation about Agile suitability and build consensus around the method of choice in close relationship with all the project stakeholders.

The process that handles the selection is the Agile Assessment. As explained on clause 5.3, the Agile Assessment is done as soon as possible, that is during the bid phase, and includes as a minimum the stakeholders.

There are also other aspects to take into consideration that are not project related. An organisation/company plays a critical role in assessing the suitability of adopting Agile and should therefore be part of this process. Even if the project is a great fit for Agile in theory, if management or other stakeholders are against that approach, then its application carries significant risk.

NOTE Non-project related factors, related to the organisation itself, are recommended to be included in the assessment.

These criteria can be grouped into 4 contexts as follows:

- Customer context: evaluation focuses on customer ability to receive frequent deliveries, to define the need and to collaborate with the project's team.
- Supplier context: evaluation focuses on the partner capabilities to develop the project and to deliver the product or the solution to the customer.
- Project context: evaluation focuses on the solution to be developed in terms of characteristics, architecture and design.
- Team context: evaluation focuses on team ability to perform Agile development (including size, co-location, multi-disciplinary)

Figure 5-1 shows the different factors/criteria that are covered by each context.

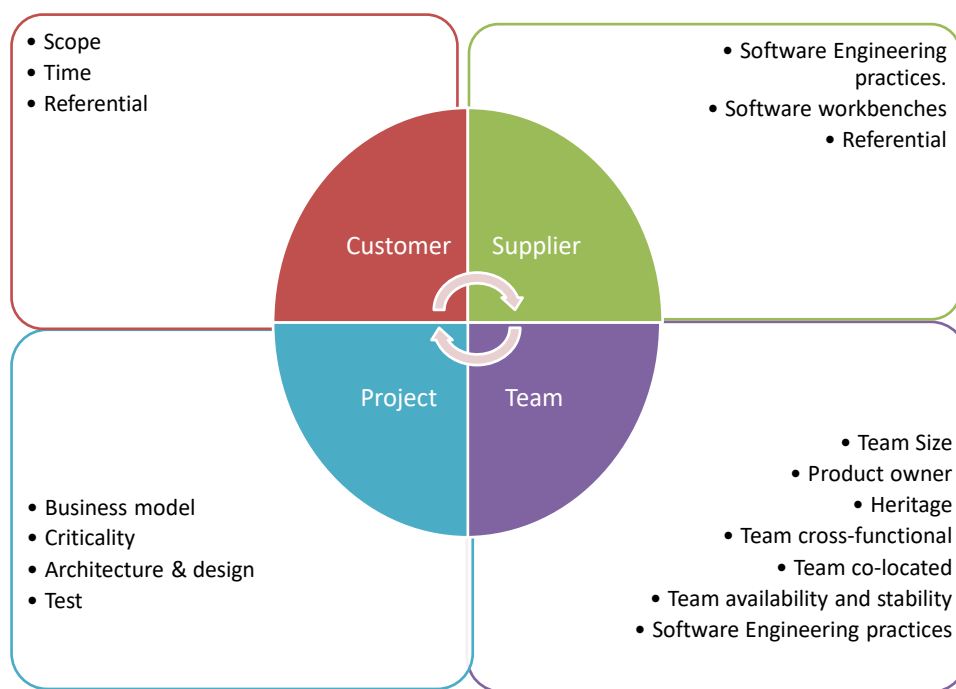


Figure 5-1: Factors for adopting Agile process

NOTE Not all the contexts and the underlying criteria are equally important and it is recommended to weigh them. This weighting is based on the experience of Agile experts and strongly related to the company environment.

All factors are evaluated according to the same scale with five values. This scale is depicted in Figure 5-2. For each context, an average is calculated using the scale.

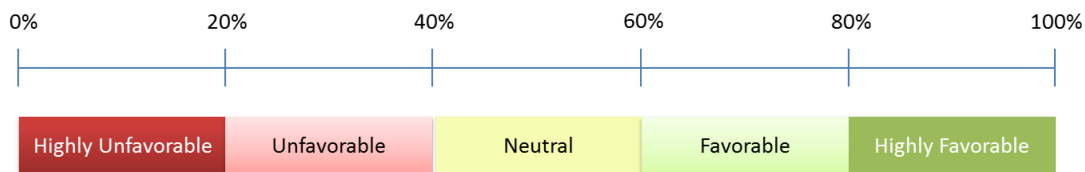


Figure 5-2: Agile selection factors scale

The global average can be calculated by applying weighting coefficients to the four individual context averages. The weighting coefficient would in this case express the perceived importance of a particular context and can be decided by the relevant stakeholders.

5.2.2 Customer context

The customer context covers the following factors:

- **Scope:** Are the requirements (or user-stories) of the project well known? Are the requirements stable throughout the development life cycle? Agile methods allow for change and uncertainty, then projects that have not well defined requirements are well suited to an Agile process. The refactoring effort involved for incremental development may not be justifiable when the requirements are frozen upfront. The Agile approach is more adapted when requirements changes requested after the beginning of the project are frequent and need to be taken into account quickly in a new iteration, but the agile approach can also be adequate when all requirements are well-defined and understood from the beginning, though some of its practices will not be needed.
- **Time:** Is the customer request for a rapid (i.e. first project release) and then regular incremental delivery? Agile methods bring value due to its emphasis on working software, therefore projects that require an incremental delivery with an early project release are well suited to an Agile process. Has the project a fixed time frame? The Agile approach is particularly useful for getting maximum business benefit within the given time. So Agile is favoured when intermediate software deliveries can provide added value to end user/customer.
- **Experience and awareness:** Does the customer understand Agile methods? Is the customer willing to change/adapt project management practices? This criterion allows to assess whether or not an Agile mind set is in the customer's referential. Applying an Agile process in a project with a customer without any knowledge on Agile carries significant risks.

Context	Factor	Unfavourable	Favourable
Customer	Scope	Well-known and stable requirements	Not-well-defined or not-very stable requirements
	Time	No or few intermediate deliveries required	Intermediate deliveries required regularly.
	Experience and awareness	Lack of experience and mind set in agile.	Interest/experience in Agile

5.2.3 Supplier context

The supplier context covers the following factors:

- Software engineering practices: Is the supplier experienced with software engineering practices, which are typically used in Agile software development projects, such as continuous build and integration, test driven development and pair programming well mastered by the supplier? How long has the supplier been developing software using Agile process? How mature are the processes (and the people) relative to software development?
- Software workbenches: Does the supplier have the adequate software (e.g. continuous integration) workbenches in order to be able to deliver the project iteratively and incrementally? Does the supplier have the necessary knowledge and experience to use these workbenches?

Table 5-1: Supplier context

Context	Factor	Unfavourable	Favourable
Supplier	Software Engineering practices	Mature non-agile Software Engineering Processes	Mature Agile Software Engineering Processes
	Software workbenches	Tools and workbenches not available to support the agile process	Available tools and workbenches supporting agile, i.e. continuous build and integration engine and related tools

5.2.4 Project context

The essence of the Agile approach is to develop software iteratively and deliver it in usable increments that provide value to the business. The nature of the requirements, as well as the solution to meet them, play a key role in executing the project using Agile methods.

The project context covers the following factors:

- Business model: is the target an internal system, a commercial product, a bespoke system on business agreement for a customer, a component of a large system involving many different parties, an evolution of a large legacy system? This criterion allows assessing whether demonstration of direct business value for the customer using an agile lifecycle is easy or not. As an example, in software development from scratch, it could be favourable to adopt Agile because it aims at providing early working versions of the software from the very beginning. On the other hand, for developments based on legacy systems, it may not be easy to demonstrate additional business value through adoption of an agile lifecycle, if the migration does not add new functionality. This is partly because the business value of the application exists already with the legacy software. Also in such cases it may be difficult to achieve the required level of automation of the testing and continuous integration with the legacy applications.
- Criticality: What is the criticality of the project? Documentation needs to increase dramatically to satisfy external agencies who want to make sure the security of the investment and the safety of the public are guaranteed. Some critical activities may be governed by specific procedures or rules.
- Architecture and design: Is there an implicit de facto or reference architecture already in place at the start of the project? Does the project have well-defined external interfaces? Projects that have a reference architecture avoid a big up-front design phase, hence are well suited for an

Agile process. Projects that have external dependencies are not well suited to an Agile process. Those external dependencies can also include hardware constraints.

When developing software for a very constraining system (e.g. CPU, memory, real time constraints), it is preferable to start when the requirements are stable, and therefore be able to optimise it so Agile is not well suited.

- **Test:** Is testing an increment at sprint or release level too expensive in terms of effort, time and availability or test means? Is test automation easy to put in place? Is the test effort compatible with sprint or release standard duration? Typically if testing an increment is expensive, probably the team cannot afford to do it very often. Agile requests to demonstrate all the user stories within a sprint and check non regression as well. In a context, when test effort requires extending the sprint duration above 4 weeks, trying to apply an Agile method based on Scrum is not recommended. Projects that have costly testing environment (test harnesses) with regards to the project development itself are not well suited to Agile process. The Agile approach imposes the availability of integration and test means (real hardware or simulators or stubs) very early in the process to start with testing from the beginning on.
- **Resource availability (e.g. hardware):** Due to the anticipation/advancement and execution of all or some of the ECSS-E-ST-40 activities in parallel at early sprints of an agile life-cycle, additional need for availability of artefacts on which the software under development relies may be accordingly pushed forward in the schedule of the software development project. For instance the HW on which the software must be executed, would be needed, if prototyping, coding and validation are already performed in early sprints. The same applies on the need for another software, which is required for integration and for validation purposes of the software under development. Conversely the agile life cycle can help to better specify the hardware specification through prototyping activities. Such considerations must be taken into account in the project context, when selecting an agile software development lifecycle. Possible project specific solutions/mitigations may be needed accordingly (e.g. use of emulators, simulators, mock ups, etc.).

Table 5-2: Project context

Context	Factor	Unfavourable	Favourable
Project	Business Model	Legacy software, migration or evolution of an existing software system	Development from scratch or according to a reference architecture
	Criticality	Critical Software, e.g. flight software or numerical software verification facilities	Non-critical software, e.g. ground software
	Architecture and design	No existing reference architecture or not yet defined interfaces to other elements as software or hardware	Existing reference architecture or well-defined interfaces to other elements as software or hardware
	Testing	Difficult to automate tests. Complex testing environment.	Easy to define and automate tests. Simple testing environment.
	Resource availability	Dependency on artefacts external to the project (e.g. hardware)	Not clear dependencies on external artefacts (e.g. hardware capabilities to be decided based on prototypes)

5.2.5 Team context

Agile development favours individuals and interactions over processes and tools. The approach is more people-centric; hence, team composition plays an especially important role in assessing Agile suitability. The availability of the right skills and the team's ability to collaborate play significant roles in the success of the project.

The team context covers the following factors:

- **Size:** How many people are involved in the project team? The overall size of the system under development is an important criterion, as it drives the size of the team, the number of teams, the needs for communication and coordination between teams. Agile adoption recommends a small team, so teams with less than 8 people are well suited.
- **Team cross-functional:** Is the team cross-functional? Does the team have all skills to develop the project? Teams that contain all necessary expertise are in a better position to identify and resolve impediments as they occur. Having team members with different experiences and points of view provides more comprehensive feedback. Agile recommends that all team members are willing to learn and fosters a "generalist" approach.
- **Team co-located:** Linked sometimes to the size of the project. Is the development team co-located? How many teams are involved and are not collocated? This increases the need for more explicit communication and coordination of decisions, as well as more stable interfaces between teams, and between the software components that they are responsible for.
- **Agile heritage:** Does the project team have a previous experience in Agile process? This criterion allows to assess the Agile coaching effort to sustain the team and reduce the risk of Agile adoption.
- **Team availability and stability:** Is the team dedicated to project? How stable is the team throughout the development? Agile recommends stable and available team, so without this criterion deploying Agile in that context may increase risk.
- **Product owner:** Is there a single product owner who is fully dedicated to the project? Does the product owner provide end-user continuous feedback and commitment? A product owner available and working closely to the team in order to provide it with a continuous knowledge and feedback on user need is a key criterion for Agile adoption.
- **Software engineering agile practices:** Does the team have the relevant knowledge and experience in software engineering agile practices?

Table 5-3: Team context

Context	Factor	Unfavourable	Favourable
Team	Size	Big team > 9 people	Maximum 9 members
	Cross-functional	Highly specialized team	Specialized but cross-functional team
	Co-located	Geographically distributed team	Fully co-located team
	Agile Heritage	Experience in agile developments < 3 years for all members as an average	Experience in agile developments > 3 years for all members as an average
	Availability and stability	Team working on several projects.	Fully dedicated team to a single project.

Context	Factor	Unfavourable	Favourable
		Team changes over the project lifetime.	Team is stable over the project lifetime.
	Product Owner	Partial time product owner, or lacking domain knowledge or soft-skills	Fully integrated and available product-owner with soft-skills and good domain knowledge
	Software Engineering practices	Team members do not master agile practice	Team members master agile practices

5.2.6 Key Factors Summary

Table 5-4 lists the key factors considered to drive or prevent either the classical or agile lifecycles.

Table 5-4: Key factors for selection of classical or agile lifecycle

Lifecycle	Classical		Agile	
Factor	Positive	Negative	Positive	Negative
Customer: Scope		Unknown or unstable requirements		
Supplier: Software Engineering practices				Agile practices not well understood at company level
Supplier: Software Engineering Workbenches				Knowledge and deployment of tools not available at company level.
Project: Criticality	Critical software			Critical software
Project: Testing				Not easy testing, automation and environment.
Team: Size	Team size >9			Team size >9
Team: Heritage				No substantial agile experience
Team: Availability and stability				Team is shared among different projects.
Team: Product owner				No dedicated and knowledgeable product owner.

5.3 Agile assessment process

The Agile assessment process can be performed as early as possible so during the bid phase and/or development phase, the criteria are evaluated with the elements known at this given moment in the project. Then, the evaluation assessment is only valid for a given time period.

GOOD PRACTICE

It is recommended to perform the Agile assessment process as early as possible and re-assess it in a periodic manner.

Factors that can impact the decision within an organisation are provided in the four contexts described in clause 5.2. Furthermore, this evaluation is performed in a recursive way according to the Product Breakdown Structure (PBS) & Organisation Breakdown Structure (OBS) of the project under consideration.

GOOD PRACTICE

It is recommended to consider the global picture of the development. As a result of that, the Agile assessment process can be performed in a recursive way at each stage of the development organisation.

Finally, it is important that this engineering practice selects an appropriate Agile method and is led by someone with enough Agile experience and background from the organisation, acting as an independent assessor within the environment during a dedicated meeting with all the project stakeholders.

GOOD PRACTICE

After the project assessment, it is also recommended to have the commitment and the full support from management at the customer and supplier sides.