Adapted Agile:
How to setup your agile development model

Summary

Agile development is used in many different environments, although the level of implementation differs. Clearly it is here to stay, with its promises of flexibility, reduced time to market and customer centric development. Implementing agile according to its original definition with context requirements like large scale development or functional safety is not straightforward. This is where Adapted Agile comes in. This whitepaper outlines how to establish an adapted agile development model in challenging contexts.

addalot QUALITY IMPROVEMENT

Adapted Agile

1. Introduction

Agile development methods have reduced delivery times, improved productivity, increased quality, and made customers more satisfied within software development.

Succeeding with agile describes how agile increases productivity, quality, and employee satisfaction

Several companies have though realized that their particular context make agile implementation more difficult. It can be e.g. large scale, hardware, security, or open-source usage that makes it difficult to fully apply the agile methods. Several of the contexts e.g. expecting documentation and detailed planning are less valued by Agile.

Embracing Agile development is not a binary decision where you either need to do it by the book or not at all. To meet the expectation of the different applicable contexts you need to adapt your agile implementation.

(*) From Addalot survey : 90% are trying Agile 70% need to adapt Agile 55% struggle adapting Agile

Our experience (*) is that many companies have difficulties with these adaptations. So how can you succeed with establishing an agile development model adapted to your context?

One important challenge is to have enough understanding of agile principles and methods and what the requirements of the contexts mean to be able to perform the adaptation. This knowledge is a starting point. Then you need to understand your own context to be able to understand the fit.

It is like selecting shoes that match your feet and context. First you must find a pair with the right size. But fit is just the first step, some shoes are light and fast, perfect for track and fields but might not be what you need for a mountain hike. The white paper brings ideas on how to succeed with Adapted Agile so you can become more agile and more efficient than you are today!

Paper Structure

addalot QUALITY IMPROVEMENT

2. Agile development

Agile development is used in many different environments, although the level of implementation differs. Clearly it is here to stay, and not without reason:

- Time for product definition is limited and it needs to happen in parallel with development.
- Development is becoming more complex and difficult to drive top down, favoring exploratory development over "expert teams".
- Software companies must deliver new functionality fast to meet changing market needs.
- Many customers expect small continuous updates rather than large chunks of functionality.
- There is a need to be able to quickly react on changes.

Agile started in small IT settings where it quickly proved its value, but as with any solution to a complex problem, not even Agile is a silver bullet. Implementing Agile according to its original definition in other domains turned out to be difficult.

Instead of throwing the agile baby out with the bathwater when faced with a complex context, **you should make your development as agile as possible but not more agile than that**.

GOAL:
You should make your development as agile as possible but not more agile than that.

Adapted agile guides you to find this sweet spot, between the extremes in the agile manifesto, see picture below. It introduces the required minimum of planning, documentation, etc. to manage the context requirements while enjoying the agile benefits.

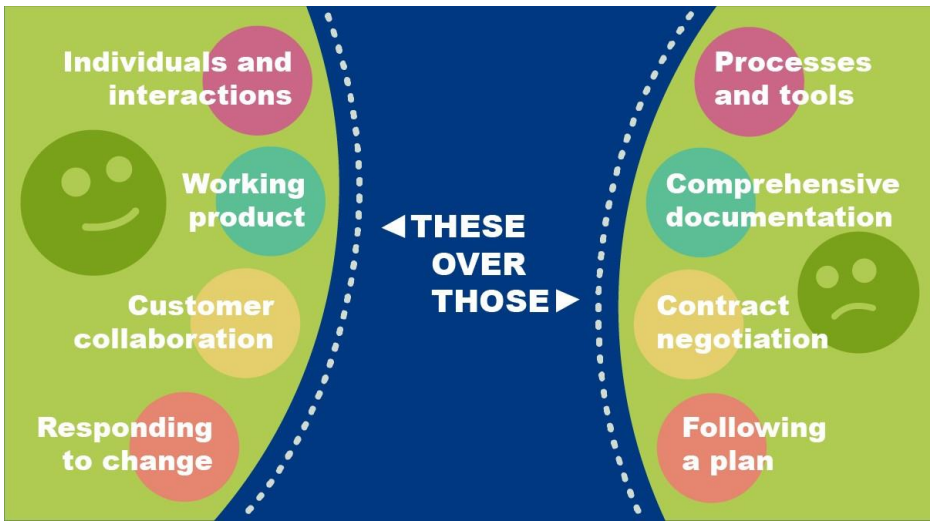


Figure 1: Agile Manifesto, favoring left side over right side.

ASSUMPTION:
The Manifesto can be seen as a sliding scale where the left side is "more agile" than the right side.

addalot QUALITY IMPROVEMENT

3. Challenging contexts

Many companies do not have a set up where agile can be applied by the book. Depending on requirements of their context, several adaptations to agile values and principles are needed in order to establish an effective way of working. This chapter presents some contexts and describe their characterizing requirements/challenges for agile development. Typically, several of these contexts come to play – and partly overlap.

Adaptations will require you to move towards the right side in the Manifesto sliding scale

Large scale

Large scale development is characterized by that the number of teams or sites grow. The challenges start already with two teams and get more serious at 9-10 teams. This corresponds to the number of interaction points that we as individuals can manage. Regarding the number of sites, already when you move from colocation to two sites significant challenges start to occur. In addition, there are also large scale aspects when you have multiple product lines.

The ideal set up for agile development is one collocated team…

Challenges:

- With more people or additional sites communication needs to be strengthened. This involves more meetings and more documented (physical/electronic) information.
- When size grows it often results in that the number of dependencies grow. With more dependencies, the need grows for more synchronization and fixed planning which may reduce flexibility.
- When the number of people and teams grows, it is not uncommon that specialization increases, e.g. a separate team for GUI or architecture. This leads to more handovers that needs to be documented.
- Large scale can result in a distancing to customer, as it is easy to focus on internal contribution and not on customer benefit.
- With multi-site development, cultural as well as time differences require more documentation.

Long life

Products (generally large) with long life impact the need for structured maintenance. There is a high likelihood that feature update and bug fixing will be done by someone else than the original developer.

Products with long life require investment in maintenance!

Architectural decisions for product with long life are extra critical which require structured decisions and that the rational for the

decisions are recorded. And is not always something than can be developed bottom up by dev teams. However, independently of how the architecture decisions are handled and by whom, it is particularly important that the rational for the decisions are documented when a product has a long lifetime (and the lifetime of software is often longer than expected).

When long life starts to make an effect is somewhat impacted by the turnover of personnel. If you experience a high turn over the effect will occur more promptly.

Challenges:

- With long lived products it will be high likelihood that feature update and bug fixing will be done by someone else than developed in the first place. When future development and maintenance will be performed by other people than the original developers, the requirements on the documentation increases. Aspects that can be difficult to understand by just reading the code need to be explained, e.g., requirements, high-level design, and design rationale.
- The expected lifetime of a product may influence the requirements on the architecture and the consequences of the design decisions. For example, decisions that have an impact on maintainability, changeability, and portability may be more important when the lifetime of the product is expected to be long.
- Like architecture decisions, decisions related to development and test tools can be extra critical when the product has a long lifetime. Changing or adding development and test tools may have a significant impact on the future development and test activities and may also involve costs to acquire the tools and to provide training.

Subcontracting

Subcontracting involves that a part of the development is made by one or several suppliers. In automotive development, you often see an OEM (Original Equipment Manufacturer) with long chains of suppliers (OEM➔Tier1 ➔ TierX)

Subcontracting includes soft context requirements related to culture

Challenges:

- Multiple suppliers imply parallel development, which requires clear interfaces and more synchronisation and documentation.
- Order mechanisms are by nature not agile. Contracts can stipulate fixed scope, fixed price, fixed schedules, competitive bidding, and tough penalties for underperformance. This

results in a need to know exactly what to produce which requires massive pre-planning. Delivery to some governments, especially in the defence industry share the same difficulties.

- If the subcontracting involves hardware, the payment terms will likely be related to the number of items and not to the development, which will make change handling more difficult.
- Sourcing departments that perform evaluation of suppliers do not always have enough technical competence resulting in that price supersedes competence and experience in most cases.
- The culture in supplier development is often less cooperative; the fundamental approach is that the suppliers shall be exchanged continuously to limit business dependencies. This leads to more formal interfaces and management.

Hardware

System development that includes both HW and SW is common for most embedded products. The presence of HW forces SW development to adjust and reduce the degree of agility in the SW development.

Challenges:

- Long lead times with key decision points related to HW is often applied on SW (even if it does not make sense…). E.g. project gate/milestone checkpoints can include aspects like "all requirements defined" or "design ready".
- Since HW is often on the critical path, SW development often must prioritize HW needs, like HW test code (that later must be thrown away)
- The increased complexity increases the need of dialogue, interfaces, and documentation
- Automation of testing is more difficult since CI/CD/DevOps with hardware is possible but more challenging.
- In many projects including SW/HW is the HW part is not decided and need a research part (i.e. solution not fully decided). This keeps SW hostage and need to keep alternatives open.
- Late problems related to HW is often transferred to SW – hard to close features and increase the amount of CR.

Hardware represents both electronic and mechanical development

Challenges for applying Agile in HW development is not considered in this paper.

Process Capability Models (ISO/CMMI/ASPICE)

Some development organizations have decided to comply with a process capability model (PCM) like ISO, ITIL, CMMI or Automotive SPICE. This can often be driven by customer requirements, particularly in the defence domain (CMMI) and the automotive domain (A-SPICE).

This white paper is not evaluating the importance or necessity of the requirements from the process capability model but merely stating that if you want to meet them you need to be less agile

The process capability models claim to be consistent with agile, but the reality is rather that they can co-exist, and an agile development organization needs to add many practices and documentation in order to live up to the different models.

Challenges:

- The process models require a lot of documentation (strategies, plans, decisions, status, records). Including:
  - that key decisions in architecture and detailed design are evaluated and formally documented.
  - documented procedures for how to perform many activities like how to write test cases, handle defects and establish baselines.
- Another key area for process models is ensuring consistency and correctness by expecting traceability, reviews, and quality assurance assessments. These areas cost both time and render in administration and documentation which are not highly valued in agile.
- Essential in process models is process management, i.e. the focus to document the way of working.

Safety

Organizations producing safety related products need to comply with different functional safety standards. There are many standards like IEC 61508 (generic), IEC 62304 (medical device), ISO 26262 (automotive) EN5012x (railway), DO-178B (aviation), and IEC 60880 (nuclear). These standards become a ticket to trade for the involved producers, without compliance products cannot be released.

Challenges:

The content of the safety standards overlap with the process capability models; thus, the challenges are very similar. In addition there are some unique challenges:

- More strict order of development, including that previous development steps are approved before work proceeds.
- Safety concept work including safety goals, safety analysis, hazard analysis and safety case. Which all result in documentation.
- Safety standards expect an established and maintained development lifecycle.
- More prescriptive development and verification methods. This includes e.g., inspections, static code analysis, test case design using equivalence classes and boundary values, and structural test coverage of requirements (statement coverage, branch

coverage, etc.). Normally, most of these methods can be used also in agile development.

Cyber Security

Largely characterized by that the development organization has decided to comply with a Cyber Security Standard (e.g. NIST Cybersecurity Framework for Manufacturing, ISO 21434, IEC 62443 and ISO 27000). Security standards have similar challenges as safety standards.

Challenges:

- Security related activities like Threat model, Attach surface analysis and criticality analysis. Some of these activities need to be performed early and in a certain sequence. They may also require external participants with special competence. When adapting an agile way of working for security related development, it needs to be clarified how these activities will be handled.
- Secure product development lifecycle, including security requirements, coding, and test.
- Deliver frequently can be challenging:
    - it is not clear when some of the security development practices should be applied, like when to update/review threat model.
    - some of the security development practices are manual and expensive to repeat, like penetration testing.
- In agile you often do not design for needs that could or will come up in the future while security advocate for that full solution is designed from the beginning.

Open-Source development

Large organizations that are actively using and contributing Open-Source (OS) components need to adhere to the OS development guidelines. Several OS aspects support agile (e.g. deliver often, focus on working software, focus on technical excellence), but there are also some points were agile and OS development are not aligned.

Challenges:

- OS is often more technology driven than customer value driven with little focus on business participation.
- Some OS licenses require documentation.
- OS does not prioritize face to face interaction, cooperation is secured more through documentation (e.g. read me files) and through electronic interaction like web pages and mailing lists.

- OS has a strict view on community roles with respect to usage and ownership, Agile is less focused on the topic except eXtreme Programming that has the principle around "collective code ownership".
- OS is not always resource effective rather driven by survival of the fittest. While agile focus on simplicity and lean approaches.
- OS excludes continuous improvement of way of working.

Summary of contexts

The different contexts and their impact on the Agile manifest are summarized in the table below (XX: high impact and X: some impact):

	Individuals & interactions over processes and tools	Working software over comprehensive documentation	Customer collaboration over contract negotiation	Responding to change over following a plan
Large scale	X	XX		XX
Long life	X	XX		
Subcontracting		XX	XX	XX
HW		X	X	XX
PCM	XX	XX	XX	XX
Safety	XX	XX		X
Security	XX	XX		X
Open Source		XX	X	

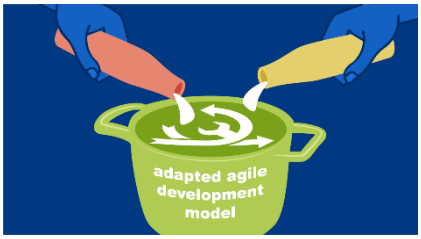
Other

Agile is applied in several other settings that require adaptations, but not covered in this white paper:

- Regulatory requirements (FDA regulations, PCI SSF, SOC2, etc).
- In organizations with elaborated budget process that require extensive pre-planning including pre-studies to develop business cases (cost/benefit analysis), etc.
- Management teams, going from more separated, document driven work to smaller assignments with higher degree of cooperation.
- Business strategies, going from long term planning to continuous evaluation with more experimentation.
- Hardware and mechanical development going from traditional plan driven to include simulations, experimentation with 3D printing.

Finally, there are also contexts with no contradiction and rather strengthens/complements agile like "Lean development" and "Data driven development".

4. How to derive your adapted agile model



To establish an adapted agile development model, there are two input sources to investigate. The first input source is related to the context requirements while the second deals with independent agile principles.

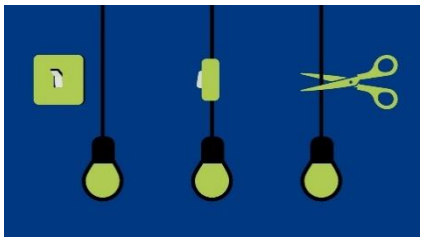
1) Fulfill context requirements. The specific context requirements that need to be met, can be implemented in solutions more or less agile. To succeed you need to understand what the context requirement really demands and implement new way of working that satisfy the context and still be remain agile as possible.

Remember:
as agile as possible
but not more agile than that

2) Apply agile principles. Many agile principles can be implemented without impacting the context requirements.

The two input sources are outlined in the following two subchapters:

4.1. Fulfil context requirements



With the first input source the requirements from the different contexts need to be analyzed and understood to be able to implement them in an as agile as possible way. The key is to understand that the expectations from the different contexts CAN be implemented in different ways. Just as putting out the light can be done in various ways. Some more agile than others.

First comes three general remedies to consider, then follows some context specific advice.

Isolation

One way to limit the implication of the context is to analyze which parts of the product and organization are impacted by the relevant context requirements? Is it the whole system or can we define the architecture in such a way that we can isolate the need to specific parts? Both safety and large process model requirements can be reduced in this way. Understanding the use of the product and how the system interacts with its environment is central to establishing an effective system architecture. For the part of the system where the context requirements are relevant there will of course be a need for meeting them.

A thought through architecture is needed to successfully isolate the more context demanding parts of your system.

For example, in a system used to supervise and control some critical

physical process, the parts of the system that controls the process can be safety critical and needs to be developed according to applicable parts of a safety standard, while the parts of the system that interacts with the users may be less critical and can be handled as normal development work.

Scheduling of practices

For the context requirements that must be considered, (additions of needed activities and documents) the level of agility should be investigated by answering the following questions:

- What must be done initially? (e.g. role description, initial requirements, and architecture).
- What can be done continuously in the iterative flow (e.g. requirement refinement, implementation, and plans).
- What can be done informally in the running sprint and formally after the final sprint (e.g. formal reviews of requirements, architecture, design, and test report).
- What can be done in specialized iterations (e.g. specific tests).
- Can new meetings/documents be avoided and instead use existing activities/documents (e.g. can parts of QA be done in the retrospective? Can risks be follow up in project meetings instead of own risk meetings?).

Not all activities are effective to iterate. Compare going shopping groceries one item at the time….

The goal of going through the questions is to find the balance between the amount of investigation, specification and planning that needs to be done before development starts, what can be done continuously, and how much documentation can be done afterwards. It saves time and increases the quality not to complete the whole puzzle upfront but to finish it over time. In the early stages focus should be on consistency, that the appropriate objects exist and their interfaces, rather than their completeness. However, starting too early without enough understanding can be very costly as it may lead to a lot of reworks later in the project.

Documentation

One of the largest challenges is to find effective ways to manage the amount of documentation expected from the context requirements. Documentation can be related to strategies, plans, estimates, product documentation, review records, traceability matrices, root cause analysis, technical evaluations, read me files, test reports, release notes, etc. Many are mandatory from the different process models and standards. While some are more optional, like expectations from OS communities

REMEMBER:
Context requirement will lead to additional activities and in particular documents….
You can only make them more or less costly to manage-
They can meaningful --- but you will become less agile…

or integration plans to manage multiple suppliers.

A starting point is to clarify that Information ≠ Documentation and that the context requirement is most often requiring the presence of some information. It can be in form of a document, or it can be embedded in a tool. For example, you do not need to create a "traceability matrix document" if you can do a query in your tool to show dependencies. Another example is that some information can be generated, like a test report that can be exported from your test tool.

Information ≠ Documentation

What documentation shall be avoided? Documentation that is needed because of handovers (Product managers sending requirements to development teams, Architect sending architectural models to development teams or development teams sending design and code to testers) shall be avoided – consider more agile approaches:

- Common start up meeting with all relevant roles (Product managers, Product owners, Architects, Development teams, Test) to ensure everyone have a common understanding from the start
- Limit handovers by using T-Shaped teams, where team members have a deep expertise in one functional area but also ability to work outside their core area, that can take responsibility from requirements to test.

Documentation that is outlining decisions is more important, level of detail can though be different depending on complexity and time validity.

Handling product documentation with reviews and approval is cumbersome. One solution is to handle the documentation as code. This is solved by using a markup language (e.g. Markdown or AsciiDoc) and applying pull requests also for document updates.

Handle documentation as code!

Many benefits:

- Documentation is easy to find.
- Product documentation can be updated together with the code (always in synch).
- Perform continuous reviews on relevant parts and not everything all the time.

Specific solutions/adaptations:

Here follows some guidelines for the specific contexts:

Large scale:

For large scale and distributed development several risks lurk around the corner.

- Documentation will increase and needs to be managed as described in documentation chapter above.
- The agile solution to coordination is meetings instead of documentation. But many large organizations fall into "meeting paralysis", where too much time is spent in unproductive meetings. Studies show that developers in large organizations spend 2.5 more hours a week/10 more hours a month in meetings than developers in smaller orgs. It's called the "coordination tax".
    - Avoid all-hand meeting
    - Consider team of team approach so secure information is tailored per level in an effective way
    - Only the needed people shall be called for a meeting (avoid good to include …)
    - To secure enough "focus time", introduce meeting free days/morning/afternoons
- Keeping track of dependencies
    - Architecture should drive the organization set-up, and not the other way around, this helps to limit coordination. (Value stream mapping can help to identify recommendable organizational boundaries.)
    - Cross functional teams that take responsibility over the full feature implementation reduce the coordination need.
- To stay away from establishing handover and bottle necks in the development flow a solution is to make the development team responsible for the whole vertical, from system requirement to system test, but to establish support people for various areas where resources are scarce (UX, ….)
- Multisite development
    - Require continuous communication
    - Invest in equipment, e.g. conference equipment
    - Physical meetings from time to time helps a lot
    - Avoid spread out teams, but rather let the different sites have functional responsibilities.

Be aware of *some process models like Scaled Agile Framework® (SAFe), include a lot of good practices and information but is very extensive and often do not start from the need of the organization. Several SAFe implementations introduce a lot of practices without understanding the intention. In addition, SAFe proclaim that the backlog shall be quite defined and locked for three increments ahead, which is about nine months ---- not so agile…*

Other concepts like "LESS" and "Team of teams" is more adding to the need of a growing organization

For more details on SAFes agility, see: SAFe and Agile Values

Long life:

Handling products with long life include aspects to secure efficient maintenance:

- Ensure that evaluations or at least rationale of decision become part of the architecture and not residing in a separate PowerPoint that no-one will find some year later.
- Not all decisions need to be formally evaluated. Ask yourself how significant the decision is now and some years from now.
- Introduction of new employees can be used to strengthen documentation. When new people read up, they can have the task to add information where they fall short of understanding.
- Maintain test coverage on all levels and prioritize automation.
- Keep track of technical debt, most statical analysis (e.g. SonarQube and klocwork) tools have checks for maintainability. Keeping the code clean is essential, apply the scout rule: "*Always leave the code cleaner than you found it"*
- To ensure the integrity of the system, proper version control is needed. This includes tracking, recording, storing, and retrieving the different versions, revisions, and modifications of software and documentation, as well as providing mechanisms for merging, branching, and comparing them.

Mindset:
Develop and document as if you are leaving and will hand over to someone else

Subcontracting:

When systems are developed by multiple parties, mechanisms to secure synchronization and coordination must be put in place. To some extent like large scale but with added challenges around setting a cooperative culture.

Subcontracting context requirements are especially noticeable in low margin domains with multiple suppliers. (e.g. automotive)

- Clear interfaces/responsibilities both with respect to architecture and way of working help to minimize coordination and synchronization.
- It is essential to not lean back and rely on contractual agreements. Follow up must be continuous and include both technical aspects and on progress. The scope/frequency shall be stipulated in the contract, often defined "Scope of Work" so that it do not cause any debate and delay the follow up.
- Establish common goals. Do not only have delivery goals from supplier to OEM but have goals on working functionality that require all stakeholders to participate.
- Common technical environments enable a better flow, especially if a frequent delivery approach shall be implemented. Where applicable, cloud environments and common open-source projects are ways going forward.
- In an agile development environment, it can be a great advantage if also supplier management is more agile, i.e., more

focused on cooperation and common goals than on specification and follow up of detailed contracts. A more cooperative approach requires mutual trust and may increase the dependency to the suppliers but can make the overall development process more efficient.

- Regular common retrospectives (lessons learned meetings) with the supplier to continuously improve the cooperation. The identified improvement activities can be included in the product and sprint backlogs and often be handled in the same way as other activities. The progress of the improvement activities should be followed up in the common retrospectives.
- There are several posts on how to perform agile contracting, here are some well described Best Practices
    - BP #1: Separate Business Risk from Software Work
    - BP #2: Define Scope at a High Level
    - BP #3: Emphasize Delivery Process Not Deliverables
    - BP #4: Define Acceptance at a High Level
    - BP #5: Time and Materials, Not Fixed Price
    - BP #6: Sharing the Gain and any Pain
    - BP #7: Go Easy on Downside Protections
    - BP #8: Contracts Do not Create Trust: People do

Hardware:

Succeeding with agile hardware development is an interesting topic – but not in scope in this whitepaper. The focus in this section is to describe how to handle the potential negative impact hardware has on agile SW development.

- Ensure that any steering model that make sense for HW is not enforced on SW, for example:
    - Detailed requirements ready at initiation
    - All functionality described at the time for ordering production equipment
    - Strict formalized test phases
- On the other hand, maximizing the freedom and agility of SW development might not optimize the whole delivery. It is important to strengthen system focus and understanding – optimizing on the whole rather than HW/SW in isolation.
- During development and test, HW can be simulated in SW or emulated using FPGA (programable integrated circuits) prototypes. The development of the simulators and emulators resembles normal SW development and can be integrated in an agile SW development process.
- Invest in rigs – automated SW testing must be as independent of HW progress as possible.
- Increase dialogue – establish regular synch,
    - Understand consequences of HW shortage
    - Do not assume, prepare for SW alternatives

Process Capability models:

- Documentation handling is a large challenge from different process capability models, expecting strategies, records, logs, and evaluations.
- Formal document reviews:
  - Most models expect that it shall be specified: who should participate, review guidelines, written feedback, clear decision of review, but not necessarily a physical review meeting.
  - Handling document review as code pull request can make them more efficient.
    1) Indicate specific changes 2) Simplify notetaking 3) Each reviewer needs to indicate if a meeting is needed or if the author is expected to handle the comments without a meeting 4) Review meetings can be skipped, if not needed (most context requirements do not require the review meeting, only that the relevant stakeholders participate, and that the comments and decisions are persistent).

- PCMs require clear scope/estimates. The solution is to balance long term planning with short term planning. This typically result in:
  - Overall release planning
  - Increment planning
  - Sprint planning
- PCMs expect architecture/design with traceability. To make this effective establish a tool chain where queries can be made to indicate coverage and missing links. This way the effort for traceability can be simplified.
- Traceability
  - The traceability that is applied often e.g. traceability between SW requirements and test cases is highly meaningful and easy to access.
  - Traceability that is seldom used does not be that easily retrieved. E.g. “blame” will indicate all commits that have impacted a specific file.

- The PCMs also require elaborate process management, including descriptions, templates, instructions, tailoring guidelines, etc. A working processes management will help to ensure common understanding of the agile way of working but the risk is that that it will be too detailed and consume a lot of resources to maintain.
- Instead of first documenting a perfect theoretical PCM compliant agile process and then trying to implement it, it is usually a better strategy to build on the current ways of working, i.e., to first document the current ways of working and then gradually improve and document the real implemented process towards PCM compliance and agility. This strategy will help making the process documentation consistent with the real ways of working and will probably also help avoiding

unnecessarily detailed documentation.

- To manage the risk of "over documentation", the focus must be to secure that the documentation is not too detailed and is only a part of making your work procedure stick. A more effective way is to build the process into the templates/tool chain.
    - Establish DoR and DoD for key work products like (requirements, epics, …)
    - Include control in the pull request (e.g., Unit test, Static Code analysis, Review, Build and integration test)
    - Quality gates in the integration & build pipeline

Safety/Cyber Security:

Several of the Safety and Cyber security aspects have been covered in the other contexts above. Safety and Cyber security may require more documentation and formality in their handling.

A related issue is how to carry out the safety/security assessment during agile development. One strategy is to perform incremental informal assessments in the sprints and a final formal assessment at the end. The purpose of the incremental assessments is to give early feedback on the safety/security related work. They can be informal and do not need to fulfill all the requirements in the standards on how the assessment should be performed. The purpose of the final assessment at the end is to perform a complete formal assessment according to the requirements in the safety/security standard. Because of the incremental assessments in the sprints, hopefully, only few major issues should be found in the final formal assessment.

A balancing act is how much of the Safety/Cyber Security specific work that needs to be done upfront?
Doing e.g., a threat model analysis is more challenging in an iterative approach. How often and how deep analysis is required? We cannot wait until the final solution is defined but need to identify when an iteration has impacted the model enough for revisiting the analysis.

An important aspect is who is doing safety/security-critical work? It is sometimes done by safety/security managers/engineers in parallel with the development projects (A process/safety/security team).
A more agile way is to make sure that it is done within the project. That these additional people become part of the teams, even though they focus on the adaptation. In the same way as testers in a development team focus on tests. However, a central function is often necessary to ensure adaptation by providing expertise and coordination.

Open Source:

Addressing the context requirements from open source are mainly adding some documentation and some additional procedures.

- Documentation (emails, read me files, etc.) needs to be managed since OS communication require this for community interaction (rather than live communication) and documentation

needed to fulfil community license expectation
- Adding perspective to open source is not violating any context requirements, it is just not expected.
- Adhering to OS roles is not large compromise. For a more active OS participation clear roles and responsibilities need to be defined related to the making commit decisions, propose solutions and perform reviews.
- To make OS more resource effective, increased communication is needed to ensure that community participants have a clear view of roadmap.
- Successful active OS participation will also require
    - development strategy that defines how the components of the product will be developed in a Make-Buy-Share strategy
    - product ownership / contribution strategy

REMEMBER:
Context requirements can be meaningful --- but you will become less agile

4.2. Apply agile principles

Even though your situation is complex with many context requirements, e.g., large scale + HW/SW development + fulfillment of ASPICE, Safety and Security, there are several agile principles that can be applied for a more agile development model without impacting the context requirements. This section will highlight several of the agile principles that can be applied and help you to become more agile independent of potential context requirements.

*The **12 agile principles** have been roughly followed, with some minor own interpretations*

Customer focus - How do we ensure continuous dialogue with the customer? Many times, you do not have the opportunity to have the customer available. Instead an internal proxy is established, often called “Product Owner”. To make a Product Owner a successful customer representative, it is important they are exposed to customer situation (review customer cases, site visits, demos, etc.). It can also be recommendable NOT to make them a member of the development team to highlight that they represent the customer, placing orders to the team. The work products by the product owner, requirements, epic descriptions, etc. shall be written with customer context.

- Why is it important for the customer?
- What is the customer problem?
- What would be the main benefit for the customer?
- How will it be used by the customer?
- Who are the “personas” that will use the solution?

Another important lesson is that to cover complex systems, the Product Owner is rather a function than an individual. So, you need several different people to be able to represent the full product.

To ensure prompt customer feedback, it is important to define a **"Minimum Viable Product" (MVP),** i.e. the minimal product that is still

large enough to get customer feedback. It may be for limited use but still provide real user experiences. The MVP approach also help to maintain the customer focus. In some contexts, the MVP approach is not so suitable, for supplier that is expected to deliver a specific scope it might be difficult to split/cut the delivery. Defining an MVP can still provide value from enabling feedback (at least internally) and enable to focus on a working product.

Focus on working software is practice in it-self. There are different ways to strengthen this focus. To continuously run demos on different levels is a good start. To make it effective, technical demos can be run within the team or between teams to show progress after sprints. On higher levels more customer-oriented demos should be run to show what the customer can do with the implemented functionality. Metrics that visualize the progress of working software can also contribute. Instead of following up on hours or passed milestones, % of implemented (working!) product features are more relevant. But maybe most important is to drive the development in small steps that are integrated, tested, and potentially delivered.

Continuous deployment is a concept that does not work easily in all contexts, e.g., for safety development since it requires specific tests and documentation that could be very time consuming. Also products with HW are not always suitable for continuous deployment. But **continuous integration**, the concept of implementing and testing small chunks of work is relevant in most contexts. If the small chunks can be established as verticals, providing customer benefit, then it will support testing in steps, enabling relevant feedback, build internal understanding, supporting the concept to perform demos showing customer benefit and visualize quality and status. **Continuous delivery** can be a sweet spot for many organizations. It requires that if the development of the chunks should be ready for delivery, potentially some of the formal steps required by some of the context requirements might remain to be done. This forces the organization to mindset that what get started should be driven to a (almost) ready state and not postpone relevant documentation and other responsibilities.

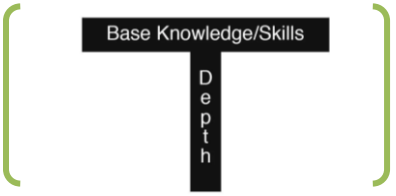
Prioritization - a key aspect of agile development is to achieve a continuous prioritization of the work. The focus is to achieve **simplicity**, the art of maximizing the amount of work not done. The Product Owner function must balance the prioritization of customer benefit and system understanding with information of cost, implementation order, and available skills. This priority then needs to be clear for the entire project. From a safety critical perspective, content that is part of the safety concept must be prioritized early

because it needs to be part of the delivery and cannot be scoped out.

Planning - agile work is both incremental (development in steps) and iterative (stepwise refinement). To achieve this, planning is required. Both to get the overall plan in place and then to establish a model for the continuous planning. For safety-critical development, scope preparation and initial planning needs to be more extensive as one needs to establish more structures. The continuous planning needs to handle:

- Control of which requirements/features are specified and ready to be developed and which need more investigation.
- Refinement of large tasks so that they do not run across multiple sprints.
- Dependencies and synchronization between teams.

Team set up and support – to minimize handovers and to make the development teams focused on customer value **cross-functional teams** are preferable to functional/component teams. Focus need to be on "T-shaped" competence profiles with the ability to collaborate across disciplines with one (or several) in depth expertise. With cross functional team there are risks:



- implementing too large teams ➔ keep them around 6-8 people
- have key people part of multiple teams ➔ one team per person
- continuous rearrangement ➔ Keep teams stable, not forever but enough to allow continuity

In addition to competence mix it is important to establish trust and team empowerment by staying away from micromanagement. The team must have clear responsibilities and provided with an environment where motivation can blossom. This includes letting teams make technical decisions and sprint planning but also providing the needed information and qualitative input (e.g. requirements)

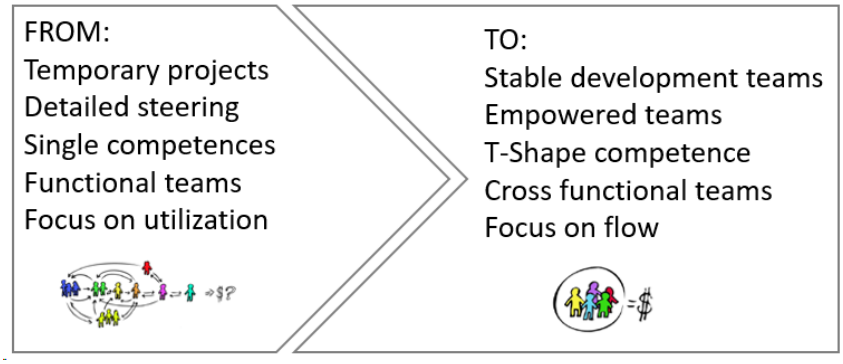
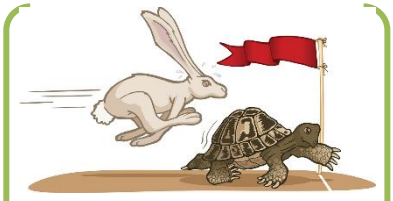


Figure 2: The shift in team set up and management

To be able to maintain the efficiency of a development organization a **sustainable pace** must be established. The mistake is burning ourselves out in completing all the work that the team has forecasted for the sprint. It is good to enable predictability but having flexibility is equally important in complex work. Unsustainable pace leads to poor quality and lowers morale.

Good practices for sustainable pace include:

- Daily check in, requiring teams to split up work in digestible pieces
- Allow time for increment and sprint planning and perform reviews to enable learning and improve estimation.
- Product owners and scrum masters shall act as gate keepers reduce the noise from other stakeholders to avoid squeezing in additional unplanned work during an ongoing sprint.
- Focus on work product flow rather than staff utilization



Sustainable phase is a winning concept!

Changes – Agile is embracing changes. A workflow where changes can be evaluated and included in a structured way is needed. One way is to handle changes at sprint review and demo. It is a natural moment to open up for feedback and changes. It is important to clarify that additional changes to working functionality will impact yet unrealized functionality. Determining whether it is better with fewer really good features, or many just-working features is a business decision. It should not be a decision for a development team (especially if the development team has limited overall product under-standing, which is the case for many complex safety-critical products). Another important aspect is to have different control levels for change, e.g. more control if the change is impacting working functionality and less if it is impacting the scope in the backlog. Internal changes (design and code refactoring) should also have their control. Refactoring should in general be encouraged but in a managed way, see below.

Agile **development principles** – Some of the agile development principles can be introduced without impact on the context requirements. Refactoring and Test first are good examples.

There is one agile practice that can be more difficult to implement when the development in done short sprints with constant focus on new development and fixed bugs. From an agile perspective there should be a **continuous attention to technical excellence and good design**. Paying continuous attention to refactoring and reducing technical debt helps obtaining maintainable and readable code and scalable design. But how to make it happen? It must be part of the team's expectation not only to develop new features but to address technical dept. Technical dept needs to be identified/measured (complexity, cohesion, coupling, duplication, test coverage, code

smells). A percentage of the increment should be dedicated to refactoring, SAFe has one sprint (Innovation and Planning Iteration) dedicated. Unfortunately, it is often used as a buffer for the functional development and refactoring is often neglected.

Face to Face meetings are a natural part for small development teams. But for large scale and distributed development, face to face meeting risk to be replaced by written information. The solution is to identify key points in the development where a meeting adds substantial value. Agile ceremonies for the single team, e.g. planning, daily meetings, review and retro are typically performed but the challenge are cross team related / hand over meetings. Increment planning similar to SAFe's PI planning or Setting up Teams of Teams to ensure information is transparent are good examples. When implementing an epic which involves several teams a good practice is to introduce an epic start up meeting to ensure that all stakeholders have the same information and agree on their agreed commitments.

Continuous learning, along with continuous improvement, is an important agile principle. Establishing retrospectives that improve the team's working methods are important. Daring change and improvement is central to high performance teams and projects. It requires that the organization allow failures as long as we do it quickly and learn from it!

7. Conclusion

To establish your adapted development model, you first need good understanding to fulfil the different context requirements in an agile way, then you need to consider what you can pick from different agile methodologies. There is no silver bullet, you need select nuggets adding different pieces to the development model puzzle.

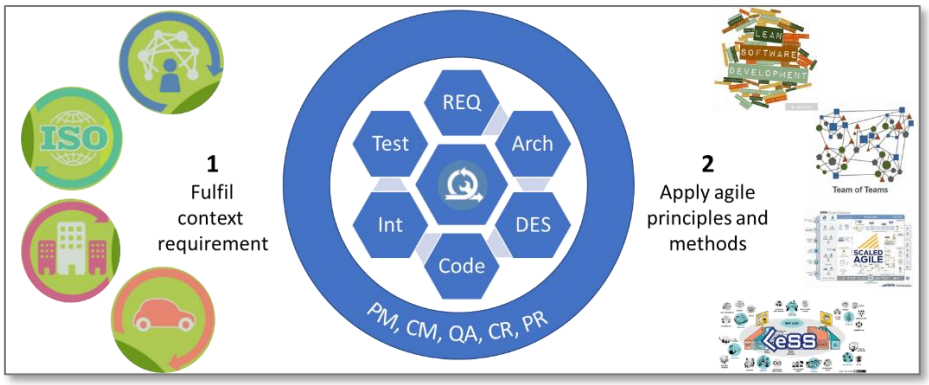


Figure 3: Overview of how to establish an effective development model.

One of the largest challenges for companies with many context requirements is that they abandon agile ways of working when they see that they cannot apply agile by the book.

The key take away is that your context is likely pretty unique and deserve a thought through approach. Your decided way of working can be more or less agile, the goal should be to make it agile as possible but still meeting the context requirements. And finally, the likelihood that there is a pre-defined model perfect for you is very small...

About Addalot

Background

Addalot Consulting has over 30 years of experience in process improvement within the field of systems and software. The company was founded in 1989 as Q-Labs, a spin-off from Ericsson, and became a leader in Europe for services related to optimizing software development companies. Q-Labs was acquired by DNV in 2006. The operations were transferred in 2011 to the newly formed company Addalot Consulting. Addalot helps organizations improve results and reduce risks by streamlining their software development processes.

Approach

Our fundamental approach is that the process, the prevailing way of working, strongly affects both the quality of the products being developed and the lead time for development work. Many companies focus on results and wish for improvements (faster, cheaper, better quality) without considering which capabilities need to be improved for this to be possible.

Efficient way of working lead to better software

Addalot's expertise:

Process Capability – Faster and more efficient and reliable processes
Adapted Agile – Succeeding with agile in complex environments
Industrial Open Source – Enabling delivery and business opportunities
Functional Safety – Handling of safety critical software (lita på)
Secure Development – Identify and address vulnerabilities proactively

Customers

Addalot helps both large and small organizations in a variety of industries: ABB, Actia, Advenica, Alfalalval, Ansaldo, Assa Abloy, Atlas Copco, Autoliv, Axis, BAE Systems, Baxter, BMW, Boing, Bombardier, BorgWarner, Bosch, CabinAir, Combitech, DB Schenker, Delaval, Diadrom, EADS, Elekta, Embitel, Ericsson, Fingerprints, FMC, FMV, GM, Handelsbanken, Ikea, Ikano, Kockums, Kongsberg, Lawson, Littlefuse, Maquet, News, Nokia, Palette, PEAB, Playtech, Postnord, Point, Qualcomm, Qlik, Readsoft, Region Skåne, Saab, Scania, Schneider, SEB, Simcorp, Sony, Stoneridge, T-Engineering, Telenor, Telia, Terma, Thales, Veoneer, Verisure, Visma, Visteon, Volvo.

Contact

We are active in Göteborg, Malmö and Stockholm, see web for contact.

www.addalot.se