IP-SOC 2012

Are we too Hard for Agile?

François Cerisier and Mike Bartley, Test and Verification Solutions
Grenoble France

Introduction

The software community are moving from linear development processes such as “waterfall” (see Figure 1below) where development is supposed to proceed down the right hand side but often has to back track up the left hand side as requirements change or are clarified, mistakes are discovered, etc.

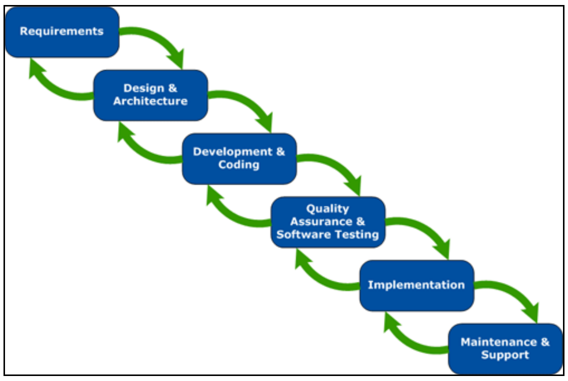

Figure 1: The Waterfall Development Model

In reality a number of software projects follow a more complex development procedure where a number of stages are actually executed in parallel as depicted by the V-model of development.

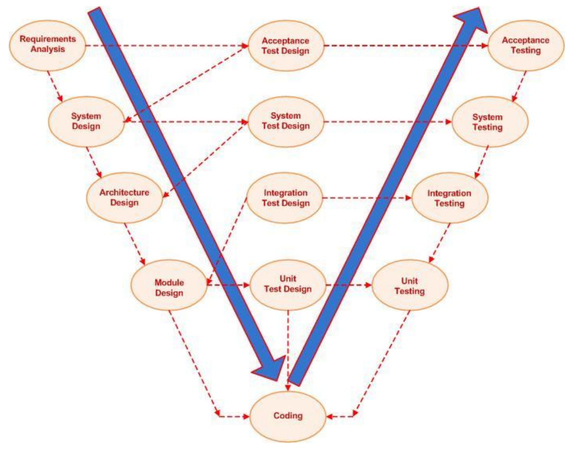

Figure 2: The V Development Model

The V model allows test design to occur at various levels of hierarchy in parallel with the development process. Test design can also inform subsequent development stages. However, the V model remains mainly a linear process and is based on a single requirements document at the start. This was the main driver for agile movement who believed that the waterfall and V models were too fragile in the face of changing requirements.

Agile development techniques are now sweeping through the software community transforming the way we develop software. But apart from a few isolated reports the hardware community has been largely untouched by this revolution. Hardware development typically follows a fixed, linear development process with well-defined stages of development (specification, design, verification, physical design and software). In this paper we first give an overview of agile development and discuss the advantages that software developers report from deploying an agile development methodology as well as the potential pitfalls. We then describe the author’s experiences of applying agile development techniques for hardware development as well as those reported in the literature. Finally we consider the applicability of agile to hardware development. The aim of the paper is to give the reader the knowledge and insight into how they can potentially apply agile within their own projects.

What is Agile?

“Agile” is a catch-all to describe a wide variety of iterative development processes with “scrum” (this breaks iterations into “sprints” which will be described in the full paper) being the most popular. The agile movement was started with “The Agile Manifesto” written in February of 2001 (see www.agilemanifesto.org). This was developed at a summit of seventeen independent-minded practitioners of several programming methodologies. The participants found consensus around four main values.

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change above following a plan.

The final lines of the manifesto state “…while there is value in the items on the right (e.g.

processes and tools), we value the items on the left (e.g. Individuals and interactions) more." What are the items on the right? They are the characteristics that always seem to represent due diligence and professional practice but, if allowed to dominate the project environment , then repeatedly lead to project failure, dissatisfied client and unhappy employees. These were in reaction to sequential development processes typified by "waterfall" and "V-model" (which does have some parallel aspects but cannot be considered iterative). The manifesto has subsequently been expanded with 12 principles:

- Customer satisfaction by rapid delivery of useful software
- Welcome changing requirements, even late in development
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Close, daily co-operation between business people and developers
- Face-to-face conversation is the best form of communication (co-location)
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity- The art of maximizing the amount of work not done - is essential
- Self-organizing teams
- Regular adaptation to changing circumstances

These values and principles outline the "underlying philosophy" of the agile movement but do not really lead to an easy a methodology or process that can be followed to successfully develop software. In order to fill this gap a number of "agile methods" have been developed such as Dynamic Systems Development Method (DSDM), Extreme Programming (XP), Feature Driven Development (FDD), Kanban and Scrum.

An Overview of Scrum

Scrum is an iterative development process. Projects progress via a series of iterations called sprints. The length of a sprint can vary but is typically 2-4 weeks long. Personnel are formed into teams which are between five and nine people. However, companies are scaling their use of Scrum into much larger teams often using "scrum of scrums".

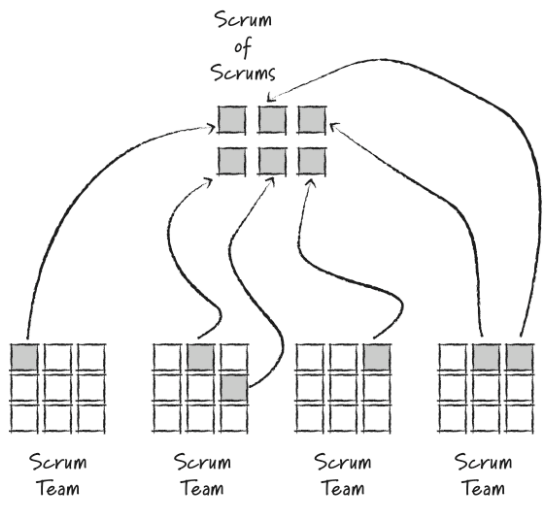

Figure 3: Scrum of scrums

Scrum (as applied to software development) does not include any of the traditional software engineering roles such as programmer, designer, tester, or architect. Everyone on the project works collectively to complete the work they have committed to complete within a sprint. The Scrum teams thus forms a common ownership of the work to be performed rather than just focusing on their particular task. This is a very important psychological aspect of scrum.

Scrum does identify two key roles within the scrum team:

- The **product owner** is the project's key stakeholder and represents users, customers and others in the process. The product owner is often someone from product management or marketing, a key stakeholder or a key user.
- The **ScrumMaster** is responsible for making sure the team is as productive as possible. The ScrumMaster does this by helping the team use the Scrum process, by removing impediments to progress, by protecting the team from external tasks and influences, etc.

Given agile prioritizes "Working software over comprehensive documentation" then as you might expect agile avoids a large initial specification process and document. Instead agile identifies features which are documented as agile User Stories. These are written in the following style.

```
"As a <role>, I want <goal/desire>
so that <benefit>"
```

For example

```
As a user, I want to search for my
customers by their first and last
names.
```

The User Stories are put into a **product backlog** which is a prioritized features list containing every desired feature or change to the product.

At the start of each sprint, a sprint planning meeting is held during which the product owner presents the top items on the product backlog to the team, and the Scrum team selects the work they can complete during the coming sprint. That work is then moved from the product backlog to a **sprint backlog**, which is the list of tasks the team has committed to complete in the sprint. This process is important as the team forms a commitment to the sprint backlog.

Each day during the sprint, a brief **daily scrum meeting** called the daily scrum is conducted. This meeting helps set the context for each day's work and helps the team stay on track. All team members are required to attend the meeting

At the end of each sprint we again focus on "Working software over comprehensive documentation". To do this the team **demonstrates** the completed functionality at a **sprint review** meeting where the team shows what they accomplished during the sprint. This is deliberately informal avoiding, for example, long PowerPoint presentation and avoiding long preparation. The meeting must not become a task in itself nor a distraction from the process.

Finally, at the end of each sprint, the team conducts a **sprint retrospective**, which is a meeting during which the team (including both the ScrumMaster and product owner) reflect on how well Scrum is working for them and what changes they may wish to make for it to work even better.

Reported advantages for agile

A number of advantages are ascribed to agile software development – for example

- Agile helps to speed up the development process and bypasses process steps that add little value to the project.
- Usually Agile methodologies promote less formal culture and encourage collaborative team approach.
- Agile facilitates smooth flow of knowledge sharing.
- Engages the stakeholders continuously so that the new requirements are gathered faster and there is no scope for guess work by the teams.
- Agile incorporates frequent and rapid changes into the product functionality and features.
- Saves cost, time and efforts by following iterative incremental work delivery and thereby identifying deviations early.
- Provides the end result of higher quality of the software delivered and a highly satisfied customer.

There are of course a number of disadvantages reported too. A few examples are given below together with the arguments from the agile community as to why they are not accurate.

- "*Agile doesn't fit our organizational culture*": The agile community would suggest the solution here is not to reject Agile but to change the organizational culture.
- "*Agile only works for small projects and our projects are big*": The scrum response would be to adopt a "scrum of scrums" and the literature does have examples of successful adoption in both large organizations and on large projects.
- "*Agile requires co-location and our staff are geographically dispersed*": It is true that agile teams are best when they are working closely together in the same physical space to encourage face-to-face, spontaneous conversation. However, co-location is not always possible. In these situations agile teams use technology to maintain open and continuous communication. They also use temporary re-location of staff to increase trust and communication between the distributed team.

Reference [1] gives examples for successful large, distributed teams.

Obviously the above advantages and drawbacks can apply equally to hardware development as much as they apply to software. However, it is not clear that hardware development lends itself to an agile development process. The rest of the paper considers this point.

Firstly we will consider aspects of hardware development that seem to promote an agile approach. We then consider aspects that could potentially make it harder.

Is hardware suited to agile?

Is this section we consider how agile *could* be applied to hardware. There are a number of differences between hardware and software development that can potentially make agile not suitable for hardware development. Below we describe some of the differences and in the following section we consider how practical that might be.

One major difference between software and hardware is the "realization step". In software this is a compilation step which more or less complex depending on the target hardware architecture. However, hardware realization involves a complex sequence of synthesis, place and route, scan insertion, etc. and then manufacture. Obviously manufacture of each iterative release is not possible. However, even the other steps make hardware realization a long process that is potentially very time consuming and iterative. This might mean that it is not practical to realize the hardware on each agile iteration. In this case the iteration may just deliver a functional model (e.g. an RTL model or some higher abstract model such as SystemC).

Reference 2 looks at whether hardware development has the same problems of software development that the agile movement was trying to solve. Specifically it considers the level of creativity in hardware development by asking pertinent questions such as:

- Do project plans remain stable over time?
- Do product requirements remain stable?
- Would a hardware team design and build a product the same way twice?

In the experience of the authors the answer to all three questions is "no" but this is too simplistic as it consider all hardware equal and does not consider the major differences between hardware and software.

Hardware tends to have more well-defined interfaces (I2C, DDR2, PICe, etc). However, the hardware will have some USP (unique selling point) that distinguishes it from the competition and it often the USP requirements that require flexibility in requirements.

Agile is often described as being most suitable in environments that have detailed user interactions and/or complex user interfaces (often graphical). Software often interacts with an end user who often finds it hard to fully specify all their user stories and the desired interface to the software. The detailed user interactions require strong customer engagement and the graphical user interface lend themselves to the high customer engagement that agile foster during the development process. It can be argued that hardware interfaces are often standardised (such as I2C, PCIe, etc) and which therefore require less customer approval. However, this does not mean that user interactions are less complex. A typical hardware product user will be a software engineer who interacts through a complex register map as the interface. The software engineer will also be concerned with system performance and be providing feedback to the hardware architect and development team.

Agile deliberately does not distinguish between developers and testers, attempting what it sees as false barriers in the development team. However, over the past 20 years hardware development has moved to separate design and verification teams. This separation has occurred for a number of reasons:

- The philosophy suggests that it is too hard for developers to find all of the bugs in their own code. It is assumed that the assumptions they made in the design are also made in their verification.
- Advanced verification techniques such as constrained random, functional coverage, assertion-based verification and formal verification require specialist skills and thus require specialist verification engineers.

Of course, as discussed above, realization of the hardware also requires different skills and most hardware development teams employ specialists for tasks such as place-and-route, DfT, etc

One of the driving forces behind agile was the desire to improve the quality of the delivered software. Bugs and debug have a similar impact in hardware as they do in software. Software and hardware are also similar in the respect that the longer a bug is left undiscovered the more expensive it is to fix. The means that early testing and debug should be promoted. Agile promotes this by promoting a bug free product at the end of each sprint. Indeed, there is research to suggest that NOT fixing bugs in a sprint can cause agile projects to fail. It seems obvious that hardware development would strive for a similar goal.

Practical considerations for applying agile to hardware

"***Feature-driven development***": Hardware implements features in the same way that software does. For example, the I2C protocol has various addressing modes (7-bit, 10-bit and General Call Addressing Modes). At a more complex level a CPU has different instructions, addressing modes, performance features, etc. These features could potentially be developed in different sprints. This would take a radical shift in development process that is captured well in the following diagram from reference [2].

Figure 4 compares the current linear "waterfall" development model with a features driven agile development model. The hardware features would create a product backlog and sprint backlog. Each release would include a new set of working features.

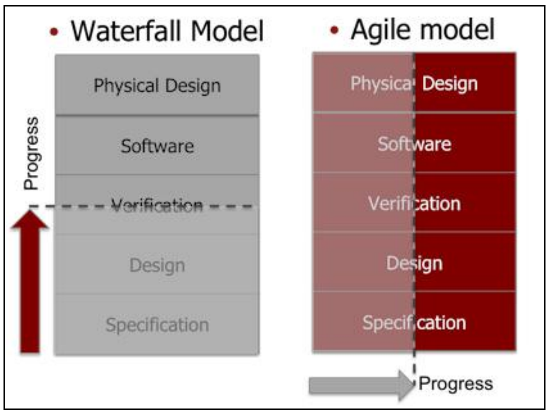

Figure 4: Agile model applied to hardware development

- For *design IP* this would allow for numerous releases to a SoC team. Currently a number of design IP teams make partial releases based on quality. A feature driven development would mean a number of releases all of high quality but increasing functionality.
- SoC designs could be broken down hierarchically into smaller subsystems forming smaller projects (i.e. a scrum of scrums). Each subsystem could be integrated incrementally through a number of planned iterations with increasing features.
- Verification sprints should also be aligned on the hardware feature releases, defining verification targets when features are made available. Many reports on agile in software development identify testing of features within an iteration as being crucial to success. Testing of features developed in iteration *N* should not be delayed until iteration *N+1*.

One criticism of feature driven development is that the architecture and structure of the code deteriorates as new features are continuously added. Projects employ a variety of techniques to avoid such issues. For example

- The first iteration may be an architecture iteration that simply constructs the hierarchy. Remembering that iterations should deliver "working software" this could mean that a top-level entity is delivered. This allows for the customer to give feedback on the pin I/O and a test bench architecture to be developed.
- Some iterations may be "re-factoring" iterations where no new features are added. The code is simply tidied to improve structure.

"***Target driven development***": Hardware designs often have targets for functionality, power, area and performance. The various iterations can potentially focus on the different targets. So for example a design IP could become fully featured before focusing on power reduction.

It is also possible that agile does not need to be applied to the complete hardware development process but could be applied to certain aspects of the development. For example, it could be applied to just VIP and IP development. In VIP development the iterations could involve deliveries of:

- Features extraction and functional coverage.
- Driver and basic sequences.
- Scoreboard and assertions.
- Complex sequences for hitting corner cases.

The above iterations and delivery are based test bench elements rather than being feature drive and in the experience of the authors this is a very practical way to deliver VIP and get feedback. An alternative is to deliver all test bench elements but provide incremental support for verification of design features – i.e. a feature driven approach. Again in the experience of the authors this is an equally successful way to develop and deliver verification IP.

When applied to design IP agile development allows early releases for beta features, incremental release to customers, allowing early IP integration and VIP setup.

<u>Discussion</u>

The main motivations behind the agile movement for software development are to allow delivery of high quality software whilst at the same time embracing change. This is done through small teams that deliver regular working software releases through iterations of length 2 to 4 weeks. The teams have very little delineation in terms of their role and in scrum (the most widely adopted agile methodology) the "product owner" and "scrum master" are the only two identified roles.

It seems that the goal of delivering higher quality is a common goal between both hardware and software development. Indeed, as the cost of bugs in hardware is generally accepted to be higher than in software (due to the time and cost involved in the manufacturing process) it could be argued that it is a higher priority in hardware. Indeed, research suggests that hardware verification costs are much higher than software testing costs.

The level of flexibility required in software does not seem to be reflected in all hardware development. A number of the features and interfaces in hardware are fixed and well defined in advance. However, some aspects of the design add more USP than others and it is often these that require more flexibility to allow a product to be better distinguished in the market.

Hardware and software differ most in the physical realization process. In hardware this is ultimately a manufacturing process which can only realistically occur for the final product. However, the steps involved in reaching the manufacturing steps also involves a number of complex, time-consuming steps involving specialist skills. This makes it harder to have no distinction between team member activities. It would seem that the differences in the realization process between software and hardware means that agile should only apply to the front end development team. However, this still leaves open the issue of the split between design and verification which is deliberately avoided in agile but is increasingly popular in hardware. Of course, it is always possible to have designers verify other people's designs which can avoid the issue of verifying one's own designs. However, there is also a big skills gap between the two roles and specialist HVLs (Hardware Verification Languages) have reinforced that skills gap. It could be that the reunification of design and verification languages under System Verilog may see the reverse of that trend. However, the skills involved in design and verification currently seem to be too far apart to be mastered by all but the exceptional few engineers in the industry.

The role of scrum master would seem a natural one to continue when applying scrum to hardware development and a product owner would also be a good addition to ensure that questions on features could be answered promptly.

The authors have witnessed iterative hardware development on numerous projects but this is based mostly on a quality basis rather than on a feature basis. Hardware development teams often deliver a feature complete (or at least 90% complete) design that may only be 50% verification complete. Subsequent iterations then add bug fixes and higher verification confidence. The authors have seen far fewer feature driven hardware developments although they do exist and have been successful.

Other agile techniques that could be used

Whilst scrum is the most popular example of a methodology that encapsulates agile philosophy, there are a number of agile techniques that could be applied equally to hard development such as:

- ***Pair programming*** in which two programmers work together at one workstation. One, the driver, writes code while the other, the observer reviews each line of code as it is typed in. The two programmers switch roles frequently.
- ***Test-driven development***: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test and finally refactors the new code to acceptable standards.
- ***Continuous integration*** (CI) is the practice of merging all developer workspaces with a shared mainline several times a day. Its main aim is to prevent integration problems which can also be found via frequent integration.

Unfortunately there is no space in this paper to investigate these further.

Conclusion and Recommendations

In the opinion of the authors agile development practices (in particular scrum) cannot be adopted in hardware development without some adaptation. In particular, it should mainly be considered for the front end process and not all back end processes should be completed on each iteration.

Having all of the team able to perform all of the development steps would seem to be too big a step especially when it comes to back end and serves as another argument to only consider front end processes. However, removing the distinction between design and verification engineers would reverse the movement of the industry over the past two decades. It might therefore need that the team accommodates a split between design and verification initially. The move to smaller teams with the two scrum roles of scrum master and product owner would seem to be a good contribution to hardware development. This would necessitate a "scrum of scrums" to be able to develop large SoC's and complex design IP. Hardware hierarchy seems to lend itself well to such an approach.

A feature driven iterative approach could also be adopted and could be advantageous to those areas of the hardware which add the USP. It would seem that standard interfaces and blocks do not need the same level of feedback although they may enable higher order features to be evaluated that could add USP. It does seem that feature driven development for those areas of the design that add more USP may add value to the development process as it can allow earlier feedback. Such iterations require close alignment with verification as all the features need to be properly verified.

Thus, as mentioned earlier, there does seem to be some scope for the adoption of agile development practices (in particular scrum) in hardware development but not without some careful adaptation

References

1. "Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum" by Craig Larman and Bas Vodde.
2. "Agile hardware development – nonsense or necessity?", Neil Johnson, www.eetimes.com