Agile development for a multi-disciplinary bicycle stability test bench

Adrian Cooke, G.Maarten Bonnema and Wim Poelman
Faculty of Engineering Technology,
University of Twente,
P.O.Box 217, NL-7500AE Enschede, The Netherlands
Email: Adrian.Cooke@utwente.nl
G.M.Bonnema@utwente.nl
W.A.Poelman@ctw.utwente.nl

***Abstract*—Agile software development methods are used extensively in the software industry. This paper describes an argument to explain why these methods can be used within a multi-disciplinary project and provides a concrete description on how to implement such a method, using a case-study to support the rationale. The SOFIE (Intelligent Assisted Bicycle) project was created to develop mechatronic appliances to make bicycles more stable, i.e. safer. A bicycle stability test bench is created within this project and is used as the case study for this research. The relative complexity of the test bench development and partner structure within the SOFIE project has many similarities with large-scale complex projects found in industry. Thus it provides a good environment to research the application of Agile software methods to a multi-disciplinary project.**

INTRODUCTION

Agile software methods have traditionally been used on pure software projects, this paper provides a method to adapt Agile software methods for use within a multi-disciplinary project. A case-study representative of complex multi-disciplinary projects is used to illustrate how this can be done.

The case-study is the bicycle stability test bench (Fig. 1), which forms part of the SOFIE project (http://mobilitylabtwente.nl/sofie/). The test bench is used within the SOFIE Project to test the stability of bicycles, of riders and the effectiveness of the mechatronic appliances which will be developed to assist in the stability of the bicycle. These devices are called Intelligent Assist Devices (IAD) within the SOFIE project. An additional use of the test bench is to validate a computer model of the bicycle rider system that is also developed within the SOFIE project.

The different partners, i.e. the people developing the IADs, the computer model and performing experiments are found in different organisations. These organisations are in the same geographical area, but they are not co-located and the members are from different professional disciplines.

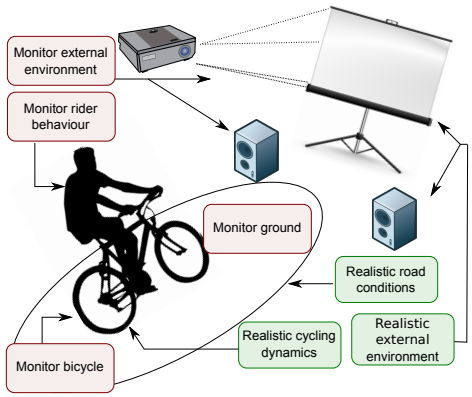

Figure 1. Bicycle stability test bench physical overview.

The design of the test bench is a non-trivial technical problem involving sensor development, mechanical appliance development, software development and advanced dynamic motion mathematics.

The project has a limited time span and clear deliverables. The team structure, multi-disciplinary technical problems and project outcomes lead to a relatively complex project topology. This topology has issues with communication, project management and the creation of a useful working end product, which are problems that are found in large scale industrial projects. This provides a good environment to perform research on the application of Agile software methods to a multi-disciplinary project.

The structure of the paper contains a brief description of Agile software methods, the bicycle stability test bench design, and how these Agile software methods are implemented in the test bench project. A discussion of this implementation and description of future work will conclude the paper.

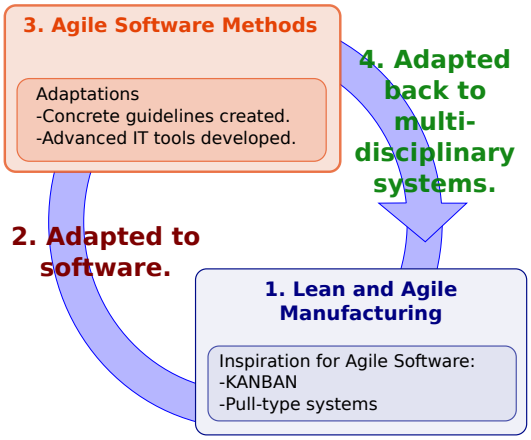

Figure 2. Agile software methods evolution.

AGILE SOFTWARE DEVELOPMENT

Agile software methods have developed over the last 15 years and the term was officially published with the Agile Manifesto in 2001 [1] and later with the Agile Declaration of Independence in 2005 [2]. The Agile software methods were designed to respond to changing environments and user requirements and they are a response to the heavy traditional software development methodologies [3]. The strategy behind the Agile software methods is to create useful software by following fast design iterations with working software instead of documentation and creating strong customer interaction.

Agile and Lean manufacturing.

Toyota started to use lean manufacturing in the 1950s and Agile manufacturing provided inspiration for the Agile software methods [4]. These techniques (such as Kanban-based scheduling systems [5]) were adapted specifically to the software development industry. With the rapid development in software technology in the last years these advancements could lead to innovative new methods to manage and develop complex multi-disciplinary projects. This paper wishes to investigate if the evolution of Lean manufacturing to Agile software methods and its further adaptation provides new methods that can benefit the seed for that evolution: multi-disciplinary projects. Figure 2 describes this evolution process and this paper focuses on step 4 in the figure.

Agile software methods promote Agile companies

For a company to be agile (to be able to handle change) it needs to incorporate the following principles into its design strategy: everything that can be standardised should be, the processes should be modularised and IT should be utilised to streamline processes [6]. It is even believed that a company's ability to change with respect to workforce, supply chains, market and software is one of the most important ways for a company to be successful, i.e. to perform profitability and sustainably [7].

Agile software methods attempt to achieve this in the software discipline. Pair-programming, which form a concrete principle of the Extreme Programming (XP) Agile software method, has been shown to improve the adherence to coding standards, which improves standardisation of the product and increases code ownership [4]. Modularised code is improved by frequent refractoring tasks (reorganisation of code design) in Agile development. Iterations in code implementation used in Scrum promotes interaction between the customer and the development team, which creates an environment where the product can change with user wishes [8]. IT solutions (such as http://www.asana.com and Git http://git-scm.com/) and the ever increasing power of the cloud, provide advanced tools for developing efficiently between dispersed teams using Agile software methods.

Thus the use of Agile software methods enables a software company to change with developing technologies and markets, which creates an Agile (software) company.

Agile software methods applied to multi-disciplinary projects

Initially it was believed that Agile was only efficient for small software teams working on pure software problems, but the use of Agile has spread into embedded companies across Europe [9]. The hardware and software present in an embedded system are often developed concurrently. The creation of the hardware components in embedded systems entails production processes, research, development and electronics. Therefore it is a multi-disciplinary complex design process. Agile software methods are thus spreading beyond pure software companies.

Pure software products differ from basic manufacturing because there is neither an assembly phase nor the physical logistics of the raw materials or components [7], although the increasing complexity of software systems can create significant overhead when deploying applications (server configuration etc.). Software changes often do not require any physical changes and the software prototype can often develop into the finished product. Physical costs for software development can often be minimal compared to the acquisition of physical production facilities in production environments.

Agile software methods focus on satisfying the customers need and effectively managing team members, but often lack tools to deal with longer-term strategy and large scale organisation factors [7]. Agile software methods attempt to create products that fulfil the customers needs, working on the premise that these needs are not fully defined at the beginning of the project. They would rather adapt to the changing environments than anticipate the needs from the beginning.

Large scale complex multi-disciplinary projects often have strict constraints on what needs to be created, these restraints can be physical by nature or caused by regulatory constraints. These sort of projects need to have a structured plan to be able to anticipate and accommodate these constraints. A systems engineering process is often applied, but these activities are generally designed for pre-specifiable, deterministic (complete and traceable) requirements and schedules [5]. With the rapid advancement in markets, requirements and schedules this is often not viable in the current industrial environment.

The radical nature of Agile software methods often leads to conflict between advocates of the methods and more structured systems engineering frameworks.

With the advancement of computer power and software capabilities, design teams are now adopting model driven design strategies that use computer aided design and rapid-prototyping techniques such as 3D printing. Virtual testing can be performed between complex subsystems in simulated environments combined with hardware in the loop components that can gradually replace virtual components. This creates an environment where it is possible to implement fast design iterations with working models or prototypes, even in complex multi-disciplinary projects.

Thus the development of mechanical systems, or hardware systems is approaching the same level of versatility observed in software development. Adapted models of Agile software methods should be able to be applied to a broader range of development areas as a consequence.

Agile software methods started developing around the same time when Agile manufacturing started finding prominence [7] and Agile manufacturing with Lean manufacturing were adapted to form the current Agile software methods. It is proposed that through the adaptation and advancement that had happened in the Agile software methods area, new insight can be gained and this insight adapted and applied to multi-disciplinary projects. Figure 2 gives a graphical description of this process. The following section will give a concrete description of some Agile methods (step 3 in Fig.2).

AGILE BACKGROUND

The two most well known Agile software methods are Scrum and Extreme Programming (XP), we use these two methods to develop an implementation for the test bench system within the SOFIE project. A brief description of each of the techniques will be discussed.

Scrum

Scrum has been developed for managing the software development process in a volatile environment [8]. It provides a series of guidelines on how to manage the development process, including project management. The core characteristics of Scrum is frequent contact between the developer and stakeholders with features implemented in small sprints. This fosters communication between stakeholders [4].

The Scrum process has a number of roles assigned to members of the team. The Product Owner is responsible for the product, for the business and technical requirements. The Product Owner helps determine what the requirements for the project are, these are features that he/she would like to get implemented. These features create the Product Backlog.

The team is responsible for developing the functionality. They figure out how to implement the features that are provided by the Product Backlog. The Scrum Master is part of the team and is responsible for insuring that the team does what it needs to do, makes sure that Scrum is understood within the team and facilitates communication with the Product Owner.

These different roles are used to implement the Scrum process. A brief skeleton of the Scrum process is now provided in Fig. 3 and further elaborated here:

1) **Product Backlog:** List of features that need to be implemented to create the functionality that is required from the project.
2) **Sprint:** A design iteration where the team works on the chosen features. The features are added to the Sprint Backlog and the team organises how the work is dispersed amongst them.
3) **Daily Scrum:** This is where people describe what has been done, what will be done and problems that occur. The problems are not discussed here, the Scrum master then notes and attempts to get a process in place to fix the problems after the meeting.
4) **Demonstration:** At the end of a sprint the functionality is demonstrated to the stakeholders and potentially other Scrum teams. The purpose is to demonstrate completely functional features.
5) **Review:** The current Sprint is reviewed, the team reorganised and optimised. The Product Backlog is

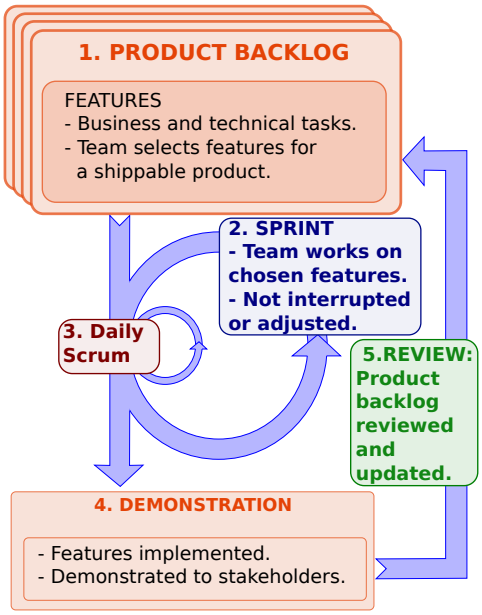

Figure 3. Scrum Agile development [10].

updated and adapted to match the (changed) goals of the project and then the cycle starts again.

Extreme Programming (XP)

XP can be summarised as a collection of best practises to enable effective software development [4]. XP is based on the four values: communication, simplicity, feedback and courage [11]. This leads to the principle best practises that should be followed when working on a software project which are the essential characteristics of XP [11]:

1) **The planning game:** Before each cycle the stories (which are task/features that need to be implemented from a business or technical perspective) are described, an estimate is put to their time and then organised in priority.
2) **Small releases:** Design the minimal amount of features that will be possible to make the next release useful. Thus make small useful changes between releases and make sure it is useful at each stage.
3) **Metaphor:** Creates a coherent story wherein everyone can work, the business people and the technical people. The metaphor provides a simple way for people to understand the basic elements and their relationships. Every story that is done in the planning game must relate to the Metaphor.
4) **Simple design:** A simple design should: pass all the tests, have no duplicate logic, state its intention to the programmers and have the fewest possible classes and methods (programming tools used in Object-Orientated programming). One should add what is needed when you use it, do not design to far for the future and erase everything that is not completely useful.
5) **Testing:** All codes should be unit tested, to gain confidence that it works and ensure that it performs the functions that it should perform. Unit testing is functional testing, all the different components of the software system are tested independently.
6) **Refractoring:** Refractoring is reorganising the code/design after it is implemented to see if it can be done simpler. Refractoring helps to design for adaptability by creating a simpler design whereby it is easier to add features to the design at a later stage.
7) **Pair programming:** Two people work on code with one computer, keyboard and mouse. One programmer implements the code and the other one thinks strategically about what is the best way to implement the feature. Roles and partners are often changed.
8) **Collective ownership:** By changing partners and tasks collective code ownership is improved. This means that each person gets to understand different parts of the system and is confident to improve them. Not one person owns one piece of code, improving redundancy in the workforce and the ability to evolve the system to what one needs.
9) **Continuous integration:** Each piece of code is individually tested, these tests are then added to the complete system. A continuous integration testing environment is created to test all new features against the system. Continuous integration is often implemented on a dedicated testing server.
10) **40-hour work week:** Designing and programming are intense tasks, people are only able to work productively a certain amount of hours per week. The project should be managed so that people work a reasonable amount of hours, if overtime occurs it should then be followed by a time when people can work less.
11) **On-site customer:** The person who is going to use the system should be involved in the design process.
12) **Coding standards:** Standards must be used to implement code, share data and document work. This creates a system where anybody works on anything and its easier to adapt and develop code,

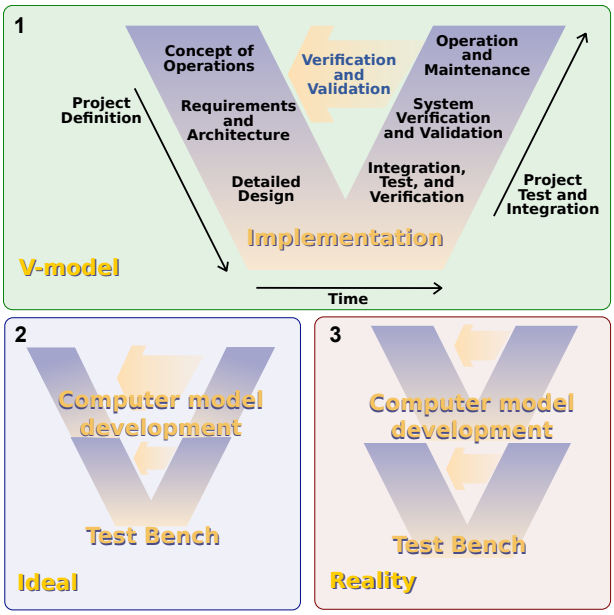

Figure 4. The V-model for systems development for the test bench in the SOFIE project. (Image adapted from [13].)

even if you have not worked on it before.

A description of XP best practises and the Scrum management technique have now been described. There is extensive literature available on both topics in [11], [10], [12]. The implementation of the test bench project is described in the next section followed by the adaptation of Scrum and XP to the test bench case study.

BICYCLE STABILITY TEST BENCH

The bicycle stability test bench forms a core sub-system of the SOFIE project. The computer model developers need data to validate the computer model, the rider behaviour researcher needs to be able to perform experiments to determine this behaviour and the IAD developers need to be able to independently test their products. Therefore all the stakeholders in the project have an interest in the use of the test bench. These stakeholders use the test bench in different sub-systems of the project with different requirements at different periods during the project life-cycle.

The V-model for systems development is often used to describe the research and development process in complex projects (Block 1 in Fig. 4).

The design of the test bench is complex because there is a lead-time in the creation of the test bench: mechanical, electrical and software components need to be created, but the other partners in the project need to use the test bench to perform experiments and validate computer models.

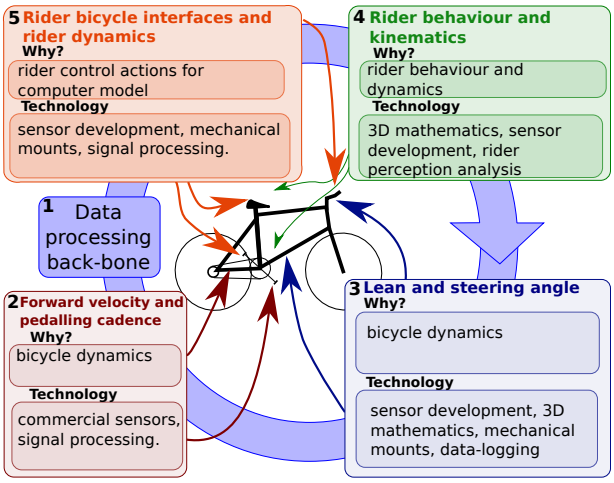

Figure 5. Bicycle stability test bench design overview.

The ideal situation is if the computer model could be developed to a stage where it needs to be validated and verified (as in block 2 in Fig. 4), then the test bench could be created and the computer model subsequently validated.

In reality the computer model and the test bench need to be developed concurrently (block 3 in Fig. 4). Complexities arise because the computer model has to give input to the test bench before the test bench is created, but the test bench needs to be created to help in the design of the computer model.

This is equivalent to the building of the frame and electrical infrastructure in large scale developments. The infrastructure needs to be completed before the development can be used (or created) but also needs input from the development to design the infrastructure. The SOFIE project does not have to deal with sub-contractors, has limited issues with logistics co-ordination but does have stringent customer deadlines to obtain funding.

Thus the test bench within the SOFIE project is comparable to larger scale complex projects with regards to the dependencies between sub-systems and partners but lacks similarities in other areas due to the relatively small size of the project.

Design

The design of the bicycle stability test bench is shown in Fig. 5. The multi-disciplinary aspects of the project and complexity of the system will be highlighted.

The design has five different sub-systems each shown in its own coloured block in Fig. 5 and described next:

1) A **'Data processing back-bone'** is a software and hardware package (https://github.com/agcooke/Sofie-HDF-Format) which handles the conversion of data from different systems to a common format. Software engineering and data processing form part of this subsystem.
2) The **'Forward velocity and cadence'** subsystem is created using commercial sensors which use the ANT+ (http://www.thisisant.com) wireless communication protocol. This involves embedded software development and sensor mounting systems.
3) The **'Lean and steering angle'** is measured using modern Inertial Measurement Units (IMU) and possibly a potentiometer based sensor system. This involves mechatronic devices, sensors, signal conditioning, signal processing and advanced 3D mathematics.
4) The **'Rider behaviour and kinematics'** will include a video camera system and an accurate rider kinematics measurement system. The measuring system could use IMUs, a Microsoft Kinect or video processing to determine the rider kinematics.
5) The **'Rider bicycle interfaces and rider dynamics'** are needed to validate the computer model and determine how a rider controls the bicycle. We are going to create force sensors on the handlebars (where the rider grips the handlebar), on the pedals using an off the shelf fitness product and on the seat. Thus mechanical and electronic components are used in the creation of this sub-system.

The design of the bicycle stability test bench has been conceptualised and conveyed to the group and implementation has begun. A good project management strategy and development framework is required to ensure that the test bench is created on time and is useful to the rest of the stakeholders in the project. The next section will give a a concrete description of the implementation of the Agile software methods to the bicycle stability test bench.

Implementation for the test bench

The XP and Scrum Agile software methods are adapted for the test bench part of the SOFIE project. The development team on the test bench part of the project is small and this influences how it will be implemented. This team is now described:

1) The lead author works full-time on the test bench part of the SOFIE project and has to divide his time between different disciplines: software development, mathematical algorithm development, sensor design and ordering, supervising students and other research duties.
2) Temporary bachelor and masters students working on systems within the project.
3) Supervisors, computer model developer, human behaviour researcher and IAD product developers as the customer.

This creates a scenario where the test bench developer gets input from different sources. The sources determine how the test bench should be developed. Thus many tasks or features are added to the system at different times from different sources. These need to be implemented by the test bench team, a system where work must be pulled into the project is created and not only pushed into the project by a schedule.

A 'pull' type system is believed to be a good way to enable efficient completion of these tasks by minimising context-switching and managing work load [5]. System engineering processes need to run in parallel to this 'pull' system to organise the tasks and features to make sure the milestones are reached on time. This is a requirement for the rest of the partners and strictly important for the funding organisations.

All these factors make it difficult to implement Scrum directly to the test bench development. The effective creation of communication with partners and team members, the project management benefits of Scrum and the best practises of XP are applied to the project as a case study to determine if it can be used effectively in such a project. A concrete description of how this is applied will now be given.

Project management implementation

Figure 6 is used to explain the development method that is used for the creation of the test bench.

Milestones for the project are created using systems engineering principles, a brief functional analysis and requirements engineering forms part of this process. These take into account the needs of the different partners, the requirements for funding, the need for academic output and the time estimated to complete different tasks.

These milestones then serve as important demonstrations to stakeholders within the project. The goal is to make sure that the partners in the project can see what is being developed through working demonstrations and not only via static presentations. The milestones are re-evaluated at each demonstration and adapted to changes in requirements or implementation time, creating a project that uses just-in-time and continuous planning. The user feedback and requirement revaluation attempt to reduce the importance of the upfront functional analysis and requirements engineering because the method wishes to create a project that adapts to what is needed, as these needs change.

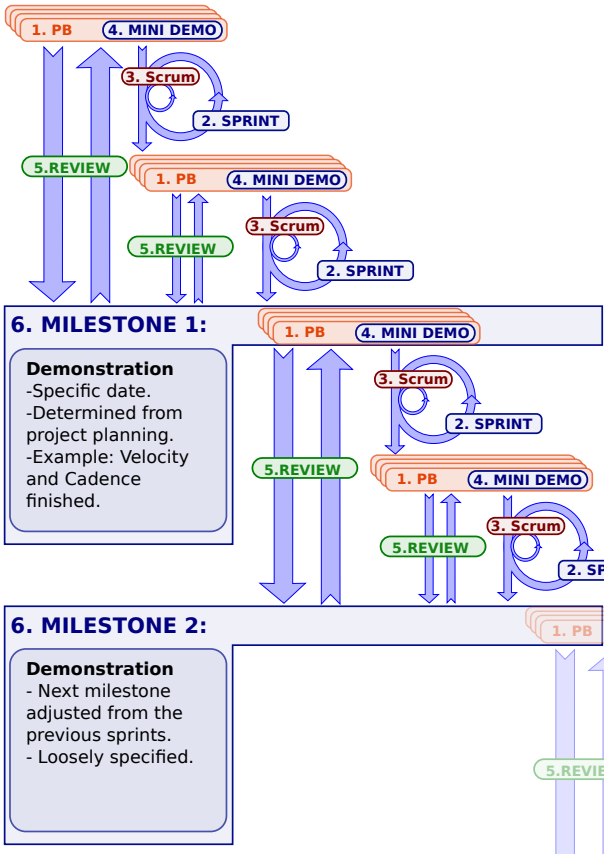

Figure 6. Scrum-Milestone cycle for the test bench.

The Scrum development model is used between these milestones, adapted to the nature of the team structure within the test bench project. The Scrum meetings are organised as follows:

1) **Fortnightly Scrum** (Analogous to the 'DAILY Scrum' from Block 3 in 3 and 6):
2) **Monthly 'Demonstration'** (Block 4 in Figure 3 and 6):
   a) Important stakeholders present.
   b) Product Backlog updated.
3) **'Milestone Demonstration'** (Block 6 in Figure 6):
   a) All stakeholders present.
   b) Same as the 'Monthly Demonstration', but the Milestones are adapted and planned.
   c) Organised every two to three months.

The time-span between meetings will be optimised as the project develops. As this is not a pure software project it may not be clear what features or tasks are. A few examples will be given to illustrate what they could be:

- If a mounting device for a sensor needs to be developed, the creation of a CAD model of the clamp could be a task or feature that is demonstrated. It is then demonstrated with a completed CAD model, possibly 3-D printed or printed on paper. User feedback, demonstrations and experiments are designed to test the functionality. These are used to show that it fulfils its requirements.
- A sensor needs to be created to measure the steering angle, possibly using a potentiometer. A feature could be a mounting of this potentiometer on a mock-up bicycle and a demonstration of the sampling of the voltage that the potentiometer uses at different angles.
- Sensor data files need to be imported into the data processing system, a feature can be demonstrated by performing the import of this data into the file and displaying it to the people viewing.

These features should be added to the 'Product Backlog' and an estimate of the time taken to do it given. There are two time units per day to ensure that the tasks cannot be too small and to allow the team member to work on unexpected tasks that crop up along the way. Having only two time units per day also helps reduce the amount of context-switching that occurs.

Best practises implementation

The best practises principles from XP is used within the project. The 'Planning game', 'Small Releases', 'Metaphor' and 'On-site customer' from XP are covered by the Scrum development cycle, but the implementation guidelines within the project still need to be defined.

The 'Simple design' XP principle is applied to the project. The creation of modularised components aids in this process. These modularised components are released to the public (e.g. open source software or hardware) to facilitate adoption and recognition of the components. Standards from industry or academia are used wherever possible within the project. The principle of not re-making the wheel, re-using software or hardware from diverse sources, is used wherever possible.

Test driven design is used throughout the project to make robust usable sub-systems. If a hardware component is created the function is tested within laboratory environments. Unit tests are used within the software components. Peer review from open source parts of the project and the Scrum demonstrations will aid in the refractoring of the code and hardware. Pair programming will not form a big part of the development process, because the team size is small, but collective code/product/system ownership is encouraged. The demonstration procedure is used to make sure that all the components are integrated successfully. The '40-hour

work week' is encouraged with workload requirements taking this into account. A coding/working style is recommended and implemented.

People who come into the project to do assignments (e.g Masters students) will work on their own Scrum-Milestone cycle if their work is separate to the test bench team work, or will slot into the Scrum-Milestone cycle for the test bench.

The description of the Scrum-Milestone method is currently being implemented and used within the test bench project.

Discussion and future work

The description of the bicycle stability test bench has been described. It has been shown that the test bench project is a relevant case study for larger scale industrial problems.

Agile software methods are already used in complex embedded system development[9]. It has been shown that there are inherent differences between pure software development and multi-disciplinary projects in terms of physical restraints, but these are slowly diminishing with the advancement of computer aided design, hardware in the loop simulations and rapid-prototyping.

This allows the modern techniques used within Agile software methods to be applied to multi-disciplinary projects, mechatronic and embedded systems. It is argued that Agile software methods can help solve many of the inherent difficulties that form a part of multi-disciplinary projects.

A description of Agile software methods has been provided, followed by a concrete description of how to apply these principles to a multi-disciplinary project using the Scrum-Milestone approach. It has been shown that it is possible to adapt SCRUM and XP to a complex multi-disciplinary project.

Future work

Research into the effectiveness of Agile software methods effectiveness is gaining substance but a lot is still not known [8]. This paper describes a new application of this technique and Agile software methods in general to a different domain. The research domain of Systems Engineering is still a young field and thus often relies on observational research [14]. We need to make sure that this research adds substance to the field by providing a good method to evaluate the effectiveness of the Scrum-Milestone adaptation outlined in this paper. We need to make sure that it is [14]:

1) **Technically feasible:** It can be implemented and works.
2) **Technically valuable:** It works better compared to something else.
3) **Practically feasible:** Works across all the stake-holders and not just within the SOFIE project.
4) **Practically valuable:** Review and analyse if it has value for the entire SOFIE project.

Criteria and methods to assess these characteristics need to be developed and presented. The test bench part of the project is set within the University of Twente. Student projects can be used as testing grounds for the method development, and there are already two students working in such a fashion within the project.

The interaction with the rest of the partners in the SOFIE project provides insights on how the approach will work in large scale industrial problems. These two characteristics of the projects structure provide a good opportunity to create a comprehensive validation environment of the proposed method. This will give insight for further research into Agile software methods and particularly their application to multi-disciplinary projects.

References

[1] J. Highsmith and M. Fowler, "The Agile Manifesto," *Software Development Magazine*, vol. 9, no. 8, pp. 29–30, 2001.

[2] "Agile Declaration of Interdependence," 2005. [Online]. Available: http://pmdoi.org/

[3] J. Erickson, K. Lyytinen, and K. Siau, "Agile Modeling, Agile Software Development, and Extreme Programming," *Journal of Database Management*, vol. 16, no. 4, pp. 88–100, Jan. 2005.

[4] T. Dyba and T. Dingsoyr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology*, vol. 50, no. 9-10, pp. 833–859, Aug. 2008.

[5] R. Turner, D. Ingold, J. A. Lane, R. Madachy, and D. Anderson, "Effectiveness of kanban approaches in systems engineering within rapid response environments," *Procedia Computer Science*, vol. 8, pp. 309–314, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2012.01.065

[6] C. Verstraete, "Planning for the unexpected," *Manufacturing Engineer*, vol. 83, no. 3, pp. 18–21, 2004.

[7] P. Kettunen, "Adopting key lessons from agile manufacturing to agile software product development-A comparative study," *Technovation*, vol. 29, no. 6-7, pp. 408–422, 2009.

[8] P. Abrahamsson, N. Oza, M. T. Siponen, T. Dingsø yr, T. Dybå, and N. B. Moe, *Agile Software Development*, T. Dingsø yr, T. Dybå, and N. B. Moe, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. [Online]. Available: http://www.springerlink.com/content/m472r5m32r85t2t6/

[9] O. Salo and P. Abrahamsson, "Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum," *IET Software*, vol. 2, no. 1, p. 58, Feb. 2008.

[10] K. Schwaber, *The Enterprise and Scrum*. Microsoft Press, 2007.

[11] K. Beck, *Extreme Programming Explained: Embrace Change*, ser. The XP Series. Addison-Wesley, 1999.

[12] H. Kniberg, *Scrum and XP from the Trenches*, 2006.

[13] "V-model for systems development." [Online]. Available: https://en.wikipedia.org/wiki/File:Systems_Engineering_Process_II.svg

[14] G. Muller, "Systems Engineering Research Validation," 2011. [Online]. Available: http://www.gaudisite.nl/SEresearchValidationPaper.pdf