Agile Hardware and Co-Design

Timo Punkka
timo.punkka at ngware.eu
Embedded Systems Conference 2012, Boston, ESC-3008

Abstract. *New product development seems to be more challenging than ever. The products are getting more complex. Product lifecycles are shorter, and the amount of change is staggering. Software industry has tried to answer these challenges with Agile software development. Agile software development relies on fast paced iterations and continuous re-planning based on feedback and learning from the past iterations. Agile software development methods are also making their way into embedded software development. However, embedded software development usually has strong dependencies to other engineering disciplines. Product development researchers have presented the idea of applying the same, Agile, principles to non-software projects. Nevertheless, evidence from real-life projects remains scarce. This technical paper presents author's observations from the industry setting. These observations support the idea that Agile development benefits also the development in other engineering disciplines and actually proposes that Agile development must be applied at system development level in order to harness the full potential.*

Introduction

Embedded system development requires combining views from multiple perspectives, such as embedded software, electronics and mechanical design perspectives. Different views have strong dependencies. This is often called co-design. The objective is to design the optimized product, but the rules for the optimization game keep changing. Change can be internal, for example changing the opinion on hardware/software –partition, or external such as change from competition or technology. The project organization needs to cope with changes from all directions throughout the project. Furthermore, the complexity of the products we develop seems to double with each generation, making tradeoff decisions harder.

Traditional development process models have been based on the idea that system requirements can be fully known at the outset of the project, that it is possible to create a detailed project plan, and that there is a chance that this plan can be executed. The above mentioned complexity and the amount of change make this idea ridiculous. The need for change in an embedded system development project is unavoidable. This is best managed with an empirical process designed for continuous collaboration and refinement of design and plan between different perspectives. Agile methods are developed with this in mind.

Agile methods are fast-paced, iterative, people-centric, business value focused, adaptive to change, reflective, learning driven, lightweight and low ceremony. Agile methods acknowledge and emphasize the potential of empowered teams. Communication overhead often talked about in traditional project management literature is limited by keeping the number of people on the team low (less than 10), co-locating everyone in the team, and focusing on "high band width" communication mechanisms[1], preferably face-to-face over a whiteboard. Agile development aims at both tackling the problems identified with traditional work, and also answering to challenges of the future such as the increasing rate of change and the amount of complexity in products we develop. An agile methodology focuses on results: It is simple, it is responsive and self-adapting, it stresses technical excellence, and it emphasizes collaborative practices (Highsmith, 2002). Early in the millennium Agile methods were considered to be suitable for small and non-critical software projects. Since then knowledge has been created in large distributed programs (Hossain, Babar and Paik, 2009), critical and real-time embedded systems, and even non-software development (Reinertson, 1998, Highsmith, 2004, and Smith, 2007, Reinertsen, 2009, Cooper and Edgett, 2009).

This technical paper presents ideas on how to adopt knowledge from Agile software development into hardware development. Ideas are collected from a number of projects. I have seen similar techniques working with fully cross-disciplined teams with embedded software and hardware developers, with globally distributed hardware teams, and in situations where a single hardware team supports multiple product lines each with dedicated software team(s). Now that Agile development is gaining foothold in the embedded industry, the main objective of this paper is to provoke thinking on whether we should continue the adoption also at system level development.

[1]Ambler, Scott, Communication on Agile Projects, http://www.agilemodeling.com/essays/communication.htm

This paper is constructed as follows. The next chapter gives a short introduction to Agile development. The following chapter presents motivation for considering Agile development also outside pure software development. Third chapter discusses the background for making Agile hardware development possible. Fourth chapter gives ideas on practices that can be adopted to make incremental and iterative Agile planning possible. Fifth chapter presents additional benefits from Agile hardware and co-design. The final chapter summarizes the technical paper.

Agile development – a brief intro

In 2001, a group of software professionals got together to discuss about methods to develop software. They wrote down their shared values in agile manifesto[1]:

We are uncovering better ways of developing software by
doing it and helping others do it. Through this work we have come to value:

Individuals and interactions *over processes and tools*
Working software *over comprehensive documentation*
Customer collaboration *over contract negotiation*
Responding to change *over following the plan*

That is, while there is value in the items on the right, we value
the items on the left more.

The authors further defined 12 principles to clarify the values present on cover page:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile development shares the values and principles (Abrahamson et al., 2003), but there is still room for a variety of different agile methods and frameworks. Out of them, Scrum (Schwaber and Beedle, 2002 and Schwaber, 2004) and Extreme Programming (Beck, 1999 and Beck, 2004) are the most adopted in the industry. Scrum is a project management framework illustrated in Figure 1. In Scrum, work is broken down to fit short increments called Sprints. The work to be done in a given Sprint is planned in a Sprint Planning Meeting. In this meeting, work is pulled from a Product Backlog, and moved into team's Sprint Backlog. The work planned for a Sprint is managed by the Scrum team. A Scrum team consists of 10 or less cross-disciplined members. Ideally, the team has all the skills necessary to complete the Sprint. During the Sprint, the team has a Daily Scrum Meeting to synchronize the information. The Product Owner is responsible for prioritizing the Product Backlog containing all the work to be done in a project. The Scrum Master is responsible for keeping the process fit and coaching the team into continuous improvement. At the end of the Sprint, a Sprint Review meeting is held to demonstrate the team's achievement to all stakeholders for gathering feedback. Between Sprints, the team holds a Retrospective meeting to gather improvement ideas. Scrum does not give

[1] http://www.agilemanifesto.org

guidance on engineering practices. For this reason, teams often supplement it with practices from other methods, such as Extreme Programming. (Schwaber and Beedle, 2002) express this as "if practices were candy the Scrum is the wrapping paper for candy".

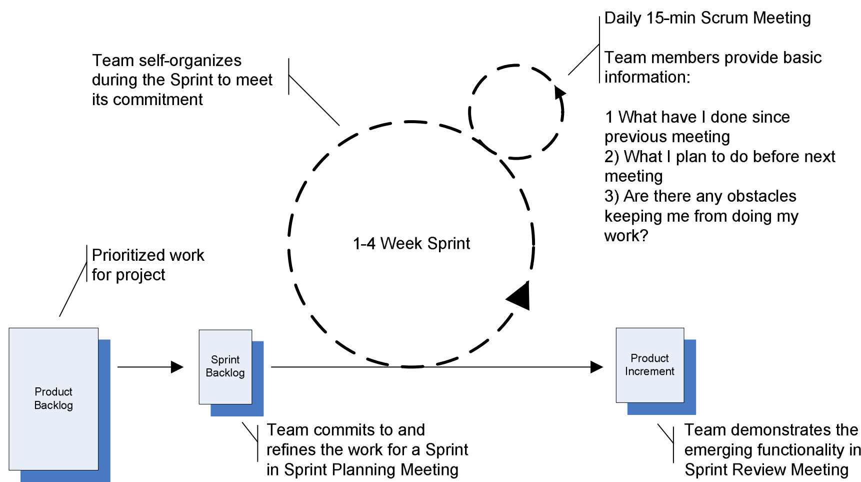

Figure 1. Scrum Framework (Schwaber and Beedle, 2002 and Schwaber, 2004).

Why should agile be considered for hardware and co-design?

> "*Each element in the system is ignorant of the behavior of the system as a whole, it responds only to information that is available to it locally. This point is vitally important. If each element 'knew' what was happening to the system as a whole, all of the complexity would have to be present* in that element*.*"
>
> *Cilliers, 1998*

> *"If each subsystem, regarded separately, is made to operate with maximum efficiency, the system as a whole will not operate with utmost efficiency."*
>
> *Skyttner, 2001*

The development of an embedded system integrates multiple perspectives from different engineering disciplines. It is impossible to optimize the whole product with knowledge of single perspective only. Let's take a naïve example. If the choice of the CPU was solely left as a responsibility of electronics designers, the resulting choice would be the cheapest processor they can find. When we involve other disciplines as well, then the selection parameters will include processing power, physical size, and alike. When we make a change in one area, it will likely affect other areas. So, continuous refinement and planning is needed, and this activity needs to involve everyone. A positive side effect of this collaboration is shared understanding of direction, which in turn enables single consciousness and empowered collective decision making. It is exactly the empowered front-line designers who are in the best position to make continuous refinement decisions.

But wouldn't it still be possible to together design everything up-front?

When we acknowledge that requirements are lacking or are vague at best, up-front planning seems completely irrational as the knowledge to base the plan on is non-existent. Having only vague requirements is not uncommon at all according (Thomke and Reinertson, 1998);

"One of the authors has worked with hundreds of product developers and has yet to find a single project in which the requirements remained stable throughout the design. Surveying more than 200 product developers over the past five years, he found that fewer than 5% had a complete specification before beginning product design. On average, only 58% of requirements were specified before design activities began. The inevitable result is changes."

New product development always involves at least a certain amount of learning and discovery. Each product, each situation, and each development organization is different. To make situation even more challenging, things do change during the project. Plan-driven process models enforcing the practice of defining the whole project up-front, and then sticking to that plan, seem less than optimal. In contrast, the requirements, the design and the process itself, need to emerge in parallel. Agile development is aimed to be a better fit for emergent work. Agile development has been strongly focusing on software development. Can it be used in other development areas?

What makes Agile hardware development possible?

We need to learn in projects. Trial and error is the most effective way of learning. Furthermore, the faster the cycles of trial are, the more efficient is the learning. Developing small, focused, experiments on partial solutions makes it possible to start learning early, and efficiently, in the hardware development. The practice is called up-front, or front-loaded, prototyping. Up-front prototyping is different from traditional prototyping in that it shifts the objective of prototyping from validating into experimenting. Traditionally prototypes have been used as a means of final validation of something that is believed to be correct. Up-front prototyping is done with much less confidence on succeeding. Prototype can be used to learn how far we are from working solution, for example. What is traditionally negatively called rework, or scrap work, is now considered valuable and mandatory learning. Today's design tools and fast prototyping technologies enable the shift in thinking.

Technology is changing the game of prototyping in many ways. Technology that goes into the final product gets more flexible. The technology that is used in development makes changes cost less. Technology for physical and virtual prototyping lowers the cost in terms of material and labour. We can see technology lowering the cycle cost of prototyping at all stages (see Figure 2).

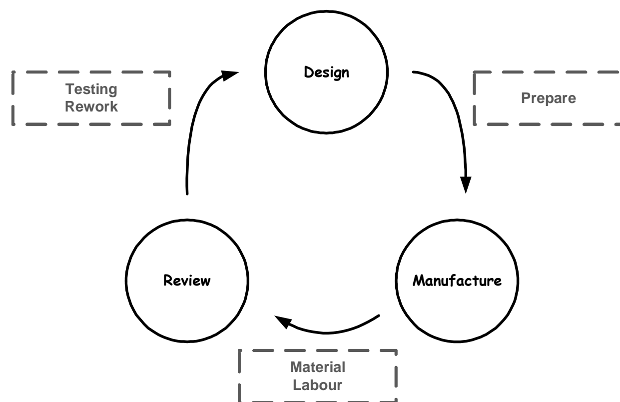

Figure 2. The cycle cost of prototyping (modified from Smith, 2007).

Regardless of advancement in technology, prototyping still costs money. When we add the extra cycle cost from prototyping to the project, it seems that we are making the project cost more. However, this front-loaded cost increase works as an insurance. We buy down the huge risk at the end of the project, which traditionally has been left lurking until the prototyping round near to the (thought) end of the project. With this practice, we verify each assumption early on, and lower the total penalty from false assumptions. This is illustrated in Figure 3a) and b). In 3a) we consider only added monetary cost. The first bar shows the traditionally estimated project cost. Second bar shows the impact of adding the cost of two additional up-front prototyping rounds. Third bar shows the typical realized cost when all the risk is left until the end. A horrible example is a mistake in a mold that is only discovered in the end. Rework of the mold at

this stage is very expensive and time consuming. If we believe that costly mistakes like this in the end can be avoided by increasing up-front prototyping, then the cost of early prototypes is minimal.

When we take calendar time into account, it makes things even more interesting. This is illustrated in Figure 3b). By lowering the cycle cost of prototyping through continuously seeking new technology, improving own processes, and the relationship with prototype suppliers, the added calendar time from prototyping can be minimized. This is seen in the second bar. Uncontrolled and unexpected rework at the end on usually means long, unpredictable delays. The team can't do anything while waiting for the new prototypes. The prototype supplier wasn't expecting these new rounds either. They have other work lined up which adds to the delay. This unpredictable delay causes extra cost in other parts of the organization as well. Again, if we can reduce this uncertainty by investing in up-front prototyping, we can make the whole organization work better.

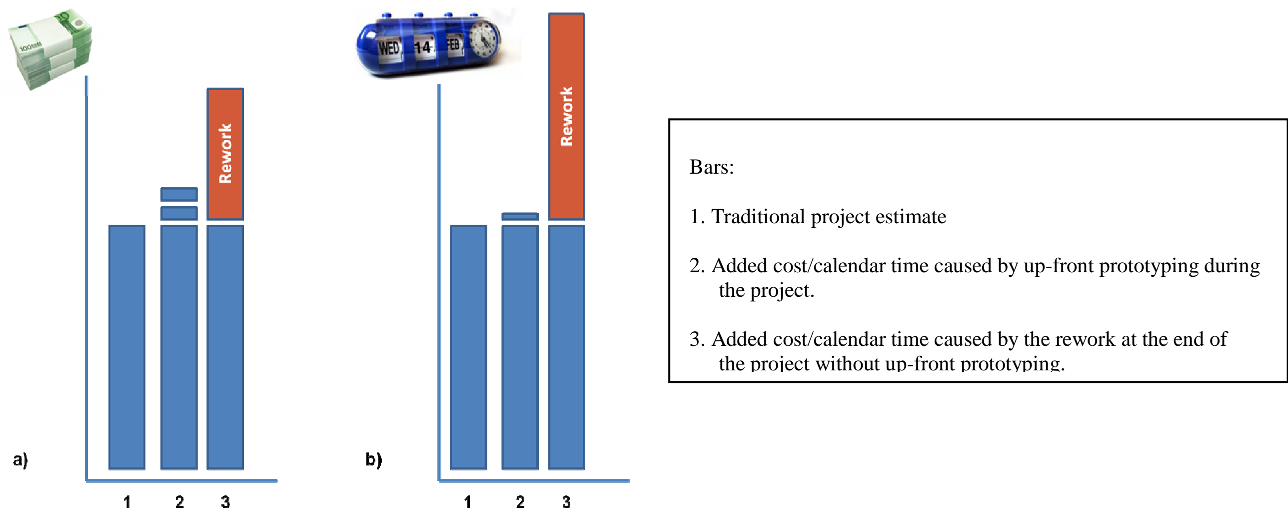

Figure 3. The (imaginary) true cost of prototyping; a) monetary cost and b) calendar time

Iterative planning in Agile hardware development

At first, hardware development with small experiments may sound like a strange idea. After all, you need the whole schematic to be able to order the PCB spins, right? Luckily, a technique from software development called vertical slicing can be adopted to hardware development as well. Vertical slicing in software development means that the development proceeds with features that require design effort at all architectural layers; user interface, business logic and persistence layer for example. These software architecture layers can be compared to the different engineering disciplines involved in hardware development, for example electronics, printed circuit board layout, mechanics, industrial design, and design for manufacturing. We can design the product involving all these principles in parallel, one small uncertainty at the time. In an Agile planning context, the uncertainties become the equivalent of the software features. The development team can use a number of different prototyping technologies in tackling the uncertainties, and to report progress;

- Schematics
- 3d models
- Simulation
- Bread board prototypes
- Re-usable generic prototypes
- Evaluation boards
- Partial prototypes
- FPGA
- 3d printers

Figure 4 illustrates how a design of one product proceeded using a combination of different prototyping technologies. The different sub-systems evolved from evaluation boards, through breadboard prototypes to partial prototypes. Finally, all sub-systems where integrated into the final product.

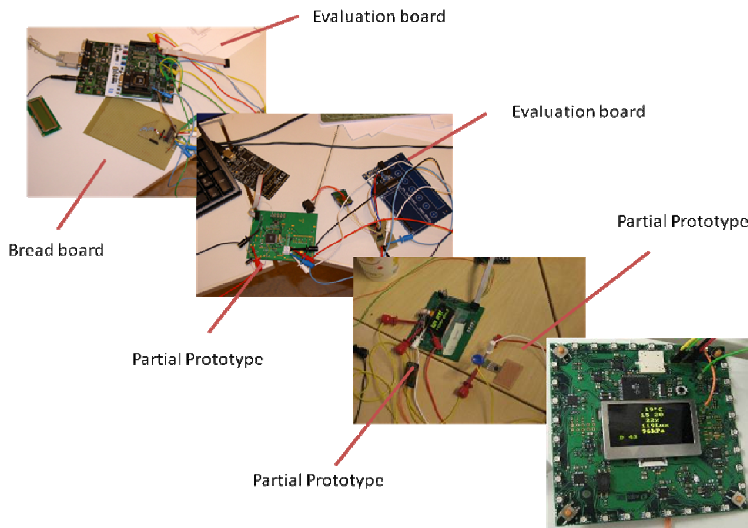

Figure 4. Using different prototyping technologies during the development.

Additional benefits from Agile hardware and co-design

Adopting iterative and incremental planning to the system level development enables the organization to achieve an organization wide rhythm, or cadence. Up-front prototyping and vertical slicing bring the hardware development closer to the Agile software development. Hardware development planning and scheduling can follow the rhythm of software development. Having a steady cadence across the organization greatly simplifies the re-synchronization of planning at different levels, from operational and execution to strategic planning. Achieving organization wide cadence brings many benefits for the whole organization, such as;

- Focus on most important issues
- Amplified learning
- Reduced variability
- Simplified scheduling and (re-)synchronization

Another example of side-effects is a positive reinforcing cycle when teams start using Agile co-design involving software and hardware developers (Figure 5). When engineers from different disciplines begin to work together they will innovate new ways how to move testing forward. An example is a technique called hardware unit testing. Typically this means semi-automated testing of software-hardware boundary. Instead of validating hardware through real application, the application provides a command line interface that can be used to drive different hardware drivers. This can be as simple as setting outputs, or more complex, for example driving different patterns to LCD to verify color control. When testing is more automated the cycle cost from prototyping lowers, and this reduces the resistance among the hardware designers. This helps in introducing shorter iterations. The differences between engineering disciplines diminish, which further enforces the collaboration.

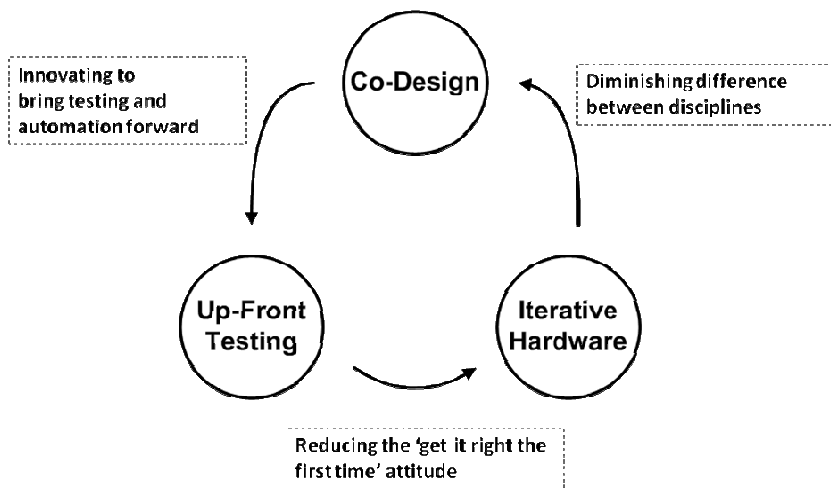

Figure 5. The positive reinforcing cycle of Agile co-design.

Summary and discussion

Product development is changing a lot. Changes happen in multiple dimensions all the time. This means that up-front planning is in practice impossible. Empirical and explorative processes are more suitable for this type of work. Agile methods are fast paced incremental and iterative processes developed in software industry. Agile methods are more and more in use for embedded software development as well. Embedded software development plays a growing role in many system development efforts, bringing Agile development close to hardware development in co-design projects. This paper presented an idea of adopting the values and principles, even practices, from Agile software development into development in other engineering disciplines.

Up-front prototyping is a term used to describe a shift in motivation to use prototyping. Traditional prototyping is used for final validation of something that has already gone through thorough analysis. Up-front prototyping is used to experiment with something, to learn. Technology brings this new approach available. Technology brings new tools that make experimenting with prototypes faster and cheaper. On the other hand, the technology used in our new products makes designs more flexible and allows the experimenting to continue further.

Vertical slicing is a practice used in software development. It means that the software is developed feature by feature in such a fashion that each new feature requires development in multiple, if not all, architectural layers of the system. Hardware development can learn from this idea by considering engineering disciplines, such as schematics and PCB layout design, equivalent to the software architecture layers. We can tackle the identified uncertainties one-by-one by developing a partial solution engaging all engineering disciplines.

Cadence means organizational rhythm. In practice cadence means that certain events in an organization happen at fixed intervals. When the rhythm is known to the whole organization it makes all the planning, scheduling, re-planning, and re-synchronization of schedules, much more simplified. By identifying the uncertainties and tackling them in prioritized order, we have an analogy to feature-driven development familiar in Agile software development.

This technical paper gives ideas on how to adopt the learnings from Agile software development to hardware development. It further suggests that doing so brings many benefits to the organization, such as better optimized designs and simplified planning and scheduling across the organization.

In the future more evidence is needed, and more hands-on guidance on working practices is needed. The economy of prototyping is an interesting area for studying. More experience on moving the testing forward in hardware development is obviously also required. Software and hardware developers collaborating throughout the project has shown promise to be able to close the gap in knowledge.

Reference

(Abrahamson et al., 2003) Abrahamson Pekka, Warsta Juhani, Siponen Mikko T. and Ronkainen Jussi, New Directions on Agile Methods: A Comparative Analysis, Proceedings of the 25th International Conference on Software Engineering (ICSE'03), 2003.

(Beck, 1999) Beck, Kent, Extreme Programming Explained: Embrace Change, Addison-Wesley 1999.

(Beck, 2004) Beck, Kent, Extreme Programming Explained: Embrace Change, 2nd Edition, Addison-Wesley Professional, 2004.

(Cilliers, 1998) Cilliers, Paul, Complexity and Postmodernism: Understanding Complex Systems, Routledge, 1998.

(Cooper and Edgett, 2009) Cooper, Robert G. and Edgett, Scott J., Lean, Rapid and Profitable New Product Development, Booksurge Publishing, 2009.

(Highsmith, 2002) Highsmith, Jim, Agile Software Development Ecosystems, Addison-Wesley, 2002.

(Highsmith, 2004) Highsmith, Jim, Agile Project Management: Creating Innovative Products, Addison Wesley Professional, 2004.

(Hossain, Babar and Paik, 2009) Hossain, Emam, Babar, Muhammed Ali and Paik, Hye-young, Using Scrum in Global Software Development: A Systematic Literature Review, 2009 Fourth IEEE International Conference on Global Software Engineering, 2009.

(Reinertsen, 1998) Reinertsen, Donald G., Managing the Design Factory, A Product Developers Tool Kit, Simon & Schuster Ltd, 1998.

(Reinertsen, 2009) Reinertsen, Donald G., The Principles of Product Development Flow – Second Generation Lean Product Development, Celeritas Publishing, Redondo Beach, California, 2009.

(Schwaber and Beedle, 2002) Schwaber, Ken, and Beedle, Mike, Agile Software Development with Scrum, Prentice Hall 2002.

(Schwaber, 2004) Schwaber, Kent, Agile Project Management with Scrum, Microsoft Press, 2004.

(Skyttner, 2001) Skyttner, Lars, General Systems Theory: Ideas and Applications, World Scientific Publishing Company, 2001.

(Smith, 2007) Smith, Prestong G., Flexible Product Development, Jossey-Bass, 2007.

(Thomke and Reinertson, 1998), Thomke, Stefan and Reinertson, Donald, Agile Product Development: Managing Development Flexibility in Uncertain Environments, California Management Review, Vol 41, No.1, Fall, 1998.