Agile and Systems Architecting

White Paper Resulting from Architecture Forum Meeting

March 16, 17, 2015, Stevens Institute of Technology, Hoboken, NJ, USA

Edited by:

Dr. Gerrit Muller, Buskerud University College, Embedded Systems Institute

Mr. Eirik Hole, Stevens Institute of Technology

Input was provided by the following participants in the Architecture Forum:

Name	Organization
Kevin Austin	Kongsberg Protech (US)
Tony Barrese	Stevens Institute of Technology
John Bergh	Ingersoll Rand
Nick Bowen	Stevens Institute of Technology
Dave Dawson	TD Ameritrade
Eirik Hole	Stevens Institute of Technology
Bas Huijbrechts	TNO-ESI
Lars Ivansen	Mycronic
Kees Kooijman	Sioux Embedded Systems
Tim Majeski	Lutron
Colin Mellars	Siemens Healthcare Diagnostics
Emil Moholth	Kongsberg Defense

Name	Organization
Gerrit Muller	HBV-NISE/TNO-ESI
Dave Murell	TD Ameritrade
Jurjen Nicolai	Bosch
Mats Rosling	Mycronic
Rolf Siegers	Raytheon
Martin Simons	Daimler
Lou Steinberg	TD Ameritrade (dinner speaker)
Andy Turner	Microsoft Devices Group
Jon Wade	Stevens Institute of Technology
Klaas Wijbrans	Philips PINS
Paul Zenden	Sioux Embedded Systems
Robert Cloutier	Stevens Institute of Technology

Published, January, 2017

1. Introduction

The forum meeting on systems integration emphasized the need for early validation. Agile methods promise early validation, which makes them attractive for systems level architecting and engineering. However, some agile advocates tend to rebel against “the big design upfront”. These observations trigger a number of questions:

- How does architecting fit in with the popular agile methods, such as SCRUM, EVO, XP, etc.?
- How can we use an agile mindset when working on systems or technologies with intrinsic long time constants?
- What benefits did we experience by using agile from architecture perspective and what pitfalls did we hit?

2. Agile methods and philosophy

Agile approaches have been proposed and used for decades with varying names, such as incremental, iterative, and evolutionary. Gilb (1981) published a paper on evolutionary development as early as 1981. The publication of the Agile Manifesto around 2001, see Figure 1, provides the foundation for the term agile, agile as philosophy or mindset, and agile ways of working. This manifesto originates in the software world, as the full title, the Manifesto for Agile Software Development, shows. The manifesto reflects a philosophical response to dominant management styles in the nineties with its focus on processes. Many software engineers in that period perceived these processes as heavy weight, limiting, and ignoring individual talents and interests.

The manifesto writers elaborate the manifesto further in 12 principles at http://www.agilemanifesto.org/principles.html. If we substitute “system” for “software” in these principles, then they still make a lot of sense. However, the principle “The best architectures, requirements, and designs ***emerge*** from self-organizing teams” shows some tension between the manifesto and the architecting community. Architects have a responsibility for the qualities of a system in its context, which requires anticipation to avoid undesired emergence.

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 a plan

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

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

From: http://www.agilemanifesto.org/

Figure 1, the Manifesto for Agile Software Development

Currently, SCRUM is one of dominating agile *methods*. SCRUM originators have commercialized SCRUM by protecting many aspects with Trademarks. Typical terms in SCRUM are backlog, sprint, SCRUM master and many more. SCRUM is an all-encompassing *method* embedding sprints as increments, basing itself on the agile *mindset*.

The meetings discussed whether formulating a “systems”-agile could or should be an output of the forum. A system-oriented version of the manifesto would be a tangible and visible result. However, such result requires a significant effort too. Marbach et al (2015) have worked on such framework, building on an earlier paper by the same authors (Rosser 2014). Similarly, Dove and LaBarge (2014b) discuss agile systems-engineering. Carson (2013) argues that Systems Engineering cannot be agile; he limits agile to limited domains.

3. The relation between agile and hardware and software disciplines

A member presented a case study of an enterprise server development in the nineties. In this case study, the hardware developers used an agile approach. To achieve short cycles for iteration, simulation played a crucial role. The software development followed a conventional approach, with a heavy integration phase at the end of the project.

The perception of most forum members is that the software world dominantly uses agile, while hardware developments are less agile because of inherent lead times of hardware. The enterprise case study shows that hardware disciplines can work agile, certainly with today's simulation and fast prototyping possibilities.

4. Experiences with agile

Break-out questions:

- What successful examples of agile developments do you know in your company?
  - What are success factors?
- When does agile fail or is it inappropriate?

The participants shared multiple success examples, such as the software and electronics of the data path for pattern generators, a camera app, and a Thinking Machines super computer.

During the discussion the success factors in Figure 2 for applying agile popped-up. In Figure 2, the classification is loosely based on Obbink's BAPO model (Business, Architecture, Process, and Organization). This classification shows immediately that Managerial and Process have most success factors, followed by the human factors in the box people and team. In the discussion, the increased motivation of employees (engineers and designers) appeared as a significant benefit of working agile.

The relation with the customer pops-up as success factor too. Remarkable is that the success factor side does not use the term "stakeholders". Is this lack of stakeholder presence a consequence of the discussion (for instance a natural bias)? Or, does an agile way of working mostly deal with customers rather than other stakeholders? Finally, we see a relative small amount of technical-oriented success factors

Customer, Stakeholders

- Customer access
- Customer does not know its own problem
- Good visibility to the customer
- Customer/vendor "buy in"
- Allow/force customer to make changes in priorities per progress

Technical

- Off the shelf hardware
- Modularity (agile in the small)
- Affordability (program cost)
- Quality (defect reduction)
- Tools, simulation
- When it is unclear "how to solve it", for instance unknown physics

People, Team

- Unifying engagement, stretch target
- Mutual respect, trust team
- Empowered to decide
- Team maturity (prior experiences, trust, some process)
- (Software) engineers see agile as "fun"
  - More motivated
  - Less procedures
  - Fast feedback

Managerial

- Code productivity (lines software per hour)
- Deadline
- Executive champion
- Fast iteration cycles
- Value driven instead of plan driven (however, may result in function focus)
- Management recognizes flexibility in scope vs budget vs lead time
- PM support (run interference form executives)
- Fund training for team members
- Scale (it is hard if it is big)

Process

- Innovation/invention/decoupled
- Start from small core requirements set (additive)
- Willingness to change corporate process
- Fight process policy
- Agile can be done independent of the "mothership", so on small sub-projects
- Start testing as early as possible
- Agile is many V's
- Ideation <-> good match for agile
- Minimal viable product

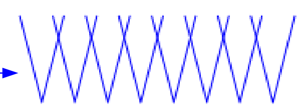

Figure 2. Success factors for agile

At the process side, participants use various mental models to think about agile, for example spiral like or a repetition of many small V-models. Literature proposes hybrid models too, for instance, Millard et al embed agile in larger hierarchical projects.

Figure 3 shows the failure factors that popped-up during the discussion. Questions that pop-up:

- How do minimal viable product and many Vs relate to main milestones?
- How to perform long term tasks as architecture?
- How to merge multiple clock speeds?

The main issues that we see at this point are:

- Agile tends to focus on functionality, which hurts the “ilities”. “Ilities” are core architecture concerns

- How to cope with multiple clock speeds?
- How to cope with strategic, long-term issues, and the "big-picture"
- When and where is agile beneficial (what success factors, what project or system characteristics)?
- How to scale agile to larger projects?

Customer, Stakeholders • Poor stakeholder collaboration • Divergent • Not really in contact with the customers (organization shields it off) • Customers knowing only 1 priority	*Technical* • Hardware: often long lead times in manufacturing and prototypes do not match fast iteration cycles • Agile approach yields a functional focus; non-functional/architectural focus abandoned	*People, Team* • Agile used as excuse for not being in control, or not wanting to communicate • Never finishing, gold plating; stop @ good enough
Managerial • Large complex projects, example airplane • Not managing deliverables across iterations	*Process* • Decision making, formal/escalation • When agile is embedded in a non-agile organization • Not recognizing the need for multiple clock speeds	

Figure 3. Failure factors for agile.

5. Agile architecting

At the end of the first breakout discussion, the participants agree that "agile" is desirable and may co-exist with architecting. At the same time, a main concern is that a definition in agile is still missing. These observations trigger the next breakout question:

How can we architect agile (=validate, verify early emerging "ilities")?

This question gets a broad variation in answers:

- (1) "Be in the weeds"
  - Ask questions
  - Try to detect where architecture is about to break
  - Adapt
- (2) Agile needs (upfront) architecture
  - Strong cohesion where agile takes place

- Loose coupling between agile teams
- Rules governing interfaces (static, dynamic, "-ilities", cross cutting)
- Watch rules by (1)

- Identify essential non-functional requirements as soon as possible
- Define review and test cycle for these essential non-functional requirements (different clock speeds)
- Identify those changes that cannot be changed easily
- Identify architectural significant requirements
- Create/select the baseline architecture
  - Decompose in modules, interfaces
  - Define the set architectural rules, e.g. all modules are using the same service interface.

One of the breakout teams captured the answer in a workflow for agile architecting, as shown in Figure 4.

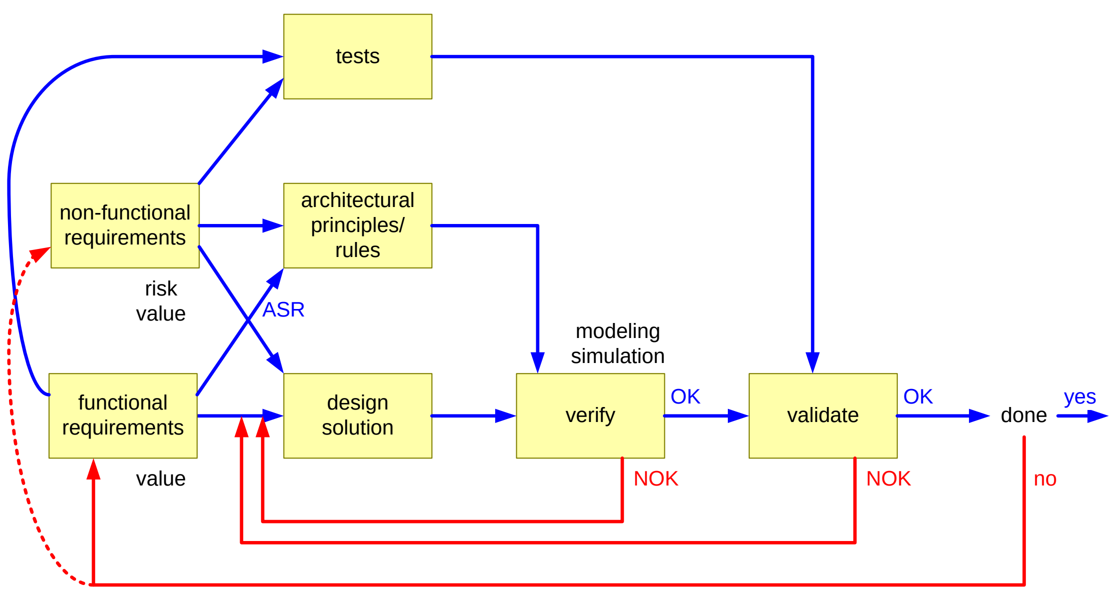

Figure 4. Agile Architecting Workflow

Further suggestions for agile architecting were:

- Iterative customer dialogues driven by
    - “day in the life” operations
    - High-level models
    - Five why techniques, etc.
- Architecture defines invariants
    - This stable ”core” facilitates agility for the agile zealots
- Scenario driven
    - –ility validation
    - Stress scenarios
    - Evolutionary scenarios
    - Exploratory/simple scenarios

Yet another approach is layering, as shown in Figure 5 to achieve agile systems [see also (Dove 2014a)]. The lower layers form a stable foundation. The higher layers cope with unknowns; the highest stable layer (Logical Data Models) copes with the known unknowns, the next layer uses acquisition to acquire in case of unknowns. The top layer needs agility to cope with the unknowns.

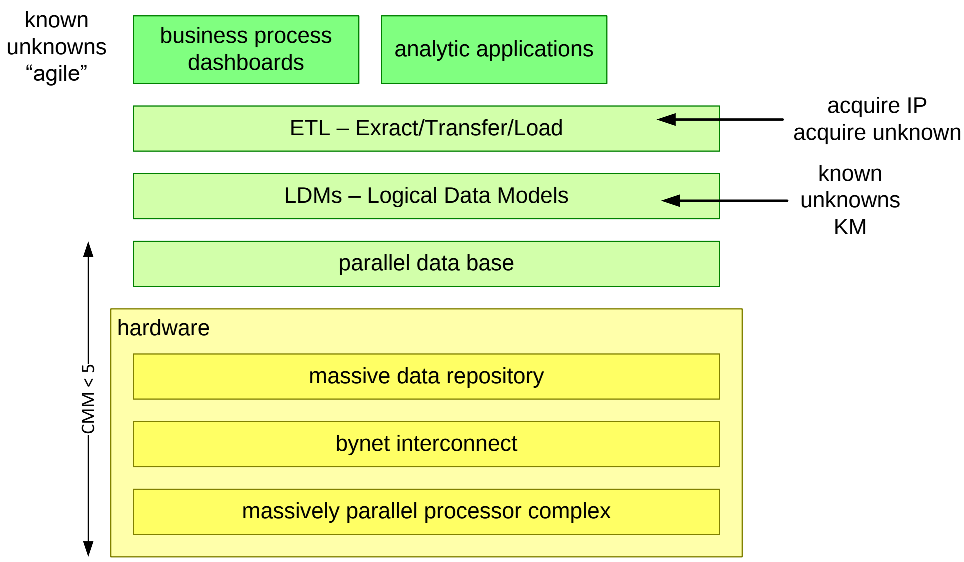

Figure 5. Layered architecture, where the lower layers are stable and higher layer need agility to cope with unknowns.

In the discussion at the end of this breakout, we concluded that:

- Agile is a mindset of validating and verifying early and using feedback
- The agile approach is context dependent
- Agile is applicable for architecting itself

> ***Principle 16.1 It makes sense to architect agile. Agile architecting strives for early validation and verification of the architecture, using fast feedback.***

6. Quality Attributes (or ilities, non-functional requirements, cross-cutting concerns, qualities, performance and usability requirements)

The previous section triggers a side step on terminology. One of the most crucial terms for architecture suffers from many labels. Wikipedia writes:

> Non-functional requirements are often called "quality attributes" of a system. Other terms for non-functional requirements are "qualities", "quality goals", "quality of service

requirements", "constraints" and "non-behavioral requirements". Informally these are sometimes called the "ilities", from attributes like stability and portability.

Scaled Agile (2016) also dominantly uses the term non-functional requirements. Gilb in a discussion at a conference remarked a few years ago that it is strange to label something with what it is not, rather than what it is. The SEBoK authors (2016) “solve” this problem by defining multiple types of requirements, such as performance and usability requirements.

Another way of looking at these quality attributes or ilities is their emergent and cross cutting nature, which makes them such important focus for architects. The resulting safety/performance/reliability/et cetera emerge from the dynamic behavior of the constituting (sub)systems and its context. Ideally, we like them to be composable, predictable, and reproducible. In practice, their static and dynamic relations are so complex that the quality attributes are not fully composable, predictable, and reproducible.

Principle 16.2 Agile architecting requires a strong focus on quality attributes of the system: early identification and definition, monitoring and measuring, early validation, guidance of allocation and interfaces.

7. How does agile scale to large projects?

We observed in the discussions that many examples of agile are computer, software, or IT-oriented, often in somewhat small projects. In large projects, we expect that multiple (multi-disciplinary) teams work concurrently with architect(s) interacting with all teams. In theory, such concurrency requires decoupling via interfaces. This triggers a new set of questions:

- How can we apply agile multi-disciplinary?
- How do architects interface with agile teams?
- Interfaces enable agility
    - How well do interfaces decouple?
    - What exactly is an interface?

One way that architects cope with qualities is by defining and monitoring boundaries for them, as shown in Figure 6. Figure 6 shows the evolution of the boundaries and the actual value of the quality attribute over time. SCRUM 0 will explore needs and design concepts to come to initial boundaries. The architecture avoids rigid constraints, rather it proposes concepts related to a rationale related to the desired quality attributes.

Once the design and build iterations progress, the quality attributes can be measured and monitored. When the implementation hits the boundary constraints too often, then the architects may adapt the architecture, and the quality attribute values. Such architecture pivot may result in refactoring sprint. Refactoring may also occur when the architecture gets polluted. An example of such pollution is when functionality is duplicated at several places in the code.

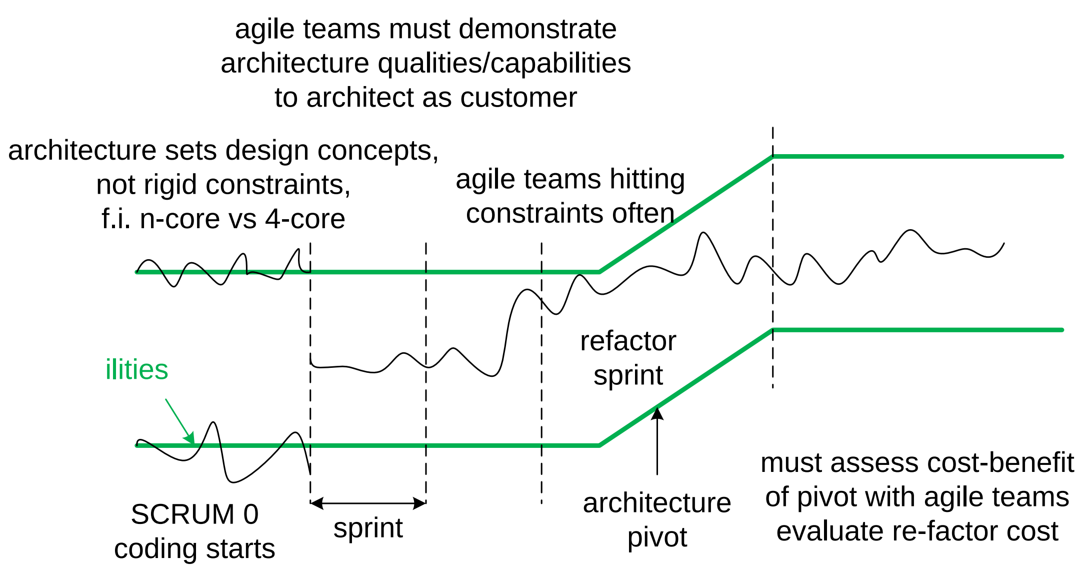

Figure 6. The development of quality attributes over time, illustrating how the architecture manages the boundaries.

The architect is a customer of agile development teams. The teams must demonstrate architecture qualities/capabilities to the architect. Architecting in such context is continuous, it does not happen solely at the beginning. The architecting mindset is to fail fast, and to find the architecture pivots.

Other recommendations from the breakout teams are:

1. Design architecture with independence – as much as possible
    - Then the architecture accommodates agile to point where realizations violate an interface
2. Plan Strategically for architecture change
    - Measure and analyze trends
3. Communicate Frequently with module teams
4. Build trust:
    - Teams + architect
    - Data driven resolution of architecture dispute
    - Responsive architecture decisions
5. Advocate for non-functional requirements (including costs), by objective
    - Integrate into sprints (definition of done)
    - Include in demonstrator (model/simulation)
    - “test harness” extra funds
6. Focus on key sources of change
7. Identify, manage stable interfaces
8. Identify, manage technical budgets
9. Stress/negative testing “gray box”
10. Architect needs, and create a strong bridge to knowledge, to ensure a strong link between system architect and implementation
11. Create a “strategy” for long-lead items (validity of learning)

Figure 7 visualizes the role of architects in typical SCRUM cycles with a backlog. The fundamental idea is that architects in dialogue with the product owner may insert architecture requirements in the backlog. The backlog will then contain functional items and architecture items.

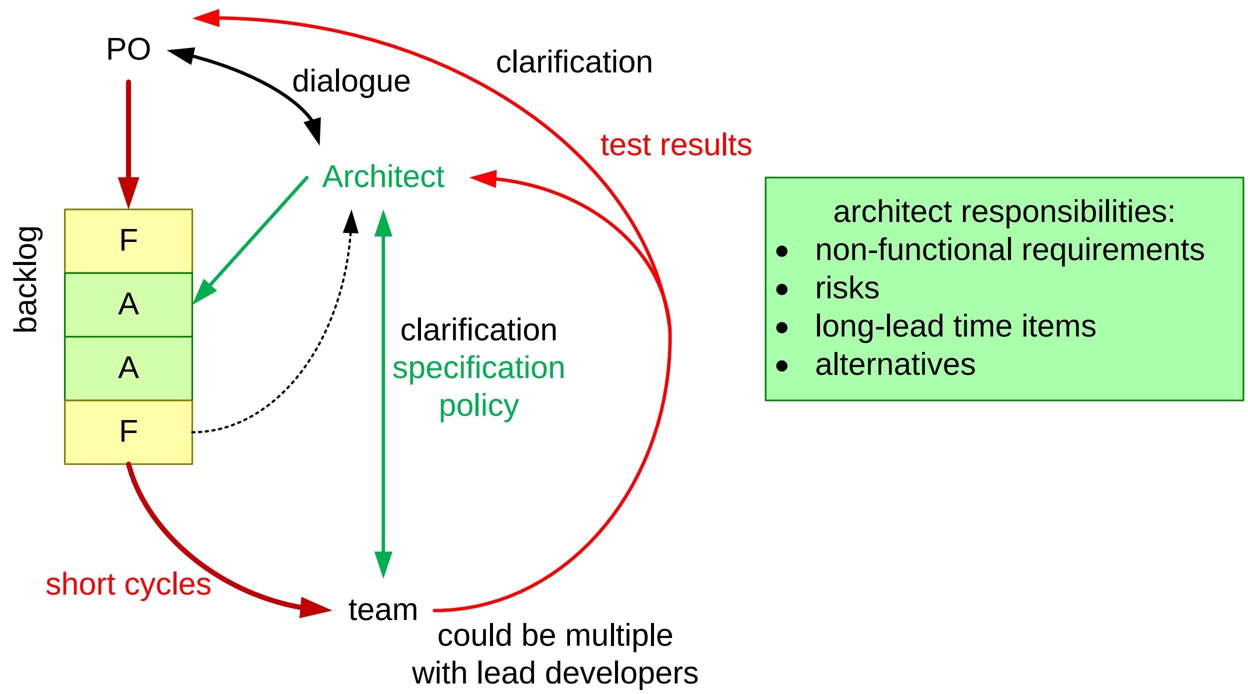

Figure 7.The role of an architect in the SCRUM cycle.

8. Conclusions

The discussions show that all participants see benefits in agile ideas. However, the main challenge is to achieve desired quality attributes when most teams iterate fast. We started with a number of questions, which we revisit after all discussions.

- How does architecting fit in with the popular agile methods, such as SCRUM, EVO, XP, etc.?

Architecting may fit well with these approaches. However, architects and managers need to ensure a continuous and harmonious relation between agile teams and architects.

- How can we use an agile mindset when working on systems or technologies with intrinsic long time constants?

The discussions did not provide clear answers on this question. Hybrid models, such as embedding agile teams in larger hierarchical organizations may work, when interfaces and constraints allow agile teams sufficient room to operate.

- What benefits did we experience by using agile from architecture perspective and what pitfalls did we hit?

Key benefit of agile approaches is early validation and verification, especially of emerging system qualities. This benefit does not emerge automagically from working agile, since it requires a strong relation between architects and agile teams. Although we didn't discuss pitfalls extensively, a dominating concern is that agile teams lose the big picture, resulting in local solutions that do not fit in the broader context and that limit future evolution.

Literature

Carson S.C., (2013). *Can Systems Engineering be Agile? Development Lifecycles for Systems, Hardware, and Software*. Proceedings of INCOSE 2013 in Philadelphia, USA.

Dove R., LaBarge, R. (2014a), *Fundamentals of Agile Systems Engineering – Part 1*. Proceedings of INCOSE 2014 in Las Vegas, USA.

Dove R., LaBarge, R. (2014b), *Fundamentals of Agile Systems Engineering – Part 2*. Proceedings of INCOSE 2014 in Las Vegas, USA.

Gilb, T. (1981). *Evolutionary development*. ACM SIGSOFT

Marbach, P., Rosser, L., Osvalds, G., and Lempia, D. (2015). *Principles for Agile Development*. Proceedings of INCOSE 2015 in Seattle, USA

Millard, D., Johnson D., Henderson J.M., Lombardo, N., Bass, R.B., and Smith J. (2014). *Embedding Agile Practices within a Plan-Driven Hierarchical Project Life Cycle*. Proceedings of INCOSE 2014 in Las Vegas, USA.

Rosser, L., Marbach, P., Osvalds, G., and Lempia, D. (2014). *Systems Engineering for Software Intensive Projects Using Agile Methods*. Proceedings of INCOSE 2014 in Las Vegas, USA.

Scaled Agile (2016). *Non-functional requirements NFRs*. Retrieved 11 July 2016 from http://www.scaledagileframework.com/nonfunctional-requirements/

SEBoK authors (2016). *System Requirements*. Retrieved 11 July 2016 from http://sebokwiki.org/w/index.php?title=System_Requirements&oldid=51435.

Wikipedia (2016). *Non-functional requirement*. Retrieved 11 July 2016 from https://en.wikipedia.org/wiki/Non-functional_requirement