Agile Systems Engineering

Bruce Powell Douglas and Kerim Cakmak

Rational Software
International Business Machines (IBM)
Systems and Software Engineering Solutions

ABSTRACT

Traditionally systems engineering is a heavily document-based approach. The arrival of the SysML modeling language changed the face of systems engineering from a process of creating and reviewing documents to the construction, analysis, and execution of high fidelity systems models. This evolution of systems engineering has resulted in improved requirements specification, better architectural definition, and better hand off to downstream engineering, including mechanical, electrical and software development. What is needed now is a process that takes advantage of the advances in model-based engineering (MBE) while at the same time reduces the time and effort required for systems engineering. Agile methods have proven successful in the software domain, but how can these incremental, iterative, and agile methods be applied to systems engineering? This is a talk about how agile methods have had a tremendous impact on the development of embedded and real-time software and how to best gain the advantages of agile methods within the systems engineering domain.

Keywords: Agile Methodologies, Model Based Engineering, System Modeling Language, Agile Practices, Agile Adaptation

INTRODUCTION

Traditionally systems engineering is a heavily document-based approach. The arrival of the SysML modeling language changed the face of systems engineering from a process of creating and reviewing documents to the construction, analysis, and execution of high fidelity systems models. This evolution of systems engineering has resulted in improved requirements specification, better architectural definition, and better hand off to downstream engineering, including mechanical, electrical and software development. What is needed now is a process that takes advantage of the advances in model-based engineering (MBE) while at the same time reduces the time and effort required for systems engineering. Agile methods have proven successful in the software domain, but how can these incremental, iterative, and agile methods be applied to systems engineering? This is a talk about how agile methods have had a tremendous impact on the development of embedded and real-time software and how to best gain the advantages of agile methods within the systems engineering domain.

CHALLENGES OF SYSTEMS ENGINEERING

Systems engineering focuses on capabilities and constraints at the systems level, above the more specific concerns of software, electronics, and mechanical implementation. The primary tasks for systems engineering include:

- System requirements specification
- System functional analysis
- System dependability analysis, including safety, reliability, and security
- Creation of a system architecture
- Allocation of requirements to subsystems
- Create hand off specifications for downstream engineering

Figure 1 shows the general system engineering workflow in terms of high level activities and milestones.

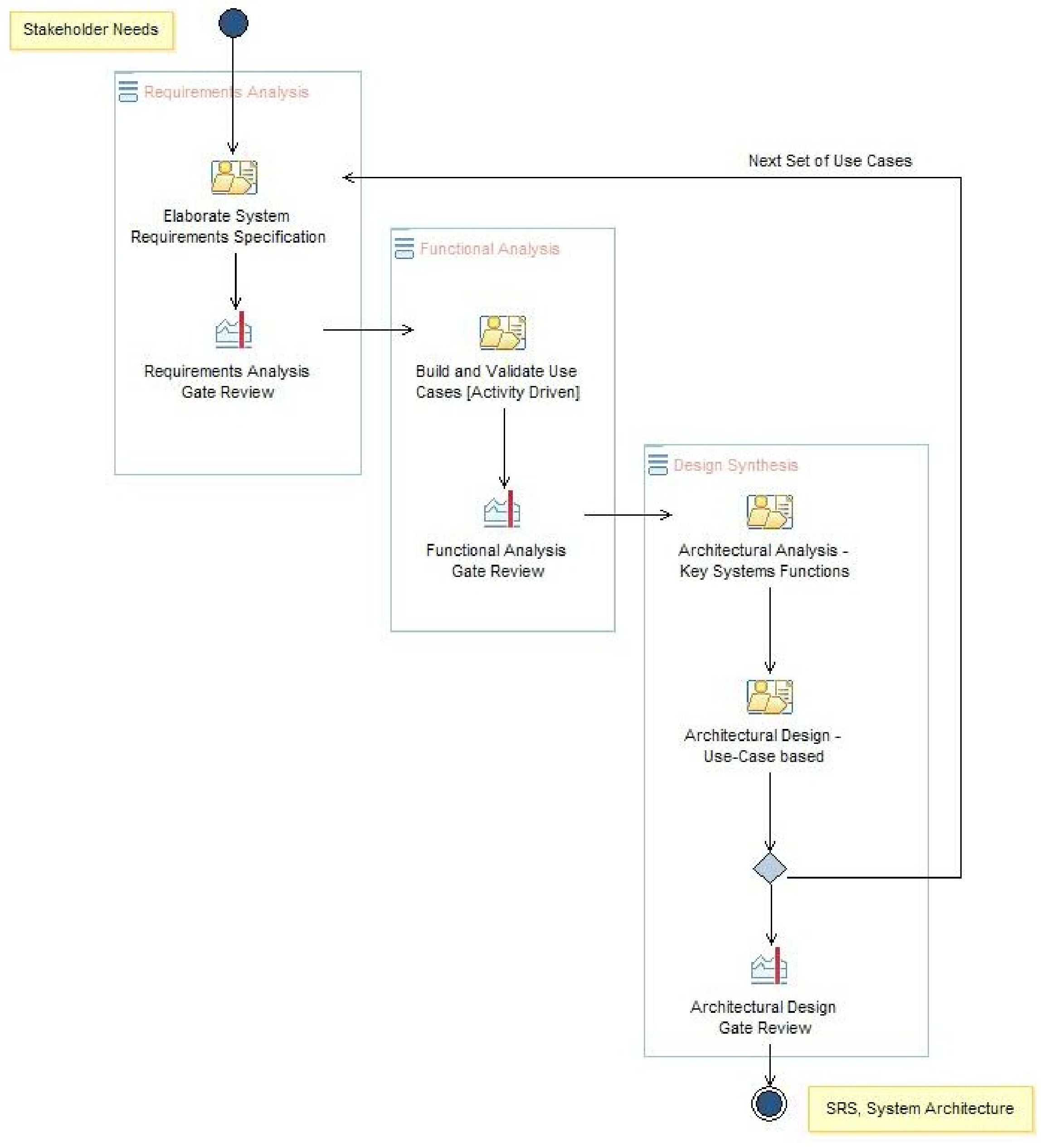
Stakeholder Needs
Requirements Analysis
Elaborate System Requirements Specification
Requirements Analysis Gate Review
Next Set of Use Cases
Functional Analysis
Build and Validate Use Cases [Activity Driven]
Functional Analysis Gate Review
Design Synthesis
Architectural Analysis - Key Systems Functions
Architectural Design - Use-Case based
Architectural Design Gate Review
SRS, System Architecture

Figure 1: System engineering workflow

System Requirements Specification

The System Requirements Specification (SRS) is a mostly textual document that defines the required behaviors and qualities of service of a system. Although the use of requirements management tools such as DOORS™ and Requisite Pro™ have improved the management of the requirements of complex systems, SRSs are often – if not usually – plagued by requirements that are incorrect, incomplete, inadequate, and inconsistent. This remains a key challenge for systems engineering that traditional approaches have improved upon, but failed to eradicate.

System Functional Analysis

System Functional Analysis (SFA) is the process of analyzing the functional needs of a system to both validate the requirements specification and to uncover requirements that are missing or problematic.

System Dependability Analysis, Including Safety, Reliability, and Security

In today’s smarter world, the various aspects of dependability – safety, reliability, and security – are becoming increasingly important. While these concerns fall within the umbrella of systems engineering, it is usually done by specialists. These analyses are used to ensure that requirements adequately address the dependability needs of the system, ensure that architectural and design choices are compatible with these needs, identify missing requirements, and to support the “safety case” when the product is submitted for approval by a regulatory agency.

Creation of a System Architecture

The system architectural creation focuses on the identification of large-scale pieces (“subsystems”) of the overall system, their responsibilities, and their interfaces. The system is usually not yet decomposed into engineering disciplines (e.g. software, electronic, mechanical, pneumatic, and hydraulic).

Allocation of Requirements to Subsystems

In preparation of the subsystem development, requirements must be allocated to subsystems prior to their being handed off to teams or subcontractors for development. These requirements must be consistent with the overall system requirements and often must result from the decomposition of system requirements into derived requirements.

Create Hand Off Specifications for Downstream Engineering

Traditionally, the output from system engineering is a large set of textual documents that, even though manually reviewed, contain various errors that will require significant rework late in the project. Part of the problem is that there is no way to automatically ensure that the textual statements are correct or even consistent with each other. The most common approach today is to hand off these hundreds to thousands of pages of textual specifications, errors and all, to the subsystem teams. The subsystem teams must then cast this information is a manner consumable by them and proceed. This is an open-loop approach that results in both low quality and high cost.

With the advent of SysML and high-fidelity model-based engineering (Hi-MBE) tools the modern systems engineering can be more effective in accomplishing their tasks.

System Requirements Specification

The System Requirements Specification (SRS) is traditionally a textual document often managed in a requirements management tool such as DOORS™ or Requisite Pro™. The SRS alone is problematic because while text is wonderfully expressive, it is imprecise and given to the creation of requirements that are incorrect, inconsistent, or inadequate. However, when used in conjunction with system functional analysis, these limitations can be addressed.

WHAT DO WE MEAN BY 'AGILE'?

Agile methods are a modern software development approach based on a dozen principles set out in the Agile Manifesto[1]:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.

[1] www.agilemanifesto.org/principles.html

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

From a practical point of view, agile methods are a set of processes – such as Extreme Programming (XP), Scrum, and the Harmony™ process that provide guidance for the application of the principles through a set of *practices* – or standard workflows. The practices emphasize development activities such as getting to execution immediately, continuous user-level testing, and continuous integration. These practices have been shown in real projects to realize the guiding principles of agile methods.

THE BENEFITS OF AGILE METHODS

The primary reason teams and companies move to agile methods is to improve the quality of their engineering work. In the IT software space, these benefits are well-documented. They are somewhat less well documented in the embedded software development space because the adoption rate has been slower, but it appears to be of similar scope. Little has been documented for the emerging application of agile methods to systems engineering, but initial work has been very promising.

The second most common reason people move to agile methods is engineering efficiency and productivity. Agile approaches have been shown to improve efficiency in a couple of ways. First, they improve understanding (and simultaneously reduce risk) of the system's problem areas. Secondly, they greatly reduce rework through an emphasis of defect avoidance over defect identification and repair.

The generally accepted benefits by Agilistas include:

- High quality systems engineering
- Efficient high quality systems engineering
- Earlier and greater project risk reduction
- Early return on investment
- Satisfied stakeholders
- Increased project control
- Responsiveness to changing requirements

High Quality Systems Engineering

In this context, we mean several things by the term "high quality." Since the ultimate output of systems engineering is engineering specifications, we mean that these specifications are complete, accurate, correct, consistent, and usable by their stakeholders. These goals are difficult to achieve with traditional methods whose only means by which to ensure quality is review. However, in the agile system engineering workflow, as we shall see in the next section, we avoid defects by building executable requirements models and executable system architectures. These allow system engineers to easily demonstrate the behavior of collected sets of requirements via execution rather than laborious and error-prone manual review of ambiguous textual statements. Model-based systems engineering uses a more precise language – SysML – to specify the requirements models, and creates traceable links to the textual requirements. It also allows us to demonstrate that the allocation of requirements into a system architecture is functionally correct.

Efficient High Quality Systems Engineering

Agile methods enhance engineering and productivity by minimizing both upfront work and later rework due to missed or incorrect specifications. Agile methods minimize upfront work by focusing on activities that have a high return on investment. They quickly reduce ambiguity, improve understanding, and lower risk. Because most defects are avoided (rather than detected later), reworking of existing specifications to remove defects is minimized.

Earlier and Greater Project Risk Reduction

In my experience, the leading cause of project failure is ignoring risk. In most projects, key risks are known early in the project. Unless steps are taken to address those concerns, the likelihood of project derailment due to risk manifestation is high. Risks are usually either due to attempting to achieve the impossible (or at least highly unlikely) or from things that are simply not known. That is, risk come from either ignoring known important information or from ignoring that you don't know things that you should. An example of the former is the consistent creation of unachievable project schedules despite historical data of poor project performance. An example of the latter is relying on a technology when it is clear that you don't understand all of the consequences of that decision.

Agile methods emphasize early active risk reduction though a combination of risk management and risk mitigation activities. Agile methods recommend a risk management plan that summarizes risks – including for each risk a description, likelihood, severity, and priority. For risks above a certain threshold of concern, risk reduction activities become scheduled tasks. These work tasks address the concern by either finding workable alternatives to impossible goals or by uncovering critical information about the technology to allow for better planning. Active risk management leads to early risk reduction, resulting in less rework by avoiding high-risk goals and technologies.

Early Return on Investment

One of the key problems with the standard V lifecycle is that the return on investment occurs very late in the project. This is because the V lifecycles assumes that each step in down the leading edge of the V can take place without making mistakes. We know this assumption to be false but we continue hoping as we run new projects. Hope, as it turns out, is a poor substitute for knowledge.

Agile methods focus on *time to value* – the time by which we have achieved measurable benefit to the stakeholders. Time of value is reduced only when the stakeholders have some true measure of project progress, and this can only come when you can evaluate (test) what you have.

Figure 2 shows the time to value for traditional text based systems engineering occurs at the end when the system is finally completed build and now, for the first time, can be tested. The use of incremental development moves this curve to the left but still have rework that needs to be performed, lengthening the time to project completion. When coupled with agile practices, the time to value curve is moved even more to the left, resulting in early true project progress.

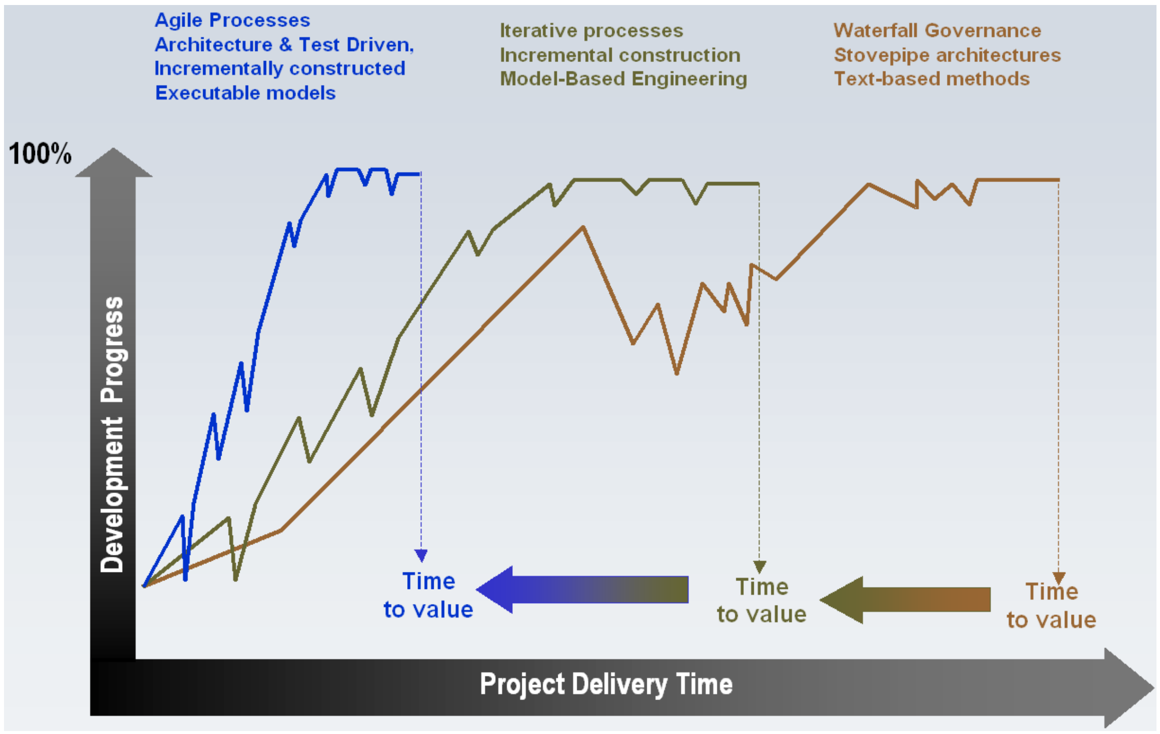

Figure 2: Time to Value

Satisfied Stakeholders

The emphasis in agile methods is achieving stakeholder needs over satisfying a contractual agreement. This is not to say that satisfying contracts is unimportant but rather that is even more important to meet the stakeholder's actual needs. By working with stakeholders, both goals can be achieved. In traditional approaches, this usually means extensive reviews of hundreds to thousands of pages of textual specifications. With executable models, the behavior of the system under a variety of different environmental circumstances can be easily demonstrated and explored even though the system is not yet built.

Increased Project Control

Agile methods use a variety of metrics to improve project governance. Traditional project governance is performed by measure progress against the project plan and is focused on achieving milestones. Agile governance stresses the importance of measuring progress against goals rather than arbitrarily decreed milestones. Agile methods use metrics such a project velocity and defect density rather than documents developed, resulting in far better visibility of true progress.

Responsiveness To Changing Requirements

Agile methods embrace change. In contrast, traditional approaches unrealistically assume that once a requirement is specified it will never change. In today's system development environment, change is a fact of life and we need procedures and practices that allow us to manage change, perform impact analysis and integrate changes into our workflow. Agile methods assume that change will happen and embrace it.

AGILE PRRACTICES FOR SYSTEMS ENGINEERING

It is important to remember that the purpose of software engineering is very different from that of systems engineering. The former is to produce deliverable implementation while the latter is to develop specifications. The application of agile methods to systems engineering must take this into account.

There are a number of key practices that can be effectively applied to systems engineering, including execution of specifications, test-driven development, model-based engineering, incremental development, and continuous integration.

Specification Execution

A key concept for agile systems engineering are the notions (and value) of executability and testability of specifications. In the pursuit of quality, we need direct evidence of completeness, accuracy, and consistency. This means we need to "test" the specification. Using a more formal language, such as SysML (and appropriate tooling), to create executable specifications means that we can simulate the system behavior, allowing us to demonstrate its functionality under different circumstances. The more traditional alternative of reading hundreds of pages of text is a poor substitute.

Test Driven Development

The practice of Test-Driven Development (TDD) is common in agile software development. It entails the development of test cases at, or slightly before, the development of some unit of software, and the immediate application of those tests. Unit level testing is performed incrementally, usually several times per day during 20-60 minute *nanocycles* to ensure that the evolving software baseline is defect free. The same approach can be applied to the development of executable specifications. As some set of related requirements are added (resulting in the addition of a few states, transitions, and system actions), test cases can be constructed to ensure that the added capabilities are properly specified. Compared with a test-at-the-end approach, TDD significantly improves quality and reduces rework.

Model-Based Engineering

Model-Based Engineering (MBE) doesn't remove textual requirements specifications but augments it with high-fidelity models that allow us to reason about the specifications that we are creating. The Systems Modeling Language (SysML) was created for exactly this purpose. MBE allows us to explore requirements in ways that ambiguous textual specifications do not and when we couple this with building executable specifications and TDD, we can construct superior specifications over using text alone. In addition, traceability between the modeled specifications and the textual requirements can be used to ensure coverage of all requirements and also assist in impact analysis when requirements change.

Incremental Development

Another key concept for agile systems engineering is the incremental development of a system functional analysis model. This incremental development occurs at two levels. First, we can incremental construct our specifications by working on one coherent set of requirements at a time. The most common way to perform MBE with SysML is to group requirements into use cases and work out the functional (and non-functional) requirements as well as their interactions. Each use case then forms an increment of the system specification. At the smaller scale of incremental development, each use case specification may itself be complex and subtle. At this nanocycle level of system engineering, the state-based behavior of the system use case can be developed using smaller increments. These nanocycle increments usually require 20-60 minutes and add some incremental piece of functionality that is then executed and tested for correctness. Many nanocycles are required to complete the specification for a use case.
Similarly, when architectural concepts are defined, the requirements can be allocated a use case at a time and the

architecture validated by execution before adding the next. In this way, the subsystem interfaces (and the corresponding Interface Control Document, or ICD) can be incrementally constructed as well.

Continuous Integration

If different teams are working on different use cases simultaneously, it is important that they requirements specified for one use case don't conflict with the requirements for another. This problem can be eliminated though a practice known as *continuous integration* in which the work from different engineering teams is brought together and tested to ensure consistency. This approach identifies conflicting requirements early at a much lower cost than in more traditional development lifecycles.

ADOPTING AGILITY IN SYSTEMS ENGINEERING

The best way to adopt new approaches and technologies are to do so incrementally and realizing return on investment quickly and efficiently. The best way is through a series of five steps:

1. Assessment

An outside assessment identifies the key strengths and weaknesses of your engineering organization in a variety of aspects, such as project planning, execution, engineering, quality, governance, and collaboration. Trained assessors can quickly zero in on the areas in your organization that offer the highest return.

2. Plan the adoption

Based on the assessment, this step plans which practices and tooling will be adopted, a pilot project for that adoption, metrics for measuring the success of the technologies and the adoption, and how training and mentoring figure in to that pilot. A key aspect is to define an appropriate set of metrics, known as Key Performance Indices (KPIs) that can be used during the execution of the pilot project and as well as at its conclusion. These KPIs will be used to ensure the pilot (and the adoption) is running well and to perform mid-course corrections to the approach as necessary.

3. Run a pilot project

Ideally a small pilot project that has both a high return on investment and low risk is used to begin the adoption. Usually external experts provide some initial training in the practices and the tooling, and then pilot is begun. As the pilot is run, the KPIs are continuously monitored and evaluated. At periodic points, mid-course corrections are made. These corrections may be due to adjustment for skill levels, business cultures, tool features, etc. The goal is to complete the pilot project as successfully as possible and learn lessons that can aid in the general adoption of the technology at the organizational level.

4. Reassess

Following the pilot, a review is held to access the success of pilot and capture lessons learned. Based on this information, a general deployment within the organization can be planned. This is normally a phased plan with at least short-, mid- and long-term actions to update the technologies and practices within the organization to take advantage of the technologies. The plan must also identify KPIs for the general deployment to maximum success. These may be the same KPIs as for the pilot or the may be different.

5. Deploy in your organization

The deployment phase of adoption means that we act out the plan outlined. In addition, reviews and retrospectives are held during the phase and plans are adjusted based on how well the deployment is working.

CONCLUSIONS

Agile methods have their roots in software development. They are a set of practices and technologies meant first to improve the quality of software work products and secondly to improve the productivity of the engineering teams. Agile methods are based on a set of principles that emphasize work that correlates to quality and de-emphasizes activities that do not. The most important of these practices are incremental development, test-driven development, continuous integration, and continuous execution.

Although agile methods come from the software world, these practices and technologies apply equally well to systems engineering. Through the use of high-fidelity modeling approaches using the SysML language and organizing the workflow to use the agile practices, the quality of the system engineering work products can be significantly improved while lowering the engineering cost.

The recommended approach is to adopt agile methods in an agile way. That is, assess where your organization is and where it wants to be, adopt the technology in an incremental fashion using a pilot project with mentoring from experts, monitor success using KPIs, reassess, and then deploy across your organization.

The System Accelerator provides a combined tooling and agile practice environment for the various workers within the system engineering environment. The OSLC-integrated tools use RESTful interfaces to support strong integration among the key tools in the systems engineering environment including requirements management, systems engineering modeling, quality management and project governance.

REFERENCES

Britcher, Robert *The Limits of Software: People, Projects, and Perspectives* (Addison-Wesley, 1999)

Douglass, Bruce Powel *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems* (Addison-Wesley, 2002)

Douglass, Bruce Powel *Real-Time Agility: The Harmony/ESW Method for Real-Time and Embedded Systems Development* (Addison-Wesley, 2009)

Douglass, Bruce Powel and Mats, Gothe *IBM Rational Accelerator for Systems and Software Engineering.* http://www.redbooks.ibm.com/abstracts/redp4681.html?Open

Ericson, Clifton II *Hazard Analysis Techniques for System Safety* (Wiley Interscience, 2005)

Hammer, Robert *Patterns for Fault Tolerant Software* (John Wiley and Sons, 2007)

IBM *Smarter Products* *http://www.ibm.com/smarterplanet/us/en/embedded_systems/ideas/index.html?re=sph*

Schumacher, Markus; Fernandez-Buglioni, Eduardo; Hybertson, Duane; Bushmann, Frank; Sommerlad, Peter *Security Patterns: Integrating Security and Systems Engineering* (Wiley and Sons, 2006)