Steel Threads: Framework for Developing Software System Architecture

Sada Narayanappa
Engineering Process and Operations
Jeppesen, Inc.
sada.narayanappa@jeppesen.com

Wan D. Bae
Mathematics, Statistics and Computer Science
University of Wisconsin-Stout
baew@uwstout.edu

Shayma Alkobaisi
Faculty of Information Technology
United Arab Emirates University
shayma.alkobaisi@uaeu.ac.ae

Narayan Debnath
Computer Science Department
Winona State University
ndebnath@winona.edu

ABSTRACT

A steel thread is a set of logically grouped scenarios that identifies the execution path through the system to meet business objectives and to demonstrate executable architecture. Steel threads are often used in the context of defining a software system architecture. Although there have been references to steel threads in software engineering literature, it is hard to find literature of steel threads in books or among the research community. This paper presents the definition of steel threads and contexts under which they are applicable in software development life cycle. The applicability of steel threads is also presented through a case study of a software system.

1. INTRODUCTION

Software architecture has emerged as a crucial part of a software system design process. The development of a new software system is an overwhelming task. In a typical scenario of software system architecture, high-level requirements are handed over to a development team to define and create the solution. There are many risks involved with giving too little thought to the overall structure of the solution and how the product can be extended and maintained in the future. Any solution of a reasonable size or complexity requires some initial analysis and design on how the solution should be implemented. Without guidelines, a solution might require extensive rework in subsequent processes which creates unmanageable implementations. Also, a technically complex solution might face difficulties to describe the essential elements of the system resulting in maintenance inefficiency as the system evolves. Figure 1 shows a cycle of developing a software system that is influenced by business requirements, technical environment, architects' experience and architectures. A business manages this cycle to handle growth, to expands its enterprise area, and to take advantage of previous investments in architecture and system building [2].

Over the last few years, as many tools and technologies have been developed, an approach called *Steel Thread* (*ST*) has been used in developing software systems. A steel thread can be considered as a software engineering tool that defines a fundamental path of execution in a software system. Steel threads focus on the basic behavior or primary flows (core usages) of the system and typically ignore the alternate and exceptional scenarios [3]. The main advantage of this approach is that it is easier to describe a complex system with a set of fundamental paths. Steel threads could describe a system without too much complication and help people to understand the basic system. In addition, this approach offers an effective way to trace the development through complex implementation both for comprehension and evaluation.

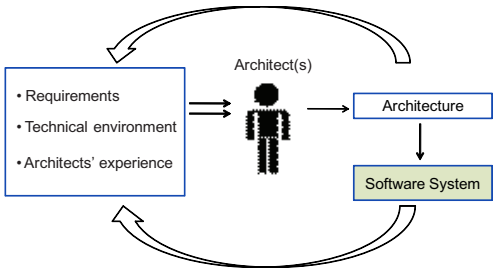

Figure 1: The architecture business cycle

While steel threads have been used in software development, little research has been done on formalization of steel threads and applicability in a real system. This paper discusses the definition of steel threads and presents their contexts and use in software system development. The applicability of steel threads are also discussed in detail. A case study of steel threads for an air traffic control system shows how steel threads are constructed in a software system.

Steel threads give architects the ability to define and communicate a solution while creating artifacts that become part of the solution. Similar to use case specifications, steel threads are identified in the early stage of software development to address critical software system concerns. The term is derived from the idea that primary functionality of a software system is a "thread" that runs throughout the system, and that the importance of this thread's role in the system makes it strong like "steel" [1]. Little regard is given to exception logic or other functionalities that are not fundamental to the nature of the system. When coded, a steel thread is a thread of implementation that weaves through the system to realize its functionality. Figure 2 illustrates the main idea of steel threads in software system development.

The rest of the paper is organized as follows: In Section 2, we provide background and related work. Section 3 presents overview of steel threads. Section 4 discusses contexts of steel threads in software systems. We discuss identification of scenarios in the development process of steel threads. We

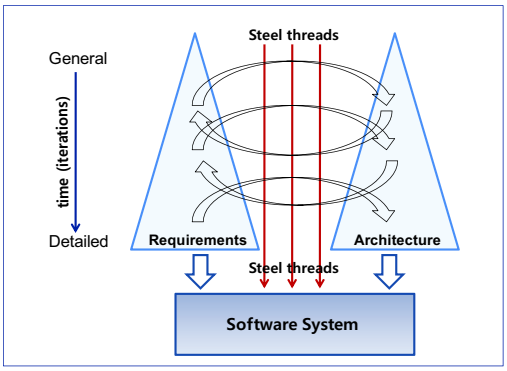

Figure 2: Steel Threads with business requirements and software architecture

also discuss roles of steel threads as well as their usability and applicability. Section 5 presents a case study of a system. Finally, Section 6 concludes the paper.

2. RELATED WORK

The architectural view of a system is abstract, separating from details of implementation, algorithm, and data and concentrating on the behavior and interaction of system elements [2]. A software architecture is developed as the first step toward designing a system. The relationships among business goals, requirements, architects' experience, system environments, and architectures form a software system life cycle with feedback loops. Software processes in the architecture business cycle are the processes of software development activities. These activities include the following [2]: 1) creating the business, 2) understanding the requirements, 3) creating or selecting the architecture, 4) documenting and communicating the architecture, 5) analyzing or evaluating the architecture, 6) implementing the system based on the architecture, 7) Ensuring that the implementation conforms to the architecture.

An approach that minimizes system risks to particular implementation and focuses explicitly on the functionality of the system is required. Accordingly, the high-level architecture has been developed in line with the principles of the model driven architecture [4], promoting clarification of "what" the system is required to do, rather than "how" it is to do it. The essential logic of the system can be captured in a Unified Modelling Language (UML) [5] model that is independent of implementation technology and platform.

The term *steel threads* is often used in software system development. In [1], a URL link in Wikipedia, describes steel threads as "construct that indicates the most important path(s) of execution in a computer system". However, this lacks many aspects such as, who, what, how, when, etc. Literature search to understand steel threads turns up slim to non results.

3. OVERVIEW OF STEEL THREADS

3.1 What are Steel Threads?

Steel threads are defined as a set of logically grouped objectives that comprise an end-to-end or top-to-bottom thread of execution through a system, that satisfy chosen business objectives, and do not consider alternate or exceptional scenarios. It is often considered as a collection of use case scenarios that represent the functionality of the proposed system. A steel thread includes that portion of the system that satisfies these scenarios. These set of scenarios are chosen to address the project concerns. The primary goals of steel threads are to:

- Mitigate and provide mechanism for addressing technical risks associated with the architecture
- Demonstrate scenarios that are critical to both the customer and the overall system
- Establish foundations on which the proposed system is built

Suppose that the system being developed is integrating with external systems such as billing, payment, and shipping systems: *An online bookstore system that interacts with three external systems such as billing system, credit card service, and shipping service systems may identify many use case specifications.*

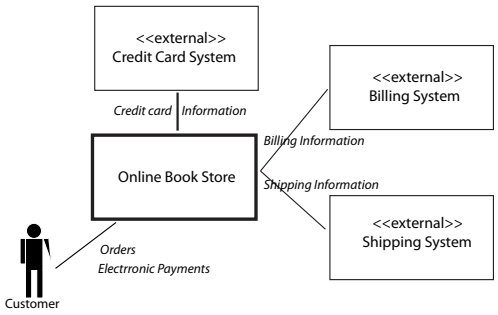

Figure 3: Architecture example: online bookstore

This is a typical system as encountered in many software environments. Figure 3 illustrates the system contexts of the architecture for this online bookstore system. We omit the full description of the system and rather focus on the relationships with the external systems. Customers order books online and during the checkout process, a billing system provides a payment after computing discount, taxes, etc. Customer provides the credit card payment information which is processed by the credit card service system. If the credit card transaction is complete, a message is sent to shipping department. Finally, the books will be shipped to the customer.

There are possible dependencies among these systems that must be carefully considered. Proving the initial system development with external systems will help to establish a solid foundation and communication to external system architects and project managers - this facilitates early integration and resolution of any quality attributes of the external systems such as availability, reliability, security, etc. A listing of steel threads for the above system may consist of the following scenarios:

ST1 : A customer orders books through online

ST2 : A billing system computes the final bill

ST3 : A credit card service system completes a transaction

ST4 : Completed payment triggers a shipping service system to ship the items

It considers the risk of external systems correctly acting to triggered events; it ignores the scenarios of checking the inventory before collecting the payment or verifying the destination address of the customer or how the customer is created and authorized to order books.

3.2 How are Steel Threads different?

This section addresses the common question that often arises, "how are steel threads different from requirements or use cases?". A model of software development [8] is a partial view of the well-known spiral development model. It focuses explicitly on requirements and architecture, allowing for their concurrent and independent evolution. Software system development proceeds with successive iterations of both these concerns, leading to increasingly detailed requirements and refinement of the system architecture.

In short, steel threads need not be always different from requirements; steel threads are subset of requirements. They represent the minimal set of requirements, which when coded raise the confidence level of delivering a system. They aid in clarifying or addressing technical challenges which may be because the team is not familiar with the technology or the problem domain is new and the team does not have experience in delivering such a system. Steel threads form significant requirements which drive the architecture. For example, a requirement that says, that customers will be able to access the system remotely from anywhere in the world via a browser, this necessitates the need for an application on the internet (given the prevalence of such approach for current technology).

Steel threads often ignore the trivial cases to keep the focus on developing a central functionality that once built, other functionalities can be wrapped around it mechanically or also by junior developers. Steel threads need not be limited functional characteristics; they can include non-functional aspects as well if they constitute a risk that must be addressed by the architecture. It should also be noted that steel threads need not be always be created from scratch. If a use case satisfies the goals of steel threads, this can be considered as one of the steel threads. Steel threads also eliminate the need for constant prototyping the system to demonstrate value to customers in which teams never deliver a real system [9].

4. CONTEXTS OF STEEL THREADS

4.1 Identification of Scenarios

A steel thread is a technical proof of concept that considers all possible technologies in a solution [6]. Steel threads are stated in the context of a use case and represented by a sequence diagrams in UML. The scenarios for a steel thread are picked from one or more use cases rather than inventing new ones. Steel threads should not be confused with programs executing in a thread of control (as in multi-threaded applications). Identification of the business objectives that comprise a steel thread considers two factors: the technical risk associated with these scenarios and the areas of importance to the customer.

During discovery of requirements, many factors constitute to the risk. It is critical to move between problem and solution space during the architecture and requirements discovery. This also necessitates the need for architect as a critical lead is discovering requirements. This approach has several draw backs in discovering some requirements: 1) requirements that are technically infeasible - results in repeated work to collect new set of requirements; Repeated misunderstanding will result in customer dissatisfaction or confusion, 2) requirements that are outdated by newer approaches due to advanced technology, 3) requirements if slightly altered will not affect the system objectives but will result in huge saving in development cost due to reuse of internal assets. Other extreme is to stay in focus on solution domain while little attention to problem domain. This again causes projects to fail as the solution always takes the form of the teams past experience and approaches; thus offers fewer innovations and address to user concerns.

There could be many factors that play to determine what constitutes a steel thread such as teams experience with the problem domain, technology expertise, maturity and understanding of external systems, etc. The criteria used to identify steel threads are mainly to identify and mitigate technical risks. In addition, one may consider flows that are critical to customers. Once steel threads are implemented they form the basis upon which the other functionality is wrapped around to continue development. As a summary, pitfalls of identifying steel threads are as follows:

- Identifying scenarios that do not relate to any existing use cases
- Identifying scenarios when implemented do not yield reusable artifact
- Identifying scenarios that do not offer any useful structure for testing
- Identifying scenarios that waste development time
- Identifying scenarios that do not offer higher level of abstraction to understand the system
- Overlooking existing use cases that can serve as a steel thread

4.2 Roles in System Development

Steel threads serve many purposes at various stages of system development and also during system evolution.

First, identifying the steel threads of a software system reduces technical and project risks. Steel threads allow the development team to figure out any technical issues early in the project life cycle. Having steel threads of the software solution handed off to the development team eliminates much ambiguity as to the expectations of the development team. Steel threads facilitate developers to build and verify key system functional and nonfunctional requirements early in the project life cycle and to mitigate or resolve risks. This permits modification of the architecture design, if necessary, prior to committing to a proposed solution and implementation of the system. The approach of steel threads builds what is a main success scenario for a significant objective-based requirement.

When the development of steel threads is complete, remaining functionality and alternate scenarios are wrapped around the basic structure with higher confidence and understanding of the expected solution. Hence steel threads

provide a framework for software system architecture. The development environment including software configuration management and continuous integration capabilities is established for the software development team. Steel threads can also be used to evaluate the architecture and design to assess its effectiveness.

Further more, steel threads can be used as a tool to communicate with various stakeholders. An effective architect plays multiple roles on any given project. Beyond defining a solution to a software system for a business problem, an architect's responsibilities often include communicating the solution to multiple stakeholders: business sponsors, potential end users, and development organizations. Usually steel threads are represented similar to how requirements are represented in the project. The advantage of representing them as such is to have consistent way of understanding the requirements by the project. This makes the communication between architects and stakeholders consistent and efficient. In this paper, we assume that the requirements are captured using use case specifications [2, 7], which are more widely used.

4.3 Usability and Applicability

Defining and developing steel threads offers many advantages to a software system development. The following advantages summarizes and illustrates the usability of steel threads:

- Defining the basic structure of implemented solution and capturing the essential elements and functionality of the system
- Early identification and mitigation of technical risks
- Enabling demonstration of systems end-to-end critical functionality to stakeholders and facilitating means of communicating of architecture to detailed design and implementation
- Providing a basis for validating estimates for the tasks associated with implementing the steel threads, and applying the new information to the other estimates
- Serving as a concrete example for software test discipline to plan test verification and validation activities
- Providing a proof of architecture concept
- Providing a reference architecture with deliverables
- Providing a reusable and tangible architecture
- Providing a basis for many other systems to be built from basic functionalities
- Serving as a mechanism to get a higher level of confidence of the proposed architecture

There might be some steel threads to identify and mitigate critical and technical risks in the proposed solution. For example, the scenario when a large number of users are concurrently ordering books. It is possible that some Use Case Specifications, by omitting alternate scenarios, may serve as a steel thread. In these cases, it is best to use the existing specification as a steel thread.

Steel threads are used throughout the software system development. Their applicability can be found in the following processes: 1) requirements, 2) architecture, evaluation, and proof of concept, 3) detailed design, 4) implementation, 5) test, 6) system evolution.

5. A CASE STUDY: AIR TRAFFIC CONTROL

The case study of steel threads presented in this section demonstrates selective business functionality by implementing an executable architecture. Steel threads are to demonstrate critical business objectives, which are mainly related "end-to-end" information flows. This case study identifies steel threads that are influenced by the context of customers, development organization skills, risks, architect skills, domain knowledge (in this case author's domain knowledge), etc. A variation in these influencing factors may result in identification of different set of steel threads. Therefore, the task of identifying steel threads varies depending on the system - its applicability must be evaluated for the given context.

5.1 ATC System Description

The example application for this case study is taken from [2] which describes Initial Sector Suite System (ISSS) used for the Air Traffic Control (ATC) system. The system design goal for ATC is to provide effective and efficient way to monitor and control air traffics for all types of aircrafts. High availability and high performance are the main objectives in the system.

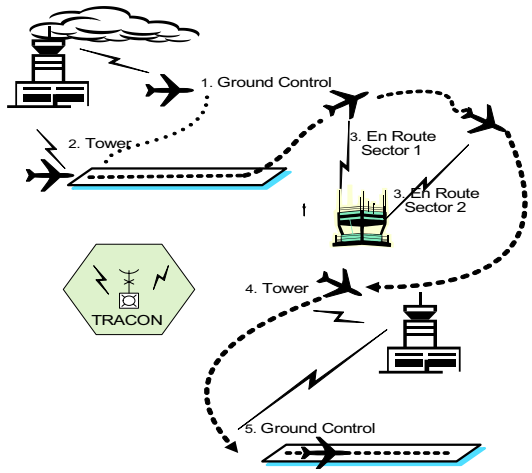

Figure 4: Overview of ATC

Figure 4 illustrates the overview of ATC. As an aircraft plans its route from an airport to another, it is monitored and controlled (or guided) by various entities. *Ground Control* monitors the movement of the aircraft while it is on the ground, *Tower Control* monitors the aircraft during its taking off until it leaves the *Terminal Control Area*, a cylindrical section of airport centered at the airport, and *En route Centers* monitors the aircraft during its en route. The aircraft is handed over from one control to another as it leaves and enters areas of the respective controllers (ground, tower, and en route). In addition, the aircraft could be handed over from a sector to another sector (a variable section of the enroute section) where each sector may have one to four control stations with at least two controllers; the first one

monitors and manages safe operation of aircrafts, and the second provides necessary data to the first. ISSS initially plans to for en route centers later to be extended to ground and tower as all these systems share many components.

5.2 ATC Risks and Business Objectives

Some of the architectural (technical) risks in ATC can be derived from the description. These risks are influencing factors in identifying steel threads although not all risks can be mitigated by steel threads.

- System may not meet the likings of Controllers (end-users of ATC) and who also have the ability to reject the system.
- Inability to upgrade or update the system due to its high availability requirement (of only allowable down-time of less than 5 minutes per year)
- Inability to meet the concurrency (and hence performance) requirements (ability to handle up to 2400 aircrafts both computationally and network band width)
- Inconsistent components - given the size of the overall system, it is challenging to assure consistent quality of components (or subsystems) and adoption of common development process across multiple vendors
- Budget cuts resulting in only a subset of functionality be required by customer
- Given the nature of its application, safety and security issues storm the system requirements, development, and compliance.
- Backward incompatibility of the interfaces and designs to existing systems

Notice, although budget cut may not appear as a technical risk, it can be partly addressed by reducing the maximum depth of the dependencies among the software elements, in other words, decomposing the system into smaller self contained functional subsystems. A steel thread is identified such that it lies within these functional subsystems so as to influence and isolate development and testing within these subsystems.

A couple of high level business objectives need to be considered for steel threads for ATC. Business objectives that take into account influencing factors such as risks and skills are identified in order to further refine and constrain the scope of steel threads. Notice that these objectives may vary from overall system principles because the objective of steel threads is to address risks and to demonstrate executable architecture by providing a framework for other executable threads to be woven around it to build the completed system. The following enumerates a number of very high level objectives that steel threads to fulfill:

BO1 : Demonstrate basic functionality

BO2 : Create a core fundamental framework around which the identified subset of the system relies upon

BO3 : Demonstrate the peak performance that can be achieved (concurrent flight data and radar handling)

BO4 : Demonstrate the fail over mechanism from primary to secondary

BO5 : Demonstrate the security and safety aspects of the system

In this case study, we have the following contexts and assumptions about the system: 1) The system is developed by a number of external vendors, who work under one primary vendor and who will assemble the overall system, 2) The developing organizations may not have skills and experience in developing similar systems in the past - furthermore, these organizations will not have training facility to build those skills. Therefore, quality must be tested before acceptance, 3) If during the demonstration of steel threads, the system proves to be infeasible or unfit for the purpose, then the customer will choose to halt the project as long as it needs to rectify the situation by identifying new suitable requirements, 4) The hardware and software for the system development and deployment are identical to the host system and available for developing organizations.

5.3 Steel Threads for ATC

Based on the risks, business objectives and assumptions, we identify four high level steel threads for the ATC system. These steel threads may appear trivial and ignore many alternate scenarios that is normally encountered in the requirements.

ST1 : Steel thread that demonstrates acquisition of radar surveillance data and data visualization on controller screen.

ST2 : Steel thread that demonstrates handoff from one sector to another sector. This uses the already acquired data (which may be cached) within the same controller center.

ST3 : Steel thread that demonstrates the replay functionality of the system by replaying the consoles and communication. In addition, this demonstrates the sufficient logging, configuration of logging, exception and error logs.

ST4 : Steel thread that demonstrates the archival and cache clean up when the aircraft leaves the area controlled by the enroute center.

Due to space limitation, we present an overview of *ST1: Acquisition of Surveillance Data* but omit its detailed analysis.

ST1: Acquisition of Surveillance Data

This steel thread defines a scenario that the system acquires radar surveillance data and displays the data in plan view on controller screen. In addition the steel thread includes that data controller retrieves relevant flight plan data strips. Business objectives met are BO1 and BO2. This steel thread does not need to take into consideration fail over to secondary host. The surveillance data is acquired for all flights within the specified En Route center. However, flight plan data is available for limited flights. The system is initialized and ready for operations, which means host computer has the access to hardware to acquire surveillance data and flight plan. The contexts of this steel thread are as follows:

1. The system is in operation and continues to monitor the data from surveillance equipments
2. Upon data detection from surveillance equipment, the controller is able to see the surveillance data in plan view on the controller screen (see Figure 5)
3. The flight plan information is also available for display; Data Controller can access the flight plan for those aircrafts within the controller's scope
4. When the aircraft leaves one sector to another, the related data migrate to the corresponding display
5. When the aircraft leaves the area of controller, it is disappeared from the controller display
6. Using the display, controller can monitor the safe separation of aircrafts

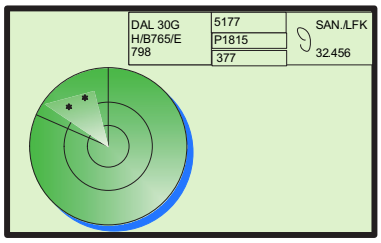

Figure 5: Sample view of controller screen

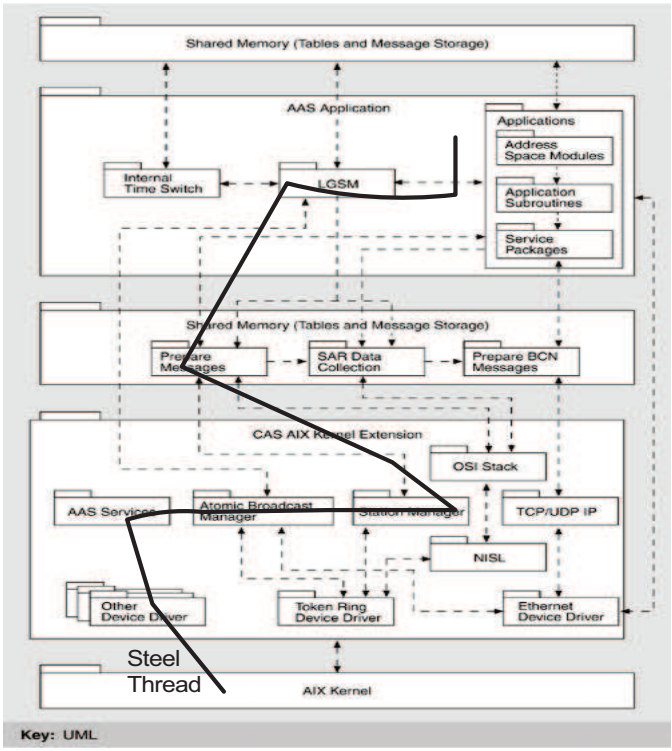

Figure 6: Steel Thread - weaves though various layers in architecture

Implementing this steel thread results in partial realization of deployment architecture with single host, networking elements, and integration with other external systems. It also establishes and validates Code View architecture. In the layered view, the implementation passes though every layer. For example, device drivers, network drivers in kernel extensions layer; prepare messages package in shared memory (layer above kernel extensions layer). In addition, this steel thread provides the basic functionality through application layer. Figure 6 is an example of a system where the steel thread touches various modules in layered architecture. Other steel threads are similar in that they are woven around the architecture; eventually, the over all system is woven around these basic steel threads.

6. CONCLUSIONS

Steel threads have been used in many contexts throughout software development life cycle in industry. However, a clear definition, well defined application and areas of use in software development remained undefined and unclearly explained in the literature. In this paper, we presented definitions of steel threads as an effective tools for building software systems and discussed the context of steel threads. We illustrated applicability of steel threads through a case study where steel threads are software engineering constructs that can define fundamental paths of execution in a complex software system though a set of logically grouped scenarios. We described the system, defined the architecture risks and objectives and finally provided steel threads context of the system. We plan to investigate on defining steel threads for common business requirements. Consequently, new tools for reuse of existing steel threads can be introduced.

7. REFERENCES

[1] Steel thread: Wikipedia url. "http://en.wikipedia.org/wiki/Steel_thread", 2009.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, SEI series in software engineering*. Addison Wesley, 2003.

[3] M. Fergal. Managing software projects with business-based requirements. *IT Professional*, 4(5):18–23, 2002.

[4] O. M. Group. The model driven architecture. "http://www.omg.org/mda/".

[5] O. M. Group. Unified modelling language specification v2.1. "http://www.omg.org/spec/UML/2.1.2/", 2007.

[6] J. Hofstader. Building distributed application: Model-driven development - msdn url. "http://msdn.microsoft.com/en-us/library/aa964145.aspx", 2009.

[7] N. C. N.C. Nc and C. Gryce. Descending the twin peaks: Requirements and architecture in the egso project. "http://www.nesc/ac.uk/events/ahm2003/AHMCD/pdf/086.pdf", 2003.

[8] B. Nuseibeh. Weaving the software development process between requirements and architectures, 2001.

[9] N. Rozanski and E. Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley Professional, 2005.