Novel Lightweight Engineering Artifacts for Modeling Non-functional Requirements in Agile Processes

Weam M. Farid / Frank J. Mitropoulos
Graduate School of Computer and Information Sciences
Nova Southeastern University
Fort Lauderdale, USA
weam@nova.edu / mitrof@nova.edu

Abstract— **Agile software development methodologies, such as Scrum, have gained tremendous popularity and proven successful in quickly delivering quality Functional Requirements (FRs). However, agile methodologies have not adequately modeled Non-Functional Requirements (NFRs) and their potential solutions (operationalizations) with FRs in early development phases. This research proposes three fundamental agile artifacts to model FRs, NFRs, and their potential solutions in a visual environment. First, FRs are modeled through Agile Use Cases (AUCs). Second, NFRs are modeled through Agile Loose Cases (ALCs). Third, NFRs potential solutions are modeled through Agile Choose Cases (ACCs). AUCs are newly proposed hybrid of use cases and agile user stories. ALCs are proposed loosely-defined agile NFRs. ACCs are proposed potential solutions (operationalizations) for ALCs. The three artifacts are combined in a visual framework to promote agile modeling of NFRs (primarily) and how they are linked to FRs. The artifacts are the building blocks of a more comprehensive framework for modeling NFRs in agile software development processes.**

Agile Requirements Modeling; Agile Use Case; Agile Loose Case; Agile Choose Case; NFRs; Scrum; NORMAP Methodology

I. Introduction

Agile software development methodologies, such as Scrum and Extreme Programming (XP), have been gaining momentum and global adoption and are expected to grow in the next 10 to 15 years [1]. However, the Software Engineering community is constantly facing challenges to deliver quality software under tight schedules, hence, leading developers to quickly start coding with "ad hoc short cuts" to accelerate the development phase [2]. However, such ad-hoc methods suffer from the lack of a well-defined structure.

More specifically, agile software development methodologies have not adequately identified, modeled, and linked Non-Functional Requirements (NFRs) and their potential solutions (operationalizations) with Functional Requirements (FRs) during early requirements analysis phases. Researchers agree that NFRs have been generally ill-defined during conventional requirements engineering phases and especially ignored in agile methodologies [3][4].

Academic research and industrial case studies suggest that agile software development methodologies, such as Scrum and Extreme Programming (XP), have been gaining tremendous popularity and proven to be successful in implementing and quickly delivering quality FRs [5][6]. However, non-functional requirements (and often crosscutting concerns such as security, performance, and scalability) have been traditionally ignored or at best ill-defined in agile environments [7]. This research study is aimed at introducing three new agile artifacts that can potentially fill this gap. The three artifacts are Agile Use Cases, Agile Loose Cases, and Agile Choose Cases that map to FRs, NFRs, and NFRs potential solutions, respectively.

This paper presents three building blocks of a lightweight engineering-based yet agile methodology for identifying, linking, and modeling NFRs with FRs that can potentially improve software quality and support agility. The paper is organized as follows. Section 2 presents a brief literature survey and objectives of the study. Section 3 describes the details of the three new agile artifacts and their modeling framework. Section 4 summarizes initial results obtained from applying the new artifacts in a case study. Section 5 discusses the findings and presents areas for future work.

II. Literature Survey and Objectives

A. Insufficient Support for NFRs in Agile Processes

It was clearly stated in [6] that there was a significant lack of NFRs identification, modeling, and linking with FRs particularly in agile environments. Another research effort that supported this claim was the one conducted by [8] when they stated that agile development methods are sometimes criticized for not having explicit practices for NFRs. The authors proposed the Performance Requirements Evolution Model (PREM) but it focused more on the specification and testing of NFRs as opposed to modeling requirements and it only focused on one NFR (performance) as opposed to a generic framework that could be easily adapted to most NFRs.

Research by [9] also confirmed the semantic gap that exists between performance crosscutting concerns and functional concerns, leading developers to ignore the issue altogether. Many researchers claimed that NFRs were typically dealt with as an "afterthought" process and not treated as first-class artifacts during the software requirements phase [5]. Unlike common misperceptions about Scrum lacking engineering practices, Scrum indeed requires engineering excellence to accomplish inspection and adaptation [10]. Therefore, merging engineering with agile practices, in fact, is not an optional or a desirable characteristic, it is a "must". An unquestionable need for engineering of non-functional requirements in agile processes was well articulated when [11] stated the following:

"We understand that there is a need and an opportunity for agile methods to include techniques that make it possible to identify non-functional requirements early on and to describe them in such a way that an analysis may happen before implementation. Ideally these NFRs would have to be dealt with at the same time that User Stories are scoped with customers."

B. *Objectives*

According to [12], defined processes are vital elements in software process improvement but in order to realize the benefits, the model should simplify the complex real-world. The fundamental objective of this study is to research and develop a simple yet systematic non-functional requirements modeling framework that is specifically tailored for agile software development processes such as XP and Scrum. The building blocks for such a framework include the three agile artifacts addressed in this paper, namely, Agile Use Case (AUC), Agile Loose Case (ALC), and Agile Choose Case (ACC). The objectives of those artifacts are as follows:

- Integrate FRs and NFRs under one agile framework.
- Treat NFRs as first-class artifacts and link them to FRs to improve software quality during early requirements analysis phases without compromising agility.
- Enhance the agile user story card to a new W^8 Story Card Model that captures functional or non-functional agile requirements.
- Propose a taxonomy that visually classifies NFRs and their potential solutions into three types: source code, architecture and design, and organizational policies.
- Improve visualization through a downscaled version of Chung's NFR Framework [13] with new color-coding of clouds that is suitable for agile processes to model NFR soft goals and their potential solutions.
- Provide support for manual agile process through color-coded physical 3x5 index cards with "hook" points (called *pointcuts* in Aspect-Oriented Software Development) and color-coded soft goal clouds for improved visualization and efficiency.

III. NOVEL AGILE MODELING ARTIFACTS

A. *Background*

In order to establish the method for identifying, linking, and modeling NFRs with FRs in agile processes, a number of enhancements had to be made. First, the W^8 User Story Card Model is proposed as an enhancement of the agile index card technique used to gather high-level agile requirements in XP and Scrum. The story card is a physical 3x5 index card used to capture 3-5 sentences of high-level requirements from the stakeholders' point of view. Despite the recommendation of [14] that requirements should be expressed in one sentence and avoid complex sentences (which may contain more than one requirement), capturing a user requirement in 1-2 sentences will almost certainly not suffice. On the other hand, the traditional use case model is heavyweight and stifles agile processes which value working software over documentation. A good balance between the two approaches was to capture the most important elements of a user requirement, along with essential parameters, and follow certain guidelines that will be described shortly.

Second, a proposed Agile Requirement (AR) taxonomy was classified as either functional (Agile Use Case) or non-functional (Agile Loose Case). An Agile Choose Case is also part of the proposed taxonomy and represents the potential solutions (operationalizations) of NFRs.

B. *The W^8 User Story Card Model*

The W^8 User Story Card Model has eight "W"s that capture the most essential and basic information of an agile user story, such as the "who", "what", "why", "when", etc. Table I summarizes each "W" in the W^8 Story Card Model.

TABLE I. DATA ELEMENTS OF THE W^8 USER STORY CARD

ID	*Element*	*Description*
1	Who	Captures the actor (e.g., "As a system administrator…")
2	What	Captures the desired core functionality that the user wants (e.g., "I want to…")
3	Why	Captures the reason or business justification for the functionality (e.g., "so that I can…")
4	Without ignoring	Captures linguistically significant terms that might capture essential system qualities (e.g., "without ignoring security, scalability…") (required NFRs)
5	While it's nice to have	Captures optional quality attributes that are of lesser importance (e.g., "while it's nice to have ease of use and trainability...")
6	Within	Captures the time frame within which the desired functionality is needed (e.g., "within 12 weeks…")—just an initial time estimate according to business and market conditions
7	With a priority of	Captures the requirement's initial priority from a business value standpoint (priority of 1 means top priority and 99 means last). *Note: this research introduced two additional priority schemes, Riskiest-Requirements-First and Riskiest-Requirements-Last*
8	Which may impact	Captures a list of requirements dependencies impacted by this user story (e.g., "which may impact requirements # 101, 105, 107…")

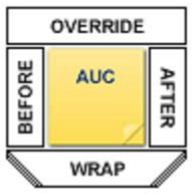
In the W^8 Story Card model, clouds represent Agile Loose Cases, which model non-functional requirements (soft goals), or Agile Choose Cases, which model the soft goals operationalizations according to Chung's NFR Framework [13]. The light-colored non-bold cloud is a minor enhancement that represents an ALC (green for source code such as an Aspect, red for architectural NFR, and blue for organizational policy NFR) while the dark-colored bold cloud represents an ACC. An Agile Choose Case—which is adapted from Chung's framework—models one or more potential solutions to the non-functional requirement. Our novel scheme suggests that solutions can be source code (e.g. an Aspect) which is color-coded as a dark green cloud, architectural which is color-coded as a dark red cloud, or organizational policy which is color-coded as a dark blue cloud.

C. *Agile Use Case (AUC)*

Researchers in [15] proposed an approach to extend traditional use case modeling to integrate FRs and NFRs and to identify crosscutting use cases. Our research, which is part of the Non-functional Requirements Modeling for Agile Processes (NORMAP) Framework, did not extend the existing traditional use case model, but rather, it combined a simplified version of it with an expanded version of a typical user story in agile processes and proposed the Agile Use Case. An AUC is a combined version of a simplified use case with an enhanced user story. In NORMAP, an Agile Requirement can be initially unclassified (neither as an Agile Use Case nor as an Agile Loose Case). The NORMAP framework proposed a taxonomy that classified an AR into three types: a) a pure AUC (functional with no matching ALCs); b) AUC with matching ALCs (both functional and non-functional); or c) a pure ALC (non-functional without any link to a particular functional requirement). The latter case is possible because it can be a global (system-wide) non-functional requirement that does not affect or link to any particular functional requirements. In this case, an AR should be modeled simply as an ALC cloud. Finally, if the AR identifies possible ALCs, then it will be flagged as an AUC with related ALCs.

Another aspect of the enhancements proposed in this methodology is to model the well-known Aspect-Oriented "pointcut" operators (also known as Composition Rule Operators). The operators simply indicate at which point the aspect will be "weaved" into the core functionality in order to remove code tangling and eliminate crosscutting concerns [16]. This simple idea is expanded beyond the source code level and is applied to any type of proposed solution (aspect or source code, architectural, or organizational policy). A pointcut simply refers to a point in time and sequence of performing the ACC, regardless of whether it is a source code, an architectural decision, or an organizational policy that should be enforced. When a potential ACC is required to be implemented before or after an AUC, then it must be linked to the "Before" or "After" pointcuts, respectively. If the ACC will override (totally replace) the entire AUC core behavior, then the "Override" operator is used. If the ACC is required before and after an AUC, then the "Wrap" operator is used. In the new W^8 model, the 4 sides of the physical index card (depicted as a yellow sticky note) modeled these 4 composition rule operators. The left edge of the story card emulates the "Overlap.Before" pointcut and would be "hooked" to any non-functional behavior/action that must be completed before the start of the corresponding functional requirement. The right edge of the card would emulate the "Overlap.After" pointcut and would hook any behavior/action that must be performed after the corresponding functional requirement. The top edge of the card would emulate the "Override" pointcut (totally replacing the core functional requirement with a non-functional behavior/action), while the bottom edge would emulate the "Wrap" composition rule operator (less obvious than the other three).

TABLE II. AGILE USE CASE STRUCTURE

AUC Components	
Graphic	**Description**
	An AUC is modeled like a yellow sticky note surrounded by four "hook" points (pointcuts)
	Before Pointcut: ALC connects to AUC before the behavior of the AUC is executed
	After Pointcut: ALC connects to AUC after the behavior of the AUC is executed
	Override Pointcut: ALC behavior connects to and overrides the behavior of the AUC
	Wrap Pointcut: ALC behavior connects to AUC before and after the behavior of the AUC is executed

D. *Agile Loose Case (ALC)*

An Agile Loose Case represents a specific instance of an identified NFR for a particular user requirement. The term "Loose" stems from the notion that non-functional requirements are "soft" goals according to Chung's NFR Framework [13] and that such "soft" goals are usually implicit and difficult to identify (i.e. ill-defined or 'loose'). An ALC models not only a high-level NFR, but it also links to one or more Agile Choose Cases, which represent potential solutions (code-level aspects, design constraints and architecture, or organizational policy). Furthermore, an ALC also captures other basic project management parameters such as relative size estimates (Fibonacci sequence), estimate confidence factors, risks, business priority and others. In NORMAP, when a possible ALC is identified and suggested, the agile team can either accept or reject it. If NORMAP fails to identify a valid ALC (or if the team wants to add a new NFR), the agile team can manually add it. Depending on the type of every ALC, specific relevant attribute values may be stored. Moreover, an ALC may also be related to one or more other ALCs (i.e., other sub-soft goals in accordance with Chung's NFR framework [13]). The NORMAP framework stipulated that an Agile Loose Case must have at least one Agile Choose Case (ACC), which must capture the operationalization or potential solution for the respective NFR. In addition, if the ALC has more than one ACC (potential solution), Boolean operators (e.g., "AND" and "OR") must be visually used to relate the potential solutions to "satisfice" (partially satisfy) the parent ALC (which still complies with Chung's NFR framework). ALC types are shown in Table III.

TABLE III. AGILE LOOSE CASE TYPES

ALC Types		
NORASP	***NORARC***	***NORPOL***
Green non-bold cloud models source code-based type ALC (i.e., Aspects)	Red non-bold cloud models architectural and design constraints type ALC	Blue non-bold cloud models "soft" organizational policies type ALC

E. *Agile Choose Case (ACC)*

An Agile Choose Case represented one or more instances of potential solutions for the identified ALC. An ACC modeled not only the potential solution (i.e., operationalization according to Chung's NFR framework [13]), but it also captured other basic project management information such as relative size estimate (effort), estimate confidence, risk of implementing the solution (automatically calculated in real-time), business priority, whether the potential solution is a preferred one or not, and others. The project-related parameters (along with requirements quality metrics) were essential in computing an improved risk-driven requirements implementation sequence plan (NORPLAN). NORPLAN was then used to generate a visual representation of the final implementation sequence in a tree-like view (NORVIEW).

In the NORMAP framework, a second taxonomy was proposed to classify the potential solutions represented in an ACC into three types (Table IV): a) NORASP; b) NORARC; c) NORPOL. NORASP (Non-functional Requirement Aspect) ACC was modeled as a source code-level potential solution which may utilize an "Aspect" from the field of Aspect-Oriented Software Development. By definition, an aspect is implemented to eliminate code tangling and scattering problems and crosscutting concerns found in NFRs. And since many NFRs are considered primarily crosscutting concerns—which impact at least one or more other functional or non-functional requirements—implementing an aspect at the source code level made it a prime choice and potentially a good solution. A NORASP ACC was visually modeled as a dark green bold cloud, which was usually (but not necessarily) linked to a soft green non-bold cloud that represented an ALC.

NORARC (Non-functional Requirement Architecture) ACC was modeled as a high-level architectural solution or design constraint for satisficing (i.e., partially satisfying) the NFR soft goal represented by the ALC. For example, an architectural solution such as "Duplicate Server" could be used as a potential solution for the "Availability" ALC. The Availability ALC is the equivalent of a soft goal NFR in Chung's NFR Framework [13]. Because this solution did not constitute a source code-based solution, it had to be visually modeled differently. Therefore, a NORARC ACC was modeled as a dark red bold cloud, which was usually (but not necessarily) linked to a light red non-bold cloud that represented its ALC parent.

NORPOL (Non-functional Requirement Policy) ACC modeled a high-level organizational policy solution that was non-technical in nature and satisficed a soft goal NFR in the ALC. For example, an organizational policy may mandate "Monthly Training" as a potential solution for the "Trainability" soft goal NFR represented in an ALC. A NORPOL ACC was modeled as a dark blue bold cloud, which was usually (but not necessarily) linked to a light blue non-bold cloud that represented its ALC parent. To ensure Accessibility for requirements engineers with color-deficient vision, each colored ACC cloud icon included the name of the ACC type inside the cloud as shown in Table IV for improved visualization.

TABLE IV. AGILE CHOOSE CASE TYPES

ACC Types		
NORASP	***NORARC***	***NORPOL***
Green bold cloud models source code-based type ACC (i.e. Aspects)	Red bold cloud models architectural and design decision type ACC	Blue bold cloud models "soft" organizational policies type ACC
NORASP ACC	NORARC ACC	NORPOL ACC

IV. CASE STUDY WITH THE EUROPEAN UNION ELECTRONIC PROCUREMENT SYSTEM

A. *Background*

One of the case studies used in this research for validating the NORMAP Methodology utilized the requirements model of the European Union (EU) eProcurement Online System [17]. The requirements in the EU case study were inter-related and represented one coherent system. However, the validation process in this case study focused primarily—among other aspects such as parsing and NFR classification—on visually identifying and linking AUCs to ALCs and their corresponding potential solutions (ACCs). Other aspects that are outside the scope of this research paper include computing and visualizing an improved risk-driven agile requirements implementation sequence given different priority schemes, risk factors, and the metrics and impacts associated with requirements.

B. *eProcurement System Requirements*

The EU case study [17] included 26 functional requirements from which other non-functional requirements were identified using the NORMAP Methodology. NORMATIC, the semi-automated tool that was developed to simulate the NORMAP Methodology, was fully or partially successful in identifying 16 non-functional requirements given only the detailed description of all 26 functional requirements. In this case study, the following 16 ALCs were identified: Confidentiality, Security, Auditability, Usability, User Interface, Accessibility, Availability, Multilingual Support, Documentation, Accuracy, Compliance, Efficiency, Performance, Interoperability, Scalability, and Configurability.

Using NORMATIC, the W^8 User Story Card Model captured the main theme of each requirement. This system parsed each sentence and prompted the requirements engineer to either accept or reject the parsed sentence as the main user story, hence, ignoring secondary or irrelevant information. ALCs were identified and linked to AUCs and potential solutions (ACCs) were selected and linked to their corresponding ALCs.

C. *Building the Initial Model*

Modeling the eProcurement system's functional and non-functional requirements and their potential solutions using the new agile artifacts is summarized in Table V. Some steps include details that are outside the scope of this paper and will be addressed in separate future papers. NORMATIC, a Java-based visual tool was implemented to simulate AUCs, ALCs, and ACCs.

TABLE V. AGILE REQUIREMENTS MODELING STEPS

Step	***Description***
1	Manually entering all 26 functional requirements as 26 Agile Use Cases. For each AUC, the entire original description of each requirement statement was entered, parsed, and processed by the system. In multi-sentence stories, only one sentence was selected as the main user story.
2	Identifying and classifying possible Agile Loose Cases for each of the 26 Agile Use Cases.
3	Linking identified ALCs with their corresponding AUCs as well as linking the ALCs with their corresponding ACCs (potential solutions). Each AUC has four "weaving points" (pointcuts): *Before*, *After*, *Override*, and *Wrap*. Depending on the context and understanding of each requirement, ALCs were connected to the appropriate pointcuts.
4	Additional steps (outside the scope of this paper) were carried out, such as, estimating and entering project management metric values and their impacts, conducting three experiments using different proposed priority schemes, calculating and updating risk scores for every requirement in the model, and visualizing an improved requirements implementation sequence. Details of those additional steps and results will be reported in separate future papers.

D. Modeling Example

Figure 1 represents the modeling of the first three of the 26 requirements in the eProcurement system. Each functional requirement is represented as an AUC (yellow sticky note icon surrounded by four pointcuts). Each AUC is linked to one or more ALC (light-colored clouds) and each ALC is linked to one or more ACC (dark-colored clouds).

The first AUC is related to the user registration requirement which included the "Confidentiality" ALC (light red cloud) connected to the "Wrap" pointcut as confidentiality must be maintained before and after the user registration process is carried out.

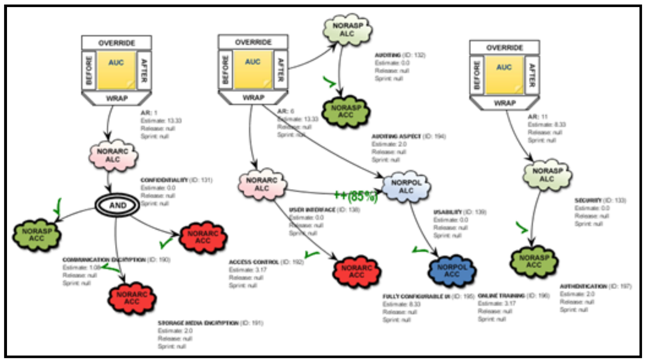


Figure 1. Agile Modeling of the User Profiling Requirement

Confidentiality can include potential solutions such as Communication Encryption (dark-green cloud), Storage Media Encryption (dark-red cloud), and Access Control (dark-red cloud). Communication encryption requires source-code level encryption of the communication stream, therefore, it needed to be modeled as a NORASP ACC. NORASP ACC is typically implemented as an Aspect (from Aspect-Oriented module). Storage media encryption and access control can be implemented through architectural and design constraints, therefore, they were modeled as NORARC ACCs (dark-red clouds). It should be noted that because all three potential solutions are required to satisfy the Confidentiality NFR, the "AND" operator was used to connect all three solutions to the Confidentiality ALC.

The second requirement is the User Profiling requirement (modeled in the middle of Figure 1 as the second AUC yellow icon). This requirement included three different ALC types: Auditability (NORASP), User Interface (NORARC), and Usability (NORPOL).

Since the Auditability ALC required capturing and storing a transaction trail of every user action performed in the system, a potential source code-based Aspect can be written and weaved into the AUC's *After* pointcut to capture user identity, actions performed, date and time after the functionality is executed. Therefore, the Auditability ALC and the Auditing Aspect ACC were modeled as Non-functional Requirement Aspect (NORASP) (light and dark green clouds, respectively).

The second requirement also included a User Interface ALC. The requirement stated that users should be able to select and save their user interface preferences (e.g., how data is searched and displayed). As a result, Fully Configurable User Interface was identified as a potential architectural ACC (dark red cloud) that was linked to its associated ALC (light red cloud), which in turn was linked to the AUC via the *Wrap* pointcut as shown in Figure 1.

Finally, the second requirement also included a Usability ALC (light blue cloud), which was also identified and was addressed by an organizational policy solution, such as requiring users to attend online training (dark blue cloud). Furthermore, addressing the User Interface ALC could also "strongly help" the Usability ALC, which was modeled by the "++(85%)" labeled on the edge that connected the two ALCs. A green checkmark on the edge between an ALC and an ACC indicated that the potential solution matched the type of non-functional requirement. If the requirements engineer attempted to link an ALC to an ACC of the wrong type, NORMATIC would flag it as an invalid link between incompatible types and would mark it with a red "X" on the edge.

The third requirement explicitly stated that users must be authenticated. Therefore, security was correctly identified as an appropriate ALC. The potential solution was to obviously provide source-code level authentication, which is why it was modeled as a dark-green cloud. Furthermore, because security must be established before and after the authentication process is completed, it was linked to the "Wrap" pointcut.

E. Results and Analysis

In this case study, there were 26 functional requirement statements with well over 100 sentences of requirement descriptions. Each of the 26 functional requirement statements was modeled as an Agile Use Case—which is modeled as a yellow sticky note surrounded by four pointcuts.

There were 57 sentences that included possible non-functional requirements. The NORMAP Methodology was successful in fully or partially parsing and classifying 50 out of 57 user requirement sentences that contained non-functional requirements—an overall success rate of 87.71%. The NORMAP methodology was fully successful in classifying 42 requirement sentences and partially successful in classifying 8 sentences.

In this case study, the following 16 ALCs were identified: Confidentiality, Security, Auditability, Usability, User Interface, Accessibility, Availability, Multilingual Support, Documentation, Accuracy, Compliance, Efficiency, Performance, Interoperability, Scalability, and Configurability. Each ALC was modeled as a light-colored non-bold cloud with three possible colors: light-green, light-red, and light-blue. Light-green modeled ALCs that can potentially be solved by a source code solution (such as Aspects from Aspect-Oriented Software Development). Light-red clouds modeled ALCs that can potentially be solved by architectural and design constraints (such as load balancing hardware for Performance ALCs). Light-blue clouds modeled ALCs that can potentially be solved by organizational policies (non-technical).

Potential solutions (operationalizations) were modeled as Agile Choose Cases and were modeled as dark-colored bold clouds with three possible colors: dark-green (such as Aspects), dark-red (architectural decisions), and dark-blue (organizational policy or non-technical solutions).

Agile artifacts were linked in three different configurations: AUC-to-ALC, ALC-to-ACC, and ALC-to-ALC. In the first case, a functional requirement is linked to a non-functional requirement. In the second case, a non-functional requirement is linked to a potential solution. In the third case, a non-functional requirement is positively or negatively impacting another non-functional requirement. This last case was modeled by selecting a value between 0 and 100 where 0-25% indicates that one ALC "breaks" another ALC (i.e. severe negative impact). When one ALC "hurts" another ALC, then the impact score should be between 26-49% (partially negative). When one ALC "helps" another ALC, the score should be 50-80% (partial positive impact). Finally, when one ALC "makes" another ALC, the score should be 81-100% (i.e. absolute positive impact). The value of impact between any two ALCs is one of many factors used to calculate a composite risk score for any given ALC.

The ability to visually distinguish between functional and non-functional requirements was of paramount importance in visual modeling of requirements in agile processes. The visual cues helped distinguish the different artifacts proposed in this study, such as Agile Use Cases (functional requirements), Agile Loose Cases (non-functional requirements), and Agile Choose Cases (potential solutions for those NFRs). The different color coding scheme could potentially improve agility in visually distinguishing between source code-based NFRs and solutions (green clouds), architectural and design constraints NFRs and their solutions (red clouds), and non-technical organizational policies (blue clouds). Furthermore, the ability to visually depict the instant of time at which an Agile Loose Case is linked to an Agile Use Case was extremely helpful using pointcut operators from the field of Aspect Oriented Software Development (Before, After, Override, and Wrap operators). Agility was also achieved through utilization of Natural Language Processing (NLP) tools that were used to parse and classify ALCs given a few requirement sentences, visual modeling of a risk-based requirements implementation sequence (i.e., the agile project plan), as well as utilizing agile project management metrics were all used to achieve agility without compromising a lightweight engineering process. In summary, the three basic artifacts (Agile Use Case, Agile Loose Case, and Agile Choose Case) form the fundamental building blocks of a more comprehensive requirements modeling framework that is specifically tailored toward agile processes such as Scrum. Future research papers will shed more light on other aspects of this agile requirements modeling framework for NFRs.

F. *Summary of the case study*

The case study used in this research utilized the requirements model of the European Union (EU) eProcurement Online System. The requirements in the EU case study were inter-related and represented one coherent system. The eProcurement system had 26 functional requirement statements, each of which consisted of multiple sentences. Results showed that 57 sentences that included possible NFRs. The NORMAP Methodology was successful in fully or partially parsing and classifying 50 out of 57 user requirements sentences that contained implicit non-functional requirements—an overall success rate of 87.71%. In addition, the case study utilized visualization features of the NORMAP Methodology that identified, modeled, and linked ALCs to other ALCs, ALCs to their potential ACC solutions, and ALCs to AUCs. The case study highlighted potential use of the concept of Pointcuts (*Before*, *After*, *Override*, *Wrap*—adapted from Aspect-Oriented Software Development) at which ALCs (NFRs) can be "weaved" into their associated AUCs (FRs) in a visually appealing, color-coded, and easy to understand lightweight engineering process using those three artifacts.

V. Conclusion and Future Work

This research proposed three fundamental agile artifacts to model functional requirements, non-functional requirements, and non-functional requirements operationalizations (potential solutions) in a visual environment used in agile processes such as Scrum. This research study is part of the NORMAP Methodology which was concerned with the identification, modeling, and linking of agile NFRs (Agile Loose Cases) and their potential solutions (Agile Choose Cases) with their functional counterparts (Agile Use Cases). An Agile Use Case is modeled as a yellow sticky note surrounded by four "weaving" points (i.e., pointcuts in Aspect-Oriented Software Development), namely, *Before*, *After*, *Override*, and *Wrap*. An Agile Loose Case is modeled as a light-colored cloud with three possible colors: light-green for source code type ALC (e.g., an Aspect), light-red for architectural and design constraint type ALC, and light-blue for organizational policy type ALC. An ALC can be linked to an AUC through one of the four pointcuts of an AUC. An Agile Choose Case is modeled as a dark-colored bold cloud with three possible colors: dark-green for source code type ACC, dark-red for architectural and design constraints type ACC, and dark-blue for organizational policy type ACC.

A visual modeling simulation tool, NORMATIC, was designed and developed in a Java-based environment that utilized visualization frameworks, such as JUNG [18] and Prefuse [19]. One case study was used to validate the new artifacts.

The EU eProcurement System Requirements model was particularly effective in visualizing and modeling functional and non-functional requirements dependencies, how ALCs can positively or negatively impact other ALCs as well as AUCs, and how ALCs can be linked to their corresponding ACCs. In addition, the development of the visual framework tremendously helped in visually differentiating between functional and non-functional requirements as well as potential solutions with different shapes and color-coded cloud icons which can all be used in agile processes, such as Scrum.

This research study hypothesized that agile processes can still benefit from lightweight engineering processes that can capture NFRs as first-class artifacts early on, and not treating them as an afterthought process. Visualization tools can be used to model primarily agile non-functional requirements (ALCs) and their potential solutions (ACCs), as well as agile functional requirements (AUCs) in an easy to understand and visually appealing environment without compromising agility.

The results of this research highlight the need to validate the new agile artifacts in real-world agile development projects and measure the effectiveness of using color-coded visual components in lightweight engineering-based agile processes, such as Scrum. Other areas include providing mobile and service-oriented architecture-based agile modeling services.

References

[1] G. Goth, "Agile Tool Market Growing with the Philosophy", 2009, *IEEE Software,* 26(2), pp. 88-91.

[2] A. Sampaio, N. Loughran, A. Rashid, and P. Rayson, "Mining Aspects in Requirements", *In Early Aspects Workshop, International Conference on AOSD*, 2005.

[3] N. Mead, V. Viswanathan, and D. Padmanabhan, "Requirements Engineering into the Dynamic Systems Development Method", *In Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference,* 2008, pp. 949-954.

[4] J. Araujo and J. Ribeiro, "Towards an Aspect-Oriented Agile Requirements Approach", *In proceedings of the Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, 2005, pp. 140-143.

[5] M. Qasaimeh, H. Mehrfard, and A. Hamou-Lhadj, "Comparing Agile Software Processes Based on the Software Development Project Requirements", *In Proceedings of 2008 International Conferences on Computational Intelligence for Modeling, Control and Automation; Intelligent Agents, Web Technologies and Internet Commerce; and Innovation in Software Engineering*, 2008, pp. 49-54.

[6] A. Marcal, F. Furtado Soares, and A. Belchior, "Mapping CMMI Project Management Process Areas to SCRUM Practices", *In 31st IEEE Software Engineering Workshop (SEW 2007)*, 2007.

[7] F. Paetsch, A. Eberlein, and F. Maurer, "Requirements Engineering and Agile Software Development", *In IEEE Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2003, pp. 308.

[8] C. Ho, M. Johnson, L. Williams, and E.M. Maximilien, "On Agile Performance Requirements Specification and Testing", *In Proceedings of the AGILE Conference (AGILE'06)*, 2006, pp. 47-52.

[9] M. Woodside, G. Franks, and D. Petriu, "The Future of Software Performance Engineering", *In Future of Software Engineering (FOSE '07)*, 2007, pp. 171-187.

[10] K. Schwaber, *Agile Project Management with Scrum*, Redmond, Washington, Microsoft Press, 2004.

[11] A. Eberlein and J. Leite, "Agile Requirements Definition: A View from Requirements Engineering", *In Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE'02)*, 2002.

[12] A.M. Christie, "Simulation in support of CMM-based process improvement", *Journal of Systems and Software* (46), 1999, pp.107-112.

[13] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional Requirements in Software Engineering*, Boston, MA, Kluwer Academic Publisher, 2000.

[14] S. Robertson and J. Robertson, *Mastering the Requirements Process*, ACM Press Books, Addison-Wesley, Harlow et al., 1999.

[15] J. Araújo and M. Moreira, "An Aspectual Use-Case Driven Approach", *In VIII Jornadas Ingeniería del Software y Bases deDatos (JISBD 2003)*, 2003, pp. 463-468.

[16] G. Sousa, G. Silva, and J. Castro, "Adapting the NFR Framework to Aspect- Oriented Requirements Engineering", *In The 17th Brazilian Symposium on Software Engineering (SBES)*, 2003, Manuas, Brazil.

[17] European Dynamics S.A., "Functional Requirements for Conducting Electronic Public Procurement Under the EU Framework Volume I", Internet: http://ec.europa.eu/internal_market/publicprocurement/docs/eprocurement/functional-reguirements-vol1_en.pdf, 2005 [Oct. 13, 2009]

[18] JUNG Development Team, "The Java Universal Network/Graph Framework", Internet: http://jung.sourceforge.net, 2003 [May 15, 2010].

[19] Prefuse Development Team, "Prefuse: The Information Visualization Toolkit", Internet: http://prefuse.org, 2007 [May 17, 2010].