Meeting Quality Demands: A Case for Improving Non-Functional Requirements Literacy

***Janet T. Jenkins* [1]*, Randy K. Smith* [2]**

Abstract – This brief report studies the perceived importance of functional requirements and non-functional requirements by university SE/CS students, SE/CS faculty and practicing professionals. The study centers on these communities' view of non-functional requirements. The contrast is stark and points to the need for SE/CS education to not only increase the depth of coverage but also emphasize the importance of NFRs to the overall success of a software effort. Of course not all SE/CS programs share the same emphasis. This paper provides anecdotal evidence to persuade or reinforce based on a programs current leaning.

Keywords: Software Requirements, Requirements Engineering, Software Engineering Education

INTRODUCTION

In the field of software engineering, much focus is placed on eliciting, defining, and implementing functional requirements. These types of requirements focus on the basic services of the software system. However, meeting only functional requirements may not produce a product that meets all the user's implicit intentions (or constraints) for the product. Such easily missed intentions are categorized as non-functional requirements (NFRs) and can be difficult to elicit, define, and implement.

The difficulty including non-functional requirements in the software process contributes to the neglect of NFRs by software developers. There are many characteristics of NFRs that make them difficult to include in the software process. Non-functional requirements are often implicit requirements. Stakeholders focus on the functional needs of the system and may not be aware of certain non-functional needs. This makes gathering NFRs difficult. Even if NFRs are elicited, there is no universal method of representing them in the software design. Methodology for modeling and representation of requirements are centered on functional requirements. Moreover, certain types of NFRs are difficult to quantify. Without some type of metric assigned to the NFR, it can be complicated to verify the requirement has been met.

In this work, we sought to determine what theories had been proposed to address the inclusion of NFRs into the software process. Further, we were interested if these approaches had carried over into industry and academics. Additionally, we wanted to know if those in industry and academics deemed NFRs as important in the software process and if level of importance could be raised among novices. We also sought to determine if experience played a part in the emphasis placed upon NFRs. We used a set of surveys as our instrument for making these determinations. Specifically, we examine how NFRs are classified, defined and represented in software designs by each group. We analyze the varying level of importance each group places on NFRs.

RELATED WORK

Functional requirements are the services a customer expects of the software system upon completion [Sommerville, 12]. The Software Engineering Body of Knowledge (SWEBOK) breaks down the requirements process into 4 categories: elicitation, analysis, specification, and validation [SWEBOK, 11]. Well-written functional requirements are "understandable by end-users, non-prescriptive, correct, complete, concise (succinct), precise, clear, unambiguous, consistent, traceable, modifiable, testable (verifiable), and feasible" [Hamlet, 7]. Assuming

[1] University of North Alabama, Box 5051, Florence, AL 35632, jtruitt@una.edu

[2] University of Alabama, Box 870290, Tuscaloosa, AL 35487, rsmith@cs.ua.edu

functional requirements meet these qualities, implementing and testing methodologies for functional requirements are common and can be straightforward.

Non-functional requirements, however, can be difficult to elicit due to their characteristics and often do not possess these qualities making them more difficult to represent and implement in a software system. Non-functional requirements include requirements that place limitations or constraints upon systems. Quality requirements that describe desirable attributes of a system fall in the category of NFRs [Boehm, 3]. NFRs are often referred to as the "ilities" of a system due to their names (reliability, usability, maintainability, etc.) [Mylopoulous, 9]. Non-functional requirements include requirements that place limitations and or constraints upon systems. They are often subjective and, therefore, not easy to quantify. Often these are not discovered until the system is in use and the absence of the NFR becomes a hindrance to the end user.

Non-Functional Requirement Categorizations

Non-functional requirements can be broken down into several categories and in multiple forms. The Rome Air Development (RADC) represents NFRs as consumer-oriented and technically-oriented [Bowen, 4]. RADC further extends the consumer-oriented approach to ask the questions "how well does it function, how valid is the design, and how adaptable is it" to be sure certain non-functional attributes are present in a system. These questions hone in on attributes such as efficiency, integrity, reliability, survivability, usability, correctness, maintainability, verifiability, expandability, flexibility, interoperability, portability, reusability, and are categorized in Table 1.

In contrast, Sommerville [Sommerville, 12] classifies non-functional requirements into three areas: process requirements (typically qualitative), product requirements (typically quantitative), and external requirements (legal, corporate, etc.).

The ISO/IEC 9126 Taxonomy of Quality Requirements begins with the highest level of requirements being functionality, reliability, usability, efficiency, maintainability, and portability, with other attributes falling under those classifications as seen in Table 2.

Table 1. RADC Categorizations [Bowen, 4]

Category	**Performance**	**Design**	**Adaptation**
Attributes	Efficiency Integrity Reliability Survivability Usability	Correctness Maintainability Verifiability	Expandability Flexibility Interoperability Portability Reusability

NFRs in Academics

First, we will look at the guidelines set in academics by ABET, Inc. for computer science and software engineering programs. We want to see what emphasis is placed upon NFRs by this accreditation body.

The Computing Curriculum 2001 recommends a total of 4 hours for software requirements and specifications for a typical four-year computing degree [Computing Curriculum 2001, 6]. Non-functional requirements are mentioned briefly as an addendum to functional requirements. The Software Engineering 2004 curriculum dedicates roughly 16 hours toward requirement elicitation, specification, documentation, and validation [Software Engineering 2004, 10]. About 1 hour of course time is explicitly listed for analyzing non-functional (or quality) requirements. Specific non-functional requirements that are intrinsic to certain areas are recommended. Areas, such as, security and safety in networking, performance in algorithm development and other areas, and usability in human and computer interaction are included throughout the recommendations of both CC2001 and SE 2004. SE2004 looks at two approaches of introducing software engineering ideals early in the curriculum. One suggestion introduces software engineering processes in the first year and another proposes the introduction into the second year. These recommendations do not specify the details of these early introductions.

Table 2. ISO/IEC 9126 Taxonomy of Quality Requirements

Quality Requirements	Sub-characteristics
Functional	Suitability Accuracy Interoperability Security Functionality compliance
Reliability	Maturity Fault tolerance Recoverability Reliability compliance
Usability	Understandability Learnability Operability Attractiveness Usability compliance
Efficiency	Time behavior Resource utilization Efficiency compliance
Maintainability	Analyzability Changeability Stability Testability Maintainability compliance
Portability	Adaptability Installability Co-existence Replaceability Portability compliance

As of November 2011, there are 21 accredited software engineering undergraduate programs [ABET, 1]. Many of these programs make efforts to at least introduce some software engineering ideals in the first year. Some simply introduce UML while others do a general overview of software engineering. Nine of these 15 programs have software engineering courses in the second year. These topics include testing, requirements, tools, design and architecture, component design, among others. Most of the software engineering related courses fall in the last two years. All schools have some version of a senior year project (typically two semesters). Six schools have a quality assurance course specifically for software engineering. Most of these courses deal with testing and validation and verification. Other schools cover quality assurance in other departments or in other courses, many times listed as an elective. Seven of the programs offer a course that focuses on requirements. The earliest offering of such a course is the second year in one program. One program offers it as a third year course. Other programs offer it as a fourth year, fifth year, or graduate course. Four of these courses make effort to mention non-functional requirements.

We see after an examination of the accredited software engineering programs that non-functional requirements as a whole are not emphasized early in the curriculum. They are considered in most programs, but certainly not to the degree of functional requirements. It does not appear that many of the ideas found in non-functional requirement research have yet reached emphasis in the curriculum. Note, however, some programs do offer security (5 elective, 3 required), human computer interaction (4 elective, 4 required), and application domain specific (6 programs require a focus in a specific domain) courses that will naturally cover specific non-functional requirements.

NFRs in The Software Engineering Body Of Knowledge

The Software Engineering Body of Knowledge (SWEBOK) provides a software engineering body of knowledge that sets out to generalize and cover common knowledge in the field of software engineering. SWEBOK is a project of the IEEE Computer Society Software Engineering Coordinating Committee. SWEBOK denotes the difference between functional and non-functional requirements and emphasizes the need to characterize a requirement as one or the other [SWEBOK, 11]. It also has a chapter dedicated to the quality of the software process overall. The method or timing of including non-functional requirements into the software process is not strictly specified. Requirements are discussed as a whole.

Summary

From the curriculum summary, we see that non-functional requirements are not explicitly introduced in the introductory phases of even the software engineering curriculum. Even though listed as a topic to be covered, they are secondary to functional requirements. There does not seem to be an apparent, common methodology for handling non-functional requirements in the software process. This observation can also be made from the guidelines set by SWEBOK. There would be little need of non-functional requirements without first having functional needs. However, simply complying to functional requirements, which in and of itself is not yet a perfected art, does not necessarily produce a software product that will please the customer. Chung et al. propose non-functional requirements need to become a more prominent part of the software process [Chung, 5]. Many have made practical suggestions about how to go about doing this and have shown some improvements to products due to these efforts.

METHODOLOGY

In this work, we focus on two research questions.

Research Question 1: Does awareness of non-functional requirements increase with experience?

As an initial step, this study examines if the importance of NFRs is apparent to software developers across varying levels of experience. We suggest as experience increases, the attention and importance placed on NFRs increases. The explicit attention to NFRs in the software process recognizes their importance on overall software quality.

Research Question 2: Are current ideas from non-functional requirement research being implemented?

Our review of the literature shows some of the issues underlying the neglect of NFRs. The subjectivity of many non-functional requirements is the root of the difficulty, making NFRs difficult to elicit, represent, and measure. Current attempts to incorporate NFRs into the software process focus on this primary problem.

Surveys

In order to measure developers' awareness of NFRs and importance placed on NFRs in the software process, we conducted a series of surveys. The surveys were given to professionals, academics (professors/instructors), and novices (students). Through the survey, we sought to obtain information to guide techniques to encourage the inclusion of NFRs into the software development process. Three separate surveys were conducted, one per group. Although they contain common questions, some questions differ based upon the recipient of the survey.

Specifically, the three groups of participants were:

Novices in computing: We solicited students majoring in computing taking upper-level computer science courses (Junior to Graduate level) at 4 state institutions.

Academics in computing: This group was composed of faculty in computing at a variety of institutions. The survey was distributed to the Special Interest Group on Computer Science Education (SIGCSE) listserv and to faculty at various institutions including faculty at ABET accredited programs in software engineering.

Software Practitioners: This group was composed of various software practitioners in industry including a regional IEEE Computer Society, a national Computer/Technical Interest monthly mailing list, and numerous technology companies throughout the country.

Assessment

For Research Question 1, we sought to determine the importance placed on non-functional requirements by the three groups (industry, academia, novices) based upon their experience as software developers. We made observations to determine if there was a significant difference between various experience levels concerning the developer's emphasis on non-functional requirements. We based the measuring of this emphasis on rankings the participants placed on various requirements in general, on the way they defined and categorized NFRs, and on various techniques used to include NFRs into the software process.

We hypothesized there was a positive correlation between the emphasis placed on NFRs and the experience of the software developer. To make this assessment, we compiled results from several components to create an Experience Value and an Emphasis Value.

The Experience Value was composed of years spent in industry in software development, years teaching computing at the college level, the number of projects completed coupled with the average time length of projects worked on or completed, and the number of years spent in undergraduate school in a computing field. The Emphasis Value was created to represent the time preference of functional and non-functional requirements, appropriate definition of NFRs, labeling of NFRs, ranking based on importance of requirements, and attention spent on NFRs (usability, reusability, efficiency, maintainability, security) during software creation.

To calculate the correlation, Spearman's rho was used. With this method, each set of values is given a ranking value. After the ranks are assigned, Pearson's rho was used to calculate the correlation value.

For Research Question 2, we asked participants to specify how frequently they made use of specific techniques covered in basic software engineering or requirements engineering text. We calculated the percentages of participants who specified they used the techniques always, often, sometimes, rarely, or never. We also calculated the percentage of participants who did not respond to these questions. There were five elicitation and representation techniques and three quantification methods. We examined the usage of the techniques individually and in two major categories (elicitation and representation, quantification). When the categorical comparison is made, we break the novice use of the methods into Regular Use, Occasional to No Use, and No Response. Under the Regular Use category, we included responses Always and Often in regard to frequency of use. Responses Sometimes, Rarely, and Never were included in the Occasional to No Use category. The No Response category consisted of survey participants who did not answer the question.

Participant Descriptions

A total of 156 people distributed over three groups took part in the study. We received roughly 5000 responses to questions on surveys. The description of the three groups follows.

Novices in computing

This group was composed of students majoring in computing taking upper-level computer science courses. There were a total of 67 participants in the initial novice survey. *Of the 67 novices participating, there were 6 sophomores, 31 juniors, 24 seniors, and 5 graduate students.*

The mean number of programming courses taken by these students was 8.08 with a standard deviation of 5.15. Of all novices, 56.7% had worked part-time and 32.8% had worked full-time developing software. The largest portion of the group worked on projects that took less than 1 month to complete, which is not surprising since most student projects must be completed within an academic semester.

Novices were students taking upper-level computer science or software engineering courses. These students were majoring in computer science (50%), software engineering (18), management information systems (24), computer engineering (1%), and other computer-related majors (5%). There was a small group of miscellaneous majors (2%).

Academics in computing

This group was composed of faculty in computing at a variety of institutions. There were 52 academic participants. Of these participants, 80.4% had a doctoral degree and the remaining 19.6% had a master's degree. These academics had a variety of undergraduate majors. Those with computer science degrees comprised 33% of the participants, 19% had math undergraduate degrees, and 11% were computer engineering majors. Other majors included accounting, chemistry, computer engineering, English, linguistics, management information systems, math, math education, mechanical engineering, physics, and other computer-related fields.

As a group, the academics had a total of 631 years of teaching experience at the college level, with a mean of 12.62 years and standard deviation of 9.37. 84.6% of the participants had more than 5 years of teaching experience at the college level. 41% of the group was at the assistant professor level, 23% were associate professors, 20% were full professors, with the remaining academics being instructors, lecturers, and senior lecturers.

Each academic was asked to choose from a list of typical computing courses they had taught in the past 5 years. Over 60% of the group taught introductory programming courses and over 60% of the group taught a software engineering course. Several courses expounding on topics typically covered in a software engineering course were listed by many of the participants. It is interesting to note there were only two faculty members who had reported teaching a requirements engineering course in the past five years.

In addition to their academic responsibility, 54% of the faculty members participate in software development. Of the projects on which faculty work, 74.5% of the projects take more than six months to complete. Of the academics, 39.2% of the projects worked on or completed required more than a year to complete.

Software Practitioners

This group was composed of various software practitioners in industry. There were 37 software practitioners who participated in the study. Of this group, 45.9% had a bachelor's degree, 45.9% had a master's degree and 8.1% had a doctorate degree. There were 23 participants who majored in computer science comprising 62.2% of the total group.

The practitioners had a total of 479 years of full-time industry experience with a mean of 12.94 years and a standard deviation of 10.9. Of the practitioners, 59.5% had 5 years or more experience. Nine of the participants held the title software developer while the others held a wide variety of titles.

The total number of projects completed by this group is 754 with an average of 20.9 projects per practitioner and a standard deviation of 24.5. Clearly, there is a wide spread of the number of projects completed by this group. Of the projects completed or worked on by this group, 58.3% of the projects took longer than six months to complete.

Of the practitioners, 6 had also taught at the college level. Years of teaching experience range from one to twenty years with 4 having less than five years experience and 2 having more than thirteen and twenty years of teaching experience.

Results

We will examine how the experience of the participant relates to the emphasis placed upon non-functional requirements. We will then discuss reasoning behind the inattention to NFRs.

Experience and Emphasis

Research Question 1: Does awareness of non-functional requirements increase with experience?

One hypothesis of this research is that as developers gain experience, they also gain an appreciation for NFRs. This assumption comes from the notion that as a developer spends more time in industry developing software, the developer has the ability to also experience many failings of the software product and process from a non-functional or quality standpoint. Often non-functional requirements are implicit in nature. One who has more experience in eliciting, specifying and validating requirements would have more appreciation for the requirements that stakeholders often are not even aware they need. This experience would theoretically point the developer toward improving the product and process in ways outside of the purely functional.

We need a way to compare the level of experience of each participant with the level of emphasis upon NFRs of each participant. Based upon the data we collected, we have constructed an Experience Value (E_{XV}) and an Emphasis Value (E_{MV}) to make this comparison. E_{XV} is a weighted sum of experienced factors as provided in the surveys. E_{MV} is a weighted sum of 12 questions on the survey related to FR and NFR understanding. A complete description of E_{XV} and E_{MV} can be found in [Jenkins, 8].

Table 3 shows the information for the E_{XV} for each group including, mean, standard deviation, and the maximum and minimum values. Table 4 shows the same information for the E_{MV}. Note the highest weighted sum for E_{MV} is 360 points.

Table 3. Experience values descriptive statistics

Group	Average	Standard Deviation	Maximum	Minimum
Novice	7.74	6.39	44	2
Academic	56.06	36.09	166	3
Practitioner	43.64	29.68	106.66	7
Overall	32.53	33.66	166	2

Table 4. Emphasis values descriptive statistics

Group	Average	Standard Deviation	Maximum	Minimum
Novice	194.3	43.6	266	83
Academic	167.7	55.7	282	64
Practitioner	189.5	53.5	298	68
Overall	184.3	51.3	298	64

To determine if there was a correlation between emphasis on non-functional requirements and experience, we used Spearman's procedure and found no correlation could be determined ($r = -0.062$, p-value = 0.449) using all three groups together. This is a surprising finding, but it fits the information in Table 3 and Table 4. The average E_{XV} for the novices is much lower than the averages for academics and practitioners, yet the average E_{MV} for the novice group is higher than both the academic and practitioner groups. Even though the E_{MV} does not seem to increase as the E_{XV} increases, it does raise interesting questions to be explored. This finding is consistent with the literature review. It points to the prospect that NFRs are secondary to functional requirements in practice.

Initially, it appears developers simply do not regard the quality of the software product or process. However, when asked to rank a list of requirements (functional and non-functional), all groups rated the requirements representing usability and security the highest among all requirements presented. These two non-functional requirements certainly have a high priority among the participants. The problem could lie in the lack of consistent methods to incorporate NFRs.

Non-Functional Requirements in Practice

Research Question 2: Are current ideas from non-functional requirement research being implemented?

We will now consider the practice of emergent non-functional requirement strategies from research among novices, academics, and practitioners. Many emergent and applied techniques give suggestions for improving elicitation, specification, and quantification of NFRs. We questioned participants to determine if any of these ideas have been put into practice.

We asked the participants to specify how frequently they made use of the following elicitation and specification techniques:

- Soft Goal Hierarchy,
- Performance Cases,
- Misuse Cases,
- Language Extended Lexicon,
- Goal-Oriented Decomposition.

In addition, we asked the participants to specify how frequently they made use of the following quantification methods in regard to non-functional requirements:

- Requirement Hierarchy Approach which used percentages on the Quality Requirement taxonomy
- Using summation, maximum, and minimum nodes in soft goal graphs
- Planguage

For each group of methods, we categorized the groups into Regular Use, Occasional to No Use, and No Response. Table 5 shows the percentages of usage for the elicitation and representation techniques. Table 6 shows the percentages of use for the quantification techniques. From these tables, we can see the percentages of Regular Use are quite small, especially in the quantification methods. The No Response category is interesting here since these two questions received the highest no response rate compared with other questions in the survey. This suggests the participants were unfamiliar with the terms. It should also be noted that some of the methods in the elicitation and specification techniques are similar in name to common techniques used among software developers. For instance performance case and misuse cases may be mistaken for use cases. Although performance and misuse cases are derivations of use cases, they are not the same. This could account for the higher percentage of use in techniques for elicitation and representation.

Table 5. Usage of elicitation and representation techniques

Group	Regular Use	Occasional to No Use	No Response
Novice	17%	74%	10%
Academic	14%	67%	19%
Practitioner	16%	75%	9%

Table 6. Usage of quantification techniques

Group	Regular Use	Occasional to No Use	No Response
Novice	10%	78%	12%
Academic	3%	75%	22%
Practitioner	2%	77%	22%

We can see from the results the methods presented are not commonplace among developers. We further asked participants to specify other elicitation, representation, and quantification methods they used.

Comments from the novices support the conclusion that they have not been taught these specific techniques. The novice comments also point to the fact that any process novices use to incorporate NFRs is at best ad-hoc. One novice commented that his method of quantifying NFRs was a "shot in the dark."

Academics contributed more promising commentary. Although some stated they do not require specification of non-functional requirements in their classes, some offered references to techniques they teach to their students. One academic uses Scenarios as defined in Bass, Clements, and Kazman's book which provide a situational framework for each attribute [2]. Another uses Usage Centered Design for interface development by Constantine and Lockwood. Others mentioned testing, prototyping, and heavy customer involvement as casual methods.

There was a mixed review from industry participants. Some stated they did not incorporate non-functional requirements at all. One stated formal methods were used to incorporate non-functional requirements. One participant was adamant that for non-functional requirements to be incorporated into the software product, they must be defined in the elicitation process and assigned a metric. Many noted the subjectivity of quantifying non-functional requirements.

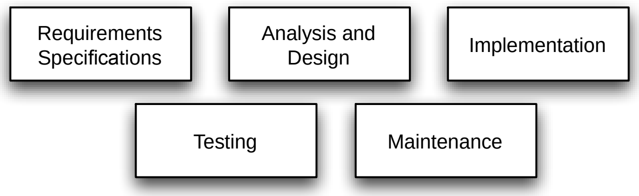
Overall, the novices seemed to be unaware of the implications of non-functional requirements. The academics are aware of them and some are apparently trying to offer some way for students to learn about them. However, no common curricular structure emerged. Likewise, no central theme among the practitioners, outside of the agreement non-functional requirements can often be too all-encompassing to specify and measure, was found.

Lack of Incorporation of Non-Functional Requirements

With the realization of non-functional requirements having little impact on methodologies in practice and in academics we examined possible reasons for their neglect. We have learned from the literature and from many of the comments shared from our participants some of the reasons. Non-functional requirements can be elusive to the requirements gatherer due to their implicit nature. Even if elicited, NFRs are difficult to specify in a way that is incorporated with functional requirements. This can also make NFRs difficult to quantify, which results in validating they have been met within the product equally difficult.

These are important reasons why non-functional requirements are often ignored in the software process, but can we glean from the information we gathered more reasoning to why this is? Let us first look at the phases of the software life cycle in which these participants believe time will be spent. We presented the participants with the general software life cycles phases in Figure 1.

Figure 1: Software Development Lifecycle Phases



We asked the participants to specify the percentage of total team calendar time spent on the average systems development project for each phase. Academics were asked to answer the question with regard to their recommendation to a novice. Table 7 shows the results. The academic recommends spreading time equally among the 5 phases. The practitioner sees the implementation phase as a significantly higher proportion than the other phases. There is a statistically significant difference (p-value = 0.0455, α=0.05) between the Implementation phase (35%) and the second highest practitioner phase Analysis and Design (18%). The Novice column in Table 7 shows the majority of time spent would fall under the Implementation category as well. There is a difference with α=0.10 between the Implementation phase and the second highest phases for the Novice.

Table 7. Average estimated time spent in each phase

Phase	Novice	Academic	Practitioner
Requirements Specification	17%	21%	12%
Analysis and Design	22%	24%	18%
Implementation	33%	24%	35%
Testing	22%	22%	16%
Maintenance	11%	10%	17%
Other	6%	3%	3%

When asked if participants thought they would spend more time on functional requirements or non-functional requirements, 80.7% of the overall group chose functional requirements. If the majority of developer time is spent in the Implementation Phase where much of the functionality of the software product is being created, it is clear why functional requirements would take up the majority of the time on a software project. Couple that fact with NFRs being difficult to include in any part of the software process and no standard procedure for ensuring their inclusion, the result is a software product that will not meet non-functional requirements.

Conclusion

In this study, we sought to increase awareness of non-functional requirements throughout the software process. The literature declares NFRs to be difficult to elicit, specify, and quantify. This is due to the subjective nature of many non-functional requirements. Current research offers some techniques to tackle these issues. We sought to explore the practices of professional software developers, professionals in the academic field, and novices to determine if there was a common understanding appreciation for non-functional requirements.

We found that awareness of non-functional requirements as defined by this study does not increase with experience. We found no common practice among any group we surveyed. We did notice many participants had no explicit protocol for managing non-functional requirements.

In addition, we would seek to find methodologies that are straightforward and fit naturally into current software models. Such methodologies would help software developers not only recognize the need for NFRs, but also have concise methods for incorporating NFRs into their work.

A portion of the study sought to determine which process models were used among these groups. It would be interesting to extend that portion of the study to see if certain software processes are more favorable to the inclusion of NFRs.

One practitioner shared "…there is always a trade-off between good enough and spending more time and money on it." Our future work would like to explore the cost-effectiveness of spending more time on non-functional requirements in industry.

REFERENCES

[1] ABET, http://www.abet.org/

[2] Bass, Len, Clements, Paul, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional. 2003.

[3] Boehm, Barry W. and H. Hoh. “Identifying Quality Requirement Conflict.” *IEEE Software* 13, no. 2 (1996): 25-36.

[4] Bowen, T.P., G.B. Wigle, and J.T. Tsai. *Specification of Software Quality Attributes*. Vol. 3, *Software Quality Evaluation* Guidebook. Rep. RADC-TR-85-37. New York: Rome Air Development Center, 1985.

[5] Chung, Lawrence, Brian Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Boston, Kluwer Academic Publishers, 2000.

[6] Computing Curricula 2001, http://www.acm.org/education/education/curric_vols/cc2001.pdf

[7] Hamlet, Dick and Joe Maybee. *The Engineering of Software.* Boston, Addison-Wesley, 2000.

[8] Jenkins, Janet, Increasing emphasis on non-functional requirements throughout the software process, Dissertation, University of Alabama, 2008.

[9] Mylopoulous, John, Lawrence Chung, and Eric Yu. “From Object-Oriented to Goal Oriented Requirements Analysis.” *Communications of the ACM*, 42, no. 1 (1999): 31-37.

[10] Software Engineering 2004. http://sites.computer.org/ccse/

[11] Software Engineering Body of Knowledge. http://www.swebok.org

[12] Sommerville, Ian, *Software Engineering*, 8th ed. Boston: Addison-Wesley, 2006.

Dr. Janet Truitt Jenkins

Dr. Janet Truitt Jenkins is a Computer Science instructor in the Computer Science and Information Systems Department at the University of North Alabama in Florence, Alabama. She completed her doctoral and master degrees in Computer Science from the University of Alabama in Tuscaloosa, Alabama. She also holds an undergraduate degree in Mathematics and Computer Science Education from the University of North Alabama. Dr. Jenkins’ research interests include requirements engineering and increasing the awareness of abstraction and generalization among math and computer science students.

Dr. Randy K. Smith

Dr. Randy K. Smith is an Associate Professor in the Department of Computer Science at The University of Alabama. Dr. Smith’s research interests are in the areas of Software Quality, Software Metrics and applications of Information Retrieval techniques to Software Evolution. Dr. Smith is a Senior Member of the ACM and IEEE. His research has been supported by numerous agencies including the National Science Foundation, NASA, and the U.S. Department of Homeland Security.