Toward Model-Based Requirement Engineering Tool Support

Hassan Reza
Rashmi Sehgal, University of California Santa Cruz
Dr. Jeremy Straub, North Dakota State University--Fargo
Nicholas Alexander, Marshall University
Toward Model-Based Requirement Engineering Tool Support

Hassan Reza  
Department of Computer Science  
University of North Dakota  
3950 Campus Road, Stop 9015  
Grand Forks, ND 58202  
701-777-4107  
reza@cs.und.edu

Rashmi Sehgal  
Department of Computer Science  
University of California Santa Cruz  
1156 High Street, Santa Cruz, CA  
95064  
rasehgal@ucsc.edu

Jeremy Straub  
Department of Computer Science  
North Dakota State University  
1320 Albrecht Blvd, Room 258  
Fargo, ND 58102  
701-231-8562  
jeremy.straub@ndsu.edu

Nicholas Alexander  
Division of Computer Science  
Marshall University  
One John Marshall Drive  
Huntington, WV 25755  
304-696-5453  
alexander50@live.marshall.edu

Abstract—The OpenOrbiter CubeSat Development Initiative is working to build a small spacecraft system using open source software and open hardware principles. Some important design considerations for the CubeSat are nonfunctional requirements. The key contribution of this work is presenting the design of a requirement engineering tool that can be used to speed up the requirement elicitation and specification of system-wide qualities such as availability, performance, and security. To this end, it uses quality attribute scenarios.

The meticulous implementation of a requirements engineering tool and its integration in the design and implementation process for the OOSDI CubeSat project is important as the communicability and proper documentation of nonfunctional requirements is one of the main issues in the successful development of mission critical systems such as a CubeSat spacecraft. Currently, these scenarios are created manually, one at a time, and are unable to be seamlessly integrated into a single, shareable, project requirement schematic.

To assist the spacecraft developers, a prototype software application, utilizing the concept of quality attribute scenarios originally proposed by Carnegie Mellon’s Software Engineering Institute (SEI), was designed and implemented. The quality attribute scenarios are the equivalent of UML scenarios and use-cases. They are built specifically to document nonfunctional requirements using the quality scenarios template.

This prototype system provides a single repository in which the nonfunctional requirements can be elicited, maintained and accessed by stakeholders. The use of this tool should be able to enable and simplify the process of documenting nonfunctional requirements and ensuring their verifiability and tractability.

This paper continues with a discussion of relevant background in Section 2. Section 3 describes the implementation of the CubeSat at the University of North Dakota. Then, in Section 4, the proposed requirement engineering tool that can be used to capture quality attributes (as discussed in [17]) is presented and applied to the engineering process of the CubeSat. Finally, in Section 5, the paper concludes with a summary of what has been determined from this exercise and a discussion of future work on the OpenOrbiter CubeSat project.

TABLE OF CONTENTS

1. INTRODUCTION ...................................................... 1
2. BACKGROUND ...................................................... 2
3. CUBESAT SOFTWARE SYSTEM ............................. 3
4. THE NEED FOR THE DEFINITION OF USE CASES .. 3
5. QUALITY TOOL .................................................. 5
6. PLANNED AND PROSPECTIVE TOOL ADDITIONS .. 7
7. CONCLUSIONS AND FUTURE WORK ...................... 8
8. ACKNOWLEDGEMENTS ........................................ 8
9. REFERENCES .................................................... 8
10. BIOGRAPHY .................................................... 9

1. INTRODUCTION

The OpenOrbiter CubeSat Development Initiative (OOSDI) is working to build a small spacecraft system using open source software and open hardware principles. Some important design considerations for the CubeSat are nonfunctional requirements. The key contribution of this work is to present the design of a requirements engineering tool. This tool can be used to speed up the requirement elicitation and specification process for system-wide qualities such as availability, performance, and security. To this end, it uses quality attribute scenarios.
2. BACKGROUND

Space system developers have been, largely, reluctant to shift to modern engineering practices. This is evidenced by several studies conducted to understand the root cause of operational failures of space mission systems. For example, the report presenting the results of the investigation into the root cause of the failure of the Mars Climate Orbiter mission indicated that the main cause of the loss of the spacecraft could be attributed to an incorrect assumption made by staff at Lockheed Martin Astronautics and the Jet Propulsion Laboratory regarding measurement units used in their calculations [1]. One used English units whereas the other used metric units. This type of error is not unexpected given the difficulty of formally specifying and verifying the qualities and functionality of space systems.

CubeSats and the OpenOrbiter Program

CubeSats are small spacecraft that provide simplified launch vehicle integration capabilities due to their standardized form factor. CubeSats commonly range in size from the 1U form factor (with dimensions of approximately 10 cm x 10 cm x 10 cm and a mass of 1.33 kg) to larger sizes such as the 2U (20 cm x 10 cm x 10 cm) and 6U (30 cm x 20 cm x 10 cm) variants. Larger sizes have also been proposed. The CubeSat general requirements are discussed in [2, 3]. These requirements, however, tell little about the actual operations of a given satellite (excepting some basic compliance requirements). CubeSat system requirements describe what (services) the system should provide, and how well these services have to be delivered in order to meet technical, financial, educational and other goals. The system requirements, among other things, may include mission objectives, interface, functional and nonfunctional, and physical requirements. An example of a CubeSat mission statement could be to encourage and educate STEM students to design and implement a small satellite system for inexpensive access to space in a manner that will meet enumerated legal, financial, educational, and scientific objectives.

The OpenOrbiter CubeSat, which this work has been conducted in the context of, is a 1-U CubeSat with a very low parts cost [4]. The OpenOrbiter designs [5] have been developed from scratch to allow their wide dissemination and use by others that seek to develop a 1-U or larger CubeSat.

System Analysis / Non-Functional Requirement Specification

Functional analysis and other methods can then be used to refine and properly identify the functional requirements of the system under construction. For example, functional tree, use-case scenarios, stories, and prototyping approaches can be utilized to refine abstract requirements into more detailed requirements that can be verified.

The difficulty of capturing the system requirements, for the most part, revolves around modelling and analyzing of the non-functionalities (also known as system quality attributes or para-functionalities) of a system, such as safety, performance, and such. Contradiction, omission, or commission (addition) of non-functional requirements (NFRs) may impact the system’s architectural designs. These designs are key artifacts that are tied to the eventual success or failure of a system. NFRs are non-orthogonal. This means that some NFRs may interact with others in positive or negative ways. For example, maintainability and portability can impact each other in positive way, whereas security and usability may interact in negative way. Therefore, the selection of a design that meets the identified collection of NFRs requires trade-off analysis.

The omission or commission of NFRs may also result in a need for the reengineering of the entire system because, in some cases, no single component (or small subset of components) can be modified to fulfill the missing or added NFR. Ambiguity in describing NFRs may also pose significant issues both during the architectural design, and verification and validation (V&V) phases. The use of terms like ‘good’, ‘easy’ or ‘high’, which are often used to specify system-wide properties like security and usability is problematic, as it will be very difficult to specify and verify conformance during front-end and back-end engineering. For instance, a statement that the system shall provide a ‘very good user interface’ or ‘very good security’ are undefined and therefore cannot be architected or verified. Therefore, it makes sense to apply robust systematic requirement engineering techniques to describe and evaluate the NFRs of cyber-physical systems such as CubeSats.

Methods / Theories to Specify Non-Functional Requirements

In this subsection, prior work related to the methods and theories used to specify non-functional requirements is described. Goal-oriented methods such as KAOA [6] and the NFR framework [7] are qualitative approaches that have been proposed to systematically document system level properties (also known as global system properties). These approaches are, for the most part, based on a divide and conquer approach and are similar to the functional tree [8] approach used to analyze the functionalities of systems.

While functional tree [8] type approaches are useful to refine top level system functionalities, derive the low level functions, and document NFRs in a more systematic way, their usability and scalability can be an issue. This problem can occur when dealing with multiple NFRs or certain types of NFRs, such as software sustainability [9].

Jürjens, in [10-12], discussed the importance of security quality attributes in the architecture of security-critical systems. An extension of the UML standard, UMLsec, for the purpose of model design for secure system development projects was presented. A set of requirements for security engineering was provided; however, the mechanisms required for implementing security were omitted. A formal application of the UMLsec extension was presented. In this example it was used in a biometric authentication system.
A UML model with the UMLsec extension could prospectively provide useful insight on the current state of the security of a CubeSat project. The UMLsec extension could prove to be of great use for other projects using the Open Prototype for Educational Nanosats (OPEN) framework [13, 14] that are classified as security-critical systems.

In [15] [27], the authors proposed a method and supporting tool to select software architecture of a system using nonfunctional requirements. The tool implemented scenarios in order to identify NFRs. To this end, a set of tables (e.g., tactics) were applied to properly fill the gap between NFRs and its corresponding software architecture.

**Tool Support**

Work related to tools that can support the automation of aspects of requirements engineering are now discussed. Visure Requirements S.L launched the Visure quality analyzer, a tool that is described by the company as being able to assist in quality assessment and improvement [16]. It allows the developer to elicit, define, assess, improve and manage the quality of individual requirements and complete requirement specifications. Visure utilizes a user-customized “process-meta model” to compose a diagram of the process required to successfully generate project requirements. This diagram shows the relationship between requirements, tests, and other components of the design process. The diagram can be made accessible to the entire team working on a project. Visure also includes discussion board collaboration capabilities.

A second tool, inteGREAT, strives to “provide all stakeholders with a common view of requirements, leading to more accurate, consistent, and unified completion of projects over time” [17]. The software enables the definability of requirements in terms of multiple “dimensions”. It also aids in managing the traceability of requirements’ attribute history, the modeling of use cases, and allows the reusability of previously input data. The inteGREAT software is integrated with Microsoft Office products, allowing users to generate requirements and transfer them to the inteGREAT software.

In addition to the foregoing, other tools also exist. For example tools have been created by Bright Green Projects, Leap SE and PACE [18].

### 3. CUBE SAT SOFTWARE SYSTEM

The software for the CubeSat spacecraft discussed herein consists of the following components: the ground station, the onboard operating software (satellite / payload), and the communications link management software.

The ground station software provides the mission operators with an interface to define tasks, plan the mission and otherwise monitor the health and status of the spacecraft [19]. The ground station software will also be responsible for receiving and sending commands to the spacecraft over amateur radio bands (using an amateur, experimental or other license). The level of communications possible will not allow all images that the satellite captures to be transmitted to the ground. However, all images received from the satellite will be maintained by the ground station for future use.

The operating software is responsible for ensuring that the satellite is always operating within the parameters defined and that it performs the tasks assigned by the operators. This all has to be done as efficiently as possible to minimize power consumption and maximize task completion levels.

Onboard operating software (satellite and payload) utilizes input from multiple sensors such as a Global Positioning System (GPS) receiver, a temperature sensor, and an Inertial Measurement Unit (IMU). These devices will be utilized to provide the operating software with the information needed to make decisions to allow efficient operations.

The payload software will run on separate processing units and these will only be powered on when they are needed, to conserve the limited spacecraft power. The operating software is also responsible for breaking up high level tasks into smaller jobs that the operating system can perform while the payload computer is not powered on [19].

Among the capabilities of the payload software is onboard image processing. This software will serve several purposes. First, because of the limited bandwidth between the spacecraft and the ground station, an approach to reduce data needed to be transmitted is required. Second, it is desirable to produce ready-for-user data products that could be downloaded directly from the satellite by a prospective user. A technique called mosaicking, where several images are stitched together (effectively discarding overlapping regions) serves both of these purposes. An image covering a larger area is produced. Super resolution can also be used to create a higher resolution image from overlapping images. This will allow a less expensive camera with a lower level of resolution to be used, but still produce a high quality image. This approach frees valuable mass and volume for use by other subsystems.

### 4. THE NEED FOR THE DEFINITION OF USE CASES

The CubeSat’s operational capabilities have been defined using UML scenarios and use cases. This section describes the use of quality-attribute (QA) scenarios (based on the approach presented in [20]) to document the nonfunctional requirements (NFR) of CubeSats. This is a prerequisite for evaluation and further investigation.

**Need for Use Cases**

The rational for using quality-attribute scenarios is that it serves two related objectives of the CubeSat project:

1. to specify quality attribute requirements by incorporating a level of detail using templates to resolve
the ambiguity normally attributed with existing approaches (e.g., natural language), and

2) to validate the suitability of architectural design decisions using tradeoff analysis to ensure that the architecture meets the quality attribute requirements.

To meet the first objective, sixteen quality attribute scenarios were generated to specify key quality attributes associated with the design of a CubeSat. The CubeSat’s main design considerations are availability, security, performance and usability.

Availability requires that the system be up and running. Security refers to the degree at which system assets are protected. Performance refers to speed by which the system reacts to the end-user, performs its operations, or the extent to which it utilizes the resources in efficient way. Usability relates to the capability of the end user (of the satellite and data) to make use of the system or data.

To meet the second objective, these scenarios will be used for architecting a CubeSat system such that the selected architecture optimizes its key quality attributes (NFRs). The key justification for architecting for non-functionalities, as discussed in [21] [15], is that it provides a context under which design decisions and their consequences can be systematically analyzed as the system evolves through continuous integration and validation. The selected software architecture (ATAM [20]) will then be precisely defined using the AADL model-based engineering language [20, 22, 23] and analyzed using OSATE [24] to ensure that the software architecture supports the specified qualities.

Design Process

The most difficult part of engineering any problem can be, in many cases, to fully understand what the problem is. Next a solution must be created or selected to fully solve the identified problem. Of course, without a complete understanding of the problem and its context, the implemented system is unlikely to succeed. Before implementation can begin, a team of designers must make a number of decisions and formalize a set of software requirements based on stakeholder needs.

This process requires the designers to gathering information to formulate requirements (and is known as requirements elicitation) [25]. It is the job of the requirements engineer to gather general information from the stakeholder and, from this information, generate a set of formal requirements. All formal software requirements must satisfy (at least) two critical properties: specificity and verifiability. Therefore, every single software requirement must be written in a precise, quantifiable and testable way to minimize subjectivity.

The first step in this process, in the context of the CubeSat, is to capture system requirements (quality and capabilities) quality attribute and use case scenarios. To this end, two templates were used to specify both functional and non-functional requirements. Extended versions of use-case scenarios that incorporated references to non-functional or functional requirements were used to establish proper treatability among requirements. Quality attribute scenarios and templates developed by Bass, et al. [20] at Carnegie Mellon University were used to document the nonfunctional requirements of the CubeSat.

Quality Attribute Scenarios

The crosscutting requirements important to the operations of CubeSat are now presented using quality attribute scenarios based on [20]. The CubeSat mission will have a relatively short duration of roughly three months (based on the currently projected orbit). In order to make the most out of the given time, it is imperative that the different components of the system be highly available.

In general, availability refers to the percentage of time (or duration) the system is able to carry out its designed tasks, which can be represented as the ratio of the time the system is up and running over the total time the system is expected to perform its task, [20]. The following scenarios (based on templates proposed in [20]) specify availability scenarios specific to the CubeSat system. Scenario one utilizes a ping/echo fault detection tactic for availability characterization and is presented by Table 1.

<table>
<thead>
<tr>
<th>Table 1. Availability Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Portion of Scenario</strong></td>
</tr>
<tr>
<td>Source</td>
</tr>
<tr>
<td>Stimulus</td>
</tr>
<tr>
<td>Environment</td>
</tr>
<tr>
<td>Artifacts</td>
</tr>
<tr>
<td>Response</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Response Measure</td>
</tr>
</tbody>
</table>

This scenario specifies a situation in which a scheduled task needs to be carried out by the satellite’s payload software. The operating software sends a ping signal and waits for a response. If the response is provided, the task is then sent to the payload software to be carried out. If a ping response is not received, recovery steps may need to be performed so that the desired task can be carried out while the satellite is still in range of the task’s target. The fault should be logged in the operating software’s storage while at the same time the payload processing system should be restarted. During the restart any further attempts at sending tasks to the payload software should be halted. Once the payload software is believed to be online, a ping request should be sent to validate
that it is in an operational state. If a response is received, tasks should continue to be sent as normal, otherwise the recovery process should be continued until the payload software is able to be reached.

Scenario two, presented in Table 2, depicts a situation in which an exception/detection fault detection tactic is used. When the satellite’s onboard camera captures a target image, the operating software must validate that the data has been captured properly before it can be stored for later use by the payload software. If the data cannot be validated, either due to a camera malfunction during the capture or data write issue after image capture, a recovery of the task should be performed. The fault should be logged in the operating software’s storage and the camera and camera software should be restarted in order to attempt to resolve any problems so that future tasks are performed properly. Once the camera is detected to be operational, the failed task should be rescheduled to be put back in the imaging queue and carried out per its schedule priority.

Table 2. Second Availability Scenario

<table>
<thead>
<tr>
<th>Portion of Scenario</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Internal to system: Operating software.</td>
</tr>
<tr>
<td>Stimulus</td>
<td>Incorrect data from camera</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal Operation</td>
</tr>
<tr>
<td>Artifacts</td>
<td>Camera</td>
</tr>
</tbody>
</table>
| Response            | • Recover from camera fault:  
  • Log fault  
  • Restart camera  
  • Put the imaging job back in the queue |
| Response Measure    | Time to recover from camera fault. |

Scenario three is presented in Table 3. This scenario utilizes a sanity check fault detection tactic for availability. The CubeSat satellite is only able to receive transmissions from the ground station while it is in range of radio communications. This only occurs within certain time windows, as the satellite orbits the Earth. Once a transmission from the ground station is received by the satellite, it is stored in the operating software’s storage. A task to process the transmission is placed in the task queue, according to its priority with relation to the other tasks already in queue. Transmissions are marked with a sequence number; thus, if there is any break in the sequence numbers received during a transmission, a failure message is immediately sent to the ground station and the operating software should go back to a listening state to prepare for receiving the next message.

This scenario is one of the more important measures for availability as there is a limited window for communications between the ground station and satellite. Thus, as many tasks as possible need to be sent to the satellite for processing during this time.

Table 3. Third Availability Scenario

<table>
<thead>
<tr>
<th>Portion of Scenario</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>External to system: Ground Station.</td>
</tr>
<tr>
<td>Stimulus</td>
<td>Transmission of data interrupted</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal Operation</td>
</tr>
<tr>
<td>Artifacts</td>
<td>Operating Software</td>
</tr>
</tbody>
</table>
| Response            | Recover from failed transmission:  
  • Send “Failed transmission” message to ground station  
  • Wait for next transmission |
| Response Measure    | Time to recover from failed transmission and resume receiving of data. |

When working with a cyber-physical system such as a CubeSat, there must be contingencies in place to deal with the event of a hardware failure. Hardware systems must be monitored to ensure that critical conditions can be avoided or remedied.

The CubeSat battery is a prime example of such a system. If the battery loses charge, the system will shut-down and becomes nonoperational. To avoid this issue occurring, the charge capacity of the battery is constantly monitored and the system is placed in a low-energy state if the battery drops to critical levels. Table 4 presents this scenario.

Table 4. Battery Power

<table>
<thead>
<tr>
<th>Source</th>
<th>Battery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stimulus</td>
<td>Low Charge Remaining</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal Operations</td>
</tr>
<tr>
<td>Artifact</td>
<td>System Power</td>
</tr>
<tr>
<td>Response</td>
<td>Enter low energy state for recharging</td>
</tr>
<tr>
<td>Response Measure</td>
<td>Battery should never fall below 10% charge</td>
</tr>
</tbody>
</table>

4. QUALITY TOOL

The implementation of software, hardware, and the integration of the two in the OpenOrbiter CubeSat project is as critical as requirements identification. Requirements communicability, both of functional and nonfunctional requirements, plays a key role in the successful development of a system using model-based engineering approaches.

In order to facilitate effective communications among the numerous stakeholders, a prototype utilizing the concept of quality attribute scenarios was designed and implemented. These quality attribute scenarios are the equivalent of UML scenarios and use-cases. Each is built specifically to document a single nonfunctional requirement (and an appropriate test for it). Quality attribute scenarios include:

- an attribute category (e.g., safety),
• the source (component providing the data to be tested), the stimulus (component or situation that might set off the test in a real world setting),
• the artifacts (or the responding component),
• the response (the desired response from the system in this event),
• the response measure (e.g., length or time).

The response measure is the specific metric upon which the success or failure of the system is assessed. Examples of such metrics include a system’s time to recovery, the frequency of data communication, or the percent accuracy [26].

The prototype design and implementation is used to mitigate the limitations of the quality attribute scenarios previously implemented (discussed in [26]). The scenarios had to be manually inputted, one at a time. This mode of operation was not only time consuming, but also error prone. In addition, they were unable to be seamlessly integrated into a single, shareable, traceable, project requirement schematic. The nonfunctional requirements documentation application, when complete, will provide a single place to input all scenarios for all quality attribute categories. It will also facilitate collaboration on editing requirement details and test metrics. It will generate a project schematic that can then be used by all parties involved in the product development process. This rectifies the previously discussed issue with nonfunctional requirement ambiguity. It eases the struggle to agree on a set of requirements that are understood and implemented correctly by all parties involved.

Presentation of the Software Tool

The nonfunctional requirement documentation application prototype allows for the generation of a set of project specifications. It utilizes quality attribute scenarios (templates) to specify nonfunctional expectations from a product and testable metrics by which to ascertain whether these expectations have been met. The application utilizes a visual tree for creating this specification set, to aid in assessing the completeness of the project specifications.

Once the tree (and requirements) are completed, users can print a formatted list of the project specifications. Figure 1 depicts a dialog box with elements corresponding to the quality attributes scenarios and the tree view.

Once a project has been created, specific attributes scenarios can be added. Each of these attributes must fall under a specific category (see Figure 2). These categories are the general quality attributes decided upon by the project lead (such as availability, maintainability, etc.). At present, only a subset of categories is supported. These initial categories may be augmented, with others, at a later time. The currently included / supported categories are safety, security, availability, completeness, and fault tolerance. They were selected due to their pertinence to the CubeSat project and others like it.

Each category includes one or more attributes. These attributes will be based on the quality attribute scenarios, discussed earlier. Each will have a source, stimulus, environment, and artifact, response, and response measure, as inputted by the user (see Figure 4). Each attribute will have a specified identifier. This identifier will be composed of the attribute’s category identifier and its number within the category (i.e., the second safety attribute will have code “sft2”). This naming convention establishes the timeline of attribute creation within the project.

Figure 1: Quality attribute Scenarios Tool Design Features

Figure 2: Quality Attribute Categories
Attributes, projects, and categories may also be removed at any time. The user is promoted to confirm removal (see Figure 3). If any category holds a single attribute that is then removed, the category will also be removed from the visual tree and will not appear in the project specifications file unless added again.

Each attribute found on the tree may be validated before it keeps in the repository. This action will bring up the selected attribute’s information in the main editor panel. This may not be edited, but it may be viewed and removed at will.

Similarly, a “pending” vs. “committed” structure could be implemented. This would allow a project or team leader to accept an attribute (or category) before it is added to the project specifications. The attribute or category could still appear on the visual tree, but be noted as having a ‘pending’ status, until it is approved. This would allow more team members to contribute to the project, while allowing project leaders appropriate oversight and decision making control. This addition may make the system more cumbersome to use as it may require sifting through pending attributes when trying to perform other tasks.

Another possible adaptation would be to incorporate a step between creating and committing an attribute: an editing step. This step would allow multiple members of a team to input their own ideas of what test parameters might be best (for down-selection), rather than a single member having control over the test. Attribute identifiers were developed so that an ‘edit’ function could concatenate a version identifier to them. This nomenclature would preserve the timeline of features. This would, similarly, increase prospective team
participation, while potentially creating more work for users and team leads.

Finally, trade-off analysis could be incorporated into the system. This step would take place after the input of the categories and attributes. It would allow the completed attributes to be visually interconnected, demonstrating inter-correlation. This would enable users, for example, to find the negative correlation between performance and security and make decisions as to which to prioritize.

6. CONCLUSIONS AND FUTURE WORK

The development of CubeSats is still ongoing at the University of North Dakota. The research presented in this paper is a critical step in furthering the analysis, design, development and testing of CPSs, such as the CubeSat, using a modern software engineering approach. A key step for selecting and validating the optimal software architecture is the proper elicitation and precise documentation of system requirements. An important part of the system requirements is quality attributes is that they can be used to test the suitability of the design. The elicitation and specification of quality attributes is problematic and time consuming.

To this end, an assisted nonfunctional requirement prototype was developed and presented herein. The proposed prototype is not complete and has many limitations. New variations of the software are under consideration and new features may be added to increase its functionality and convenience, as described in Section 5.

ACKNOWLEDGEMENTS

A portion of this research was funded by the United States National Science Foundation (NSF Award #1359224) with support from the United States Department of Defense.

This paper draws from, revises and extends [26]. A subset of the work presented herein was also presented at the Research Experience for Undergraduates Poster Session at the 2015 AIAA/USU Conference on Small Satellites.

Small satellite development work at the University of North Dakota is or has been supported by the North Dakota Space Grant Consortium, North Dakota NASA EPSCoR, the University of North Dakota Faculty Research Seed Money Committee, North Dakota EPSCoR (NSF Grant # EPS-814442), the Department of Computer Science, the John D. Odegard School of Aerospace Sciences and the National Aeronautics and Space Administration.

The involvement of the numerous other students and faculty from multiple disciplines at the University of North Dakota in this project is gratefully acknowledged.

REFERENCES


[15] H. Reza, and E. Grant. Quality Oriented Software Architecture. In the proceeding of International


BIography

Hassan Reza is an Associate Professor of Software Engineering and Director of Graduate Studies in the School of Aerospace Sciences at the University of North Dakota, Grand Forks since 2000. He received his Ph.D. from the Department of Computer Science and Operations Research at the North Dakota State University, Fargo. His primary research interests include Software Architecture, Applied Petri nets, Model-based Engineering of Cyber-physical system, Cybersecurity, and Safety/Security Engineering. His research has been funded by the NSF, DOD, and Rockwell Collins. He is a member of the Association for Computing Machinery (ACM), and the Institute of Electrical and Electronics Engineers (IEEE).

Rashmi Sehgal is an undergraduate student in the Jack Baskin School of Engineering’s Department of Computer Science at the University of California Santa Cruz. She will be graduating in May 2017. During the summer of 2016, she successfully completed the Research Experience for Undergraduates (REU) in Small Spacecraft Software at the University of North Dakota.

Jeremy Straub is an Assistant Professor in the Department of Computer Science at the North Dakota State University. He holds a Ph.D. in Scientific Computing, an M.S. and an M.B.A. and two B.S degrees. He has published over 40 journal articles and over 120 full conference papers, in addition to making numerous other conference presentations. Straub’s research spans the gauntlet between technology, commercialization and technology policy. In particular, his research has recently focused on robotic command and control, aerospace command and 3D printing quality assurance. Straub is a member of Sigma Xi, the AAAS, the AIAA, SPIE and several other technical societies, he has also served as a track or session chair for numerous conferences. Jeremy’s work at UND has been featured in numerous media publications, coast-to-coast, including coverage in the Albuquerque Journal, Houston Chronicle, Washington Times, Oklahoman and San Francisco Chronicle. Prior to returning to academia, Jeremy had a successful career in industry where he held progressively responsible positions in software and technology development, technology management and management. He was responsible for the development of North America’s first traffic-adaptive
navigation solution and his work was featured in numerous media articles in publications including Entrepreneur, Forbes and the Silicon Valley Business Journal.

Nicholas Alexander is an undergraduate student in the Weisberg Division of Computer Science at Marshall University. During the summer of 2015, he successfully completed the Research Experience for Undergraduates in Small Spacecraft Software at the University of North Dakota.