Developing Effective K-5 Mathematics Educational Software

Theodor D Richardson, Jed S Lyons
University of South Carolina
Columbia, SC 29208
richa268@cse.sc.edu

Abstract

This paper presents a software engineering pilot study on the construction and use of educational software for the K-5 classroom environment. The goal of this study is to use the software engineering life cycle to guide the development of mathematics skills practice software with the intent to produce (1) a reusable template for producing meaningful and effective educational software as well as (2) a retrospective analysis tool to help guide future software development for better technology incorporation in the classroom. As part of the development process, the principal customer stakeholders, specifically administrators and teachers, are interviewed to assist in the gathering of requirements for the software as well as guide the choice of software architecture. For the purposes of presenting a complete evaluation of whether the resultant software is successful, a preliminary set of elementary classrooms is chosen as the beta testing group, spanning dramatically different demographics within a local school district. Student interaction with the software for each group is tracked and observed to assess the value of adding the software to the classroom and determining the effectiveness of the time devoted to the technology. This paper will present and discuss the complete decision process of creating the software as well as a thorough analysis of the success or failure in meeting the gathered requirements and evaluating the student testing. The information gathered in this study is used to create a reusable template for educational software engineering and evaluation that can be applied to software specific to the elementary classroom environment.

1. Introduction

A growing area of focus for teachers in all fields and at all educational levels is the incorporation of technology into classroom instruction. Though educational software is available, its use and incorporation into the classroom is not always guaranteed or successful. In experiences with various teachers in the classroom environment of K-12 education, it has been observed that the students and teachers could benefit from technology incorporation; those teachers willing to try proper software have experienced an increase in comfort with technology and an improvement of their students' skills understanding. The goal of this study was to apply the principals of software engineering to the creation of elementary school math software. A retrospective analysis is conducted to provide guidelines for making future software more meaningful and effective. The ideal result of this process would produce not only the software but a detailed analysis of the requirements necessary for other such software to be implemented and analyzed effectively.

The three main levels on which elementary classroom software must operate are as follows: the administrative level, the classroom level, and the student level. Administration is often

Copyright © 2005, American Society for Engineering Education

responsible for the software provided to schools, whether it is the principal of a single school or an official overseeing an entire school district. The requirements of this level are that the software fulfills the needs of the overall curriculum such that it will not be an extraneous use of classroom time. On the classroom level, the software has to fulfill the needs of the teacher not only with curriculum but as a means of tracking student progress and getting meaningful feedback akin to the more traditional use of hand grading. Teachers need to feel comfortable with the software as a teaching or reinforcement tool and feel confident in allowing their students to use valuable class time on such software. The students have very different needs from the software, such as reliability and ease of use. Each of these levels represent a set of stakeholders, or those with a controlling interest in the software; as such, each will be discussed in further detail later in this paper along with the approach used to meet the resultant requirements by the software written.

It is likely that most educational software undergoes the traditional software life cycle of (i) requirements analysis, (ii) design, (iii) construction, (iv) testing, (v) installation, (vi) operation, (vii) maintenance, and (viii) retirement[1]. However, it is not always certain that the product is tested by and for the correct stakeholders in the educational system. The software herein is an update of a product called Math with Montague (MM); it was initially a senior project on record at Bethany College in West Virginia. The original software was intended for use as skills practice software based on the West Virginia state standards for mathematics. It enjoyed limited success when presented at the West Virginia Council of Teachers of Mathematics annual conference in 2002, but it was evident that the software was not able to fulfill the needs of the correct stakeholders. The children present reacted very well to the software and later teacher reports indicated a high level of engagement from their students. The teachers, however, felt it lacked a means of tracking performance to make it a useful classroom tool. Similarly, administrators decreed that the software was not enough to fulfill the West Virginia Instructional Goals and Objectives (IGOs) for which it was written. There were other issues discussed as well that will be mentioned in the individual stakeholder sections.

Based on this constructive criticism, the software has been redesigned according to the software engineering procedure detailed herein and used to construct a requirements template for all educational software geared to the K-5 elementary grade levels. To create a universal set of guidelines for this type of software, each group of stakeholders was revisited within several school districts of significantly different student demographics. The new software product that results from this study is entitled Math with Montague Online (MMO) and can be found at http://www.equinoxempires.com/k12/montague.htm. As part of the continued study of requirements analysis and software engineering, the resulting product is also scrutinized for the places in which it falls short of the fulfillment of certain requirements. Sometimes in software engineering, certain requirements conflict with each other. This usually results in a tradeoff where the needs of one requirement must be slightly sacrificed for the sake of a more important requirement; a comprehensive analysis of the tradeoffs is also presented in the section on retrospective analysis. Additionally, all of the reusable requirements and analysis methods used herein are discussed in detail in the section on development and analysis tools.

The software was developed to meet the guidelines of the South Carolina Mathematics Curriculum Standards[3] for the express purpose of increasing students' enthusiasm and

Copyright © 2005, American Society for Engineering Education

understanding in the realm of mathematics. This represents the strategy for curriculum relevance for the purposes of distribution within South Carolina. In the microcosm study of this software, this represented a comprehensive approach to providing this quality attribute; however, for the purposes of national reuse, this would have to be modified to encompass the similar state standards across each of the states in the nation. To solve the need for engaging content, a dynamic environment was created involving talking insects such as the title character, a praying mantis named Montague; every effort was made to ensure that the characters would still be interesting to the students in the older grade levels as well. Screen shots of the final MMO software product can be seen in Figure 1 below. This paper discusses the development cycle, analyzes its effectiveness, and suggests a set of best practices called the Educational Software Development and Analysis Toolkit (ESDAT).

Figure 1. Screen captures of the Math with Montague Online software demonstrating the software environment (addressing the engaging content quality attribute of the system) and various problem types: (a) counting with the complete browser environment for student users, (b) addition, (c) counting money, (d) multiplication, and (e) long division.

2. Method

The methodology followed herein is similar to a case study in software engineering which follows the software life cycle mentioned above. Since the software was not released on a large scale, the process does not include the maintenance and retirement phases. However, the retrospective analysis was conducted as a post-mortem on the MMO software release 1.0. This would then lead to maintenance and another installation. Each of the stakeholders needs is analyzed individually with a discussion of how MMO met or did not meet the requirements in release 1.0. In most cases, the requirements are not specific to MMO and can be used again by any educational software developer; the reusability of the requirements analysis was one of the primary goals of this study to provide higher standards for classroom integrated software and technology.

Copyright © 2005, American Society for Engineering Education

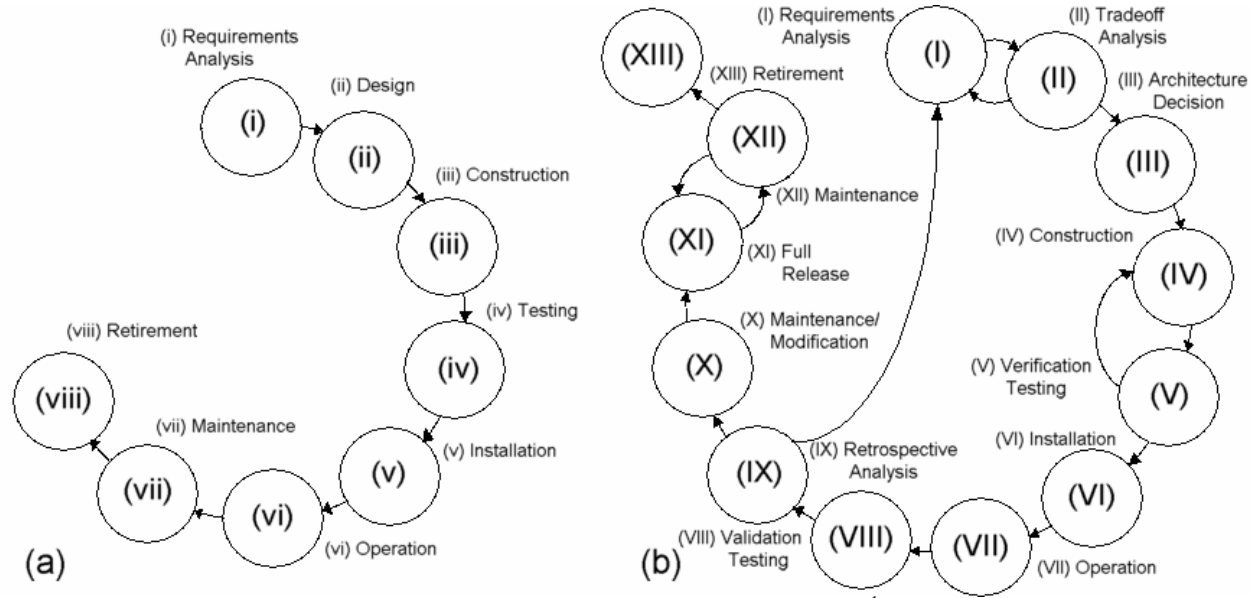

Figure 2. Illustration of the software life cycle: (a) traditional software life cycle[1] and (b) modified software life cycle for educational software development devised herein.

A Volere shell[2] is typically used for specific atomic requirements analysis, step (i) of the software life cycle. The Volere shell was designed for large projects with multiple contributors and stakeholders; it gives exacting stipulations to prevent overbuilding the software through forced accountability of the necessity of each requirement. The atomic requirements are those that cannot be broken down any further and represent a functionality or constraint upon the system. In general terms, this may be too strict for reuse, but some examples of a new educational software requirements shell are provided herein for the MMO software, such as the one in Figure 3 below.

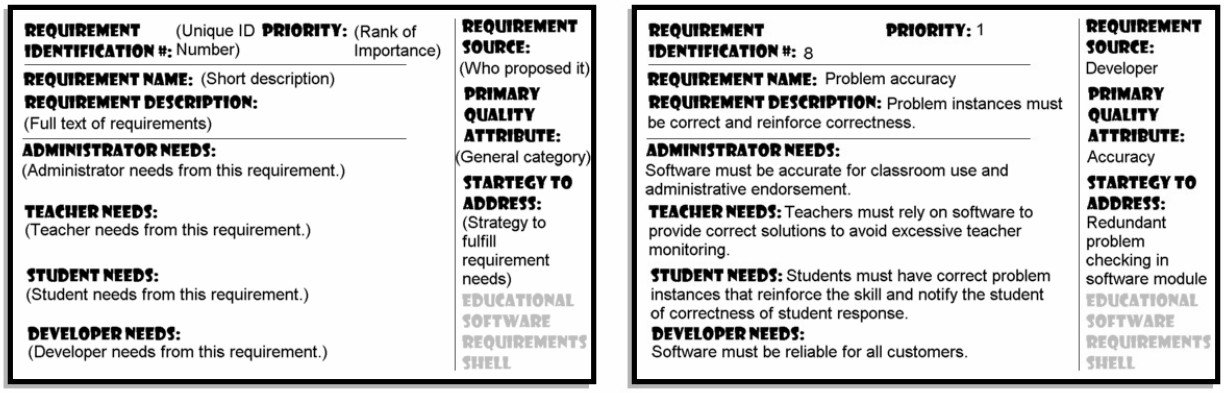

Figure 3. Educational software requirement shell: (a) blank template with descriptions, and (b) specific instance for MMO accuracy requirement.

After gathering the requirements, the next phase is usually (ii) design. However, a significant amount of the design can take place within the requirements gathering phase and the two often overlap. This phase is largely bundled with the requirements analysis in this type of software because the general sketch of the software is often known before requirements gathering begins. In terms of MMO, the software had already been determined as mathematics skills practice

Copyright © 2005, American Society for Engineering Education

software for the K-5 elementary grade levels; this information dictates certain requirements such as the one seen in Figure 3. This phase includes choosing an underlying architecture for the system, a significant endeavor that is demonstrated here by a change in the underlying architecture from MM version 2.0 to MMO version 1.0.

Construction, phase (iii) of the software life cycle, is the actual production of the software. This involves the setup of the hardware and any off-the-shelf software used in the system as well as the coding of new software. Most of the requirements decisions and definitely the architecture must be completed by this phase since it will be almost impossible to change the makeup of an existing system after this point. In terms of the software engineering and analysis, this is the phase that takes a significant amount of product specific work and is therefore largely overlooked in this case study. MMO represents a software product that actually did change architecture and distribution, which is worth mentioning in the retrospective analysis. The specific coding details, however, are irrelevant to the point of this work.

The testing phase (iv) as determined by the software engineering life cycle is largely considered to be an internal process before the product is released to the customer for consumption. However, this is not the ideal situation when dealing with educational software where the customer is actually three sets of stakeholders; this is especially true when dealing with students who are not at the same developmental level. Usability is a strong requirement with the student stakeholders, so it is ideal to allow them to be part of the testing procedure. For this reason, the installation and operation should coincide with later testing procedures to make certain that the software being released can be effectively and meaningfully used by students. Testing should consist of two main procedures: verification and validation. Verification is the determination of whether the product is being built correctly; this is the part of the testing that should be done before the product is viewed by customers. Verification assures that the product functions as promised on the platforms promised. Validation is the testing of whether the correct product is being built; this is the testing that should be conducted by the end users of the system, namely teachers and students. It was this validation testing that was the focus of this study and the motivation of this software construction. To that end, the installation, phase (v), and operation, phase (vi), coincide with the testing of the software in a small scale release. By combining these three stages, the product stands a greater chance of success in the market and will require less maintenance once released on a large scale. This is a departure from typical views of software engineering[1] in the omission of a small scale test release, something that should be very strongly considered with educational software.

It is after the results of this test release that the retrospective analysis was conducted to process the collected data and determine what questions need to be addressed by elementary education software as a whole. The weaknesses and strengths of the software were then scrutinized to see where MMO could be improved as well as determine the weakness in current practices of educational software as a whole. This retrospective analysis and subsequent findings on areas of potential improvement were also key motivations of this work. A template for future elementary classroom software engineering and analysis is provided in Section 4. It should be noted here that all respondents in both the interviews and the classroom studies will not be named within this paper for the purpose of preserving privacy; only school districts and locations will be named.

Copyright © 2005, American Society for Engineering Education

3.1 Administrative Evaluation

The administrative requirements of educational software are sometimes the only ones considered because administrators tend to be the largest customers for educational software. In various experiences within classroom environments in Richland and Lexington counties in South Carolina, it has been observed that the largest technology component the students have is what is prescribed by the district for the teachers and students to use. Sadly, it has also been seen that a large component of this software bundle was not developed for student needs, especially in terms of lower grade levels where literacy is a paramount issue. This means that the software is in use but remains ineffective for its intended purpose of helping students learn or practice a concept. It can be deduced, therefore, that the software was designed to be sold to administrators rather than used by children. Administrative requirements may carry the most weight in an educational software system; however, administrators should never be treated as the only stakeholders in such software.

The needs of the administrative stakeholders are still very pertinent to the software being used. It is often the administration that is responsible for access to the software via a network or a series of computers. Administrators are also accountable for the curriculum being taught to the fullest extent possible, a curriculum usually dictated by statewide standards. Any software that does not avail itself to ease of distribution or upholding the curriculum standards is likely to be overlooked by administrative customers. To address the needs of the administration, the math and technology consultants in the Richland County School District One Title One office as well as several principals throughout the district were kind enough to share their needs for an educational software system like MMO. Without determining specific requirements, they helped formulate a list of general quality attributes that are essential to their choice of district or school software. Though the interviews were conducted separately, the main attributes were echoed by all parties; though these were listed as priorities, it should be noted that most of the respondents agreed that the software they had personally selected did not meet all of these criteria. The prioritized list of these quality attributes and their descriptions follow:

1. *Curriculum Relevance* – This requirement means that any software used within the school system must be directly related to the state standards and curriculum dictated by the school in which the system will operate; the general consensus was that the more standards that can be wholly fulfilled by the software system, the likelier it will be that the software will be adopted for use.
2. *Accessibility* – This refers to the ability of students and teachers to navigate and use the software easily; this includes the ability of disabled persons to use the software with minimal to no assistance. In the case of educational software, it can also refer to the ability of students who are not yet literate to use the software without aid.
3. *Maintainability* – In the case of classroom software, this attribute refers to the goal of minimizing the need for the local administrator of the software to update it or fix it when an error occurs.
4. *Reliability* – A reliable system is one that can handle the kind of usage it would receive from a class, school, or district with minimal interruption and/or data loss.

Copyright © 2005, American Society for Engineering Education

5. *Availability* – The main requirement of availability is that the software be accessible at all times with minimal interruption of service even if a failure occurs.
6. *Software Residence* – Software residence refers to the physical location of the software. The ideal software residence is on an internet or intranet where it can be accessed at any time from any computer within the network. In the case of the software that resides on an intranet or even on individual computers, this attribute is coupled with maintainability in the goal to minimize the administrative interaction with the software while maximizing its availability.

Gathering requirements is usually an extensive process taking place throughout the planning stages of software engineering. Most of the requirements should be finalized before actual development begins to avoid the difficulty of later overhauls. The administrative stakeholders often have the least specific requirements for a software system and many of them are interrelated. However, they do, along with the teachers, determine the software architecture and functional requirements of the system. For the purpose of this study, teachers who wish to use the software only for their class share the role of administrator and teacher stakeholders since it then becomes their responsibility to maintain the software within their classrooms. An example of the software requirements for the MMO system gathered from the administrators would be to meet one of the mathematics standards for the state of South Carolina, such as that shown in Figure 4.

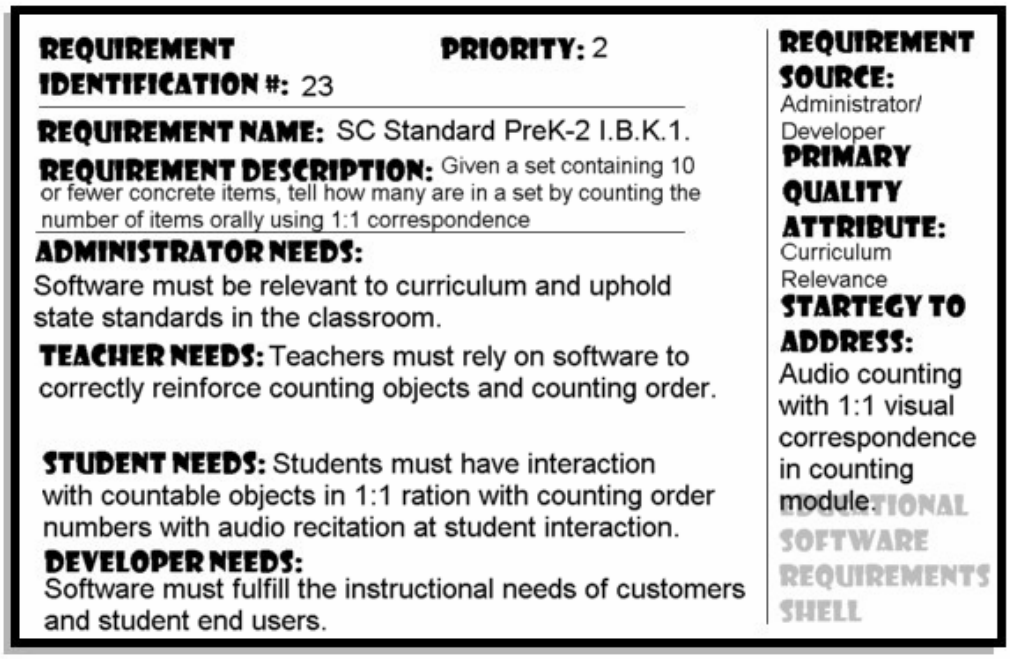

Figure 4. Software requirement for Math with Montague Online (MMO) using the educational software requirement shell. The specific requirement used is the South Carolina state standard for PreK-2: Number and Operations Standard I Expectation B.K.1: "Given a set containing 10 or fewer concrete items, tell how many are in a set by counting the number of items orally using 1:1 correspondence"[3].

The software architecture is one of the most important decisions of creating a new system. This is the underlying framework of how the software and/or hardware should be constructed.

Copyright © 2005, American Society for Engineering Education

Innumerable types of software architectures exist and have been extensively studied[4.7]. Each of these types has a specific reasoning and set of applications; strategies exist to help these architectures meet other requirements and facilitate quality attributes. The software architecture chosen for MMO is the client-server architecture. Client-server architecture is a set of interconnected computers containing separate software modules that communicate via network connections; this is a system of components and connectors in which multiple clients interact with a single server that contains the permanent data records[4]. In the case of MMO, a central online PERL script was used to control access to the various software modules and record the data necessary to track user interaction by remote procedure calls embedded in the individual software modules, which closely follows the underlying reason for client-server systems.

Architectures can be specified and/or clarified by constructing views into them, such as Philippe Kruchten's set of five views including the logical view, development view, process view, physical view, and a unifying set of scenarios[5]. Kruchten's method was used to construct the architecture for MMO and determine that the optimal architecture type for the system is the client-server web-based system. The logical view of the architecture is composed of classes used to store information and call functions; the view simply describes the interaction of the classes via functions and label subclass relationships. The development view labels processes and the interaction of processes by messages and procedure calls. The process view consists of modules and subsystems with a set of dependencies among each other where interaction occurs. The physical view describes the hardware and the location of the software components and their network interactions. Scenarios describe real world uses of the system; these unify the different views by describing the interaction of the components within each view. An example of these views for MMO can be found in the figure below.

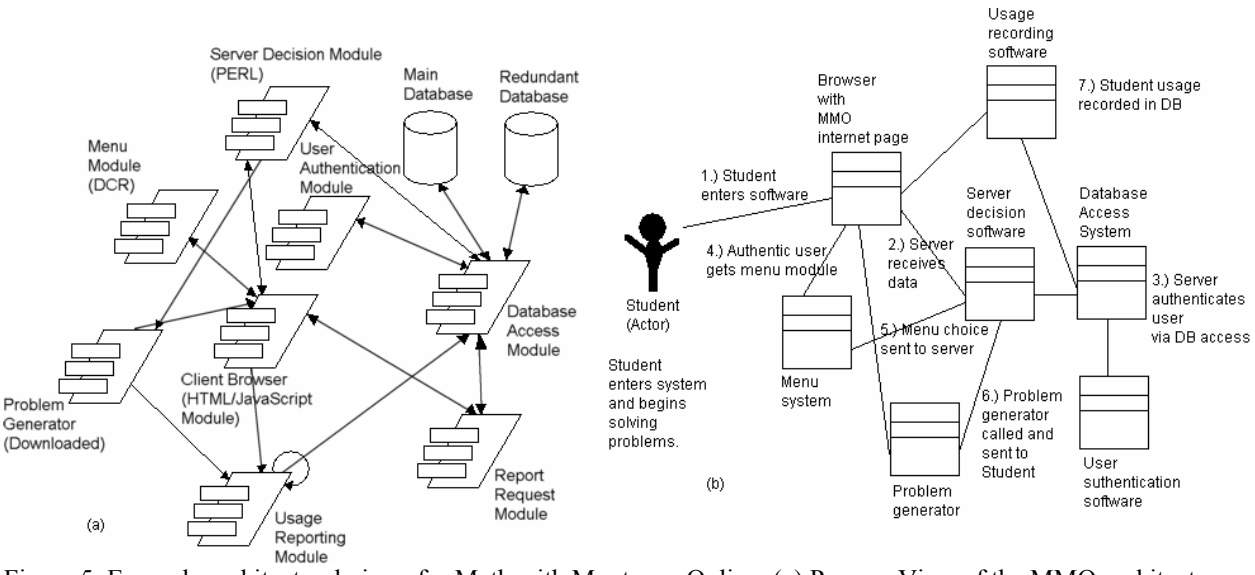

Figure 5. Example architectural views for Math with Montague Online: (a) Process View of the MMO architecture and (b) scenario of "Student enters software and begins solving [arbitrary] problems" with ordered component interactions (numbered 1 through 7).

The administrator stakeholders needs have been addressed in MMO by the use of several architectural strategies. The curriculum relevance attribute has been addressed by making each of the addressed state standards into requirements as shown in Figure 4 above. Accessibility is a

Copyright © 2005, American Society for Engineering Education

more difficult attribute to address because the users who must be able to navigate the system are the students; therefore, the accessibility will be addressed as part of the student requirements analysis and testing. Software residence has been solved by the client-server web-based architecture; this also solves the problem of availability. The one issue that could not be successfully addressed with the available hardware is reliability; MMO is located on server space that is leased with a bandwidth restriction and it is therefore incapable of dealing with school-wide volume at present.

3.2 Teacher Evaluation

The original Math with Montague, like many other educational software systems, was found to be highly engaging to students but utterly unusable for teacher analysis. To prevent this from becoming the fate of the new MMO software, several teachers were interviewed during the requirements gathering phase as well as after the initial verification testing was completed such that they could see the actual product as they gave any additional requirements necessary for their use of the system. These interviews were rewarding in terms of requirements gathering and analysis at both times within the software development phase, stages (i – v) of the software life cycle.

The first set of interviews was conducted to gather general requirements for the software while the second set of interviews was used to improve the existing software for classroom integration. The first set of interviews included teachers from Richland County School Districts One and Two as well as two teachers from Ohio County in West Virginia who were familiar with the original incarnation of Math with Montague. The requirements stressed by the teachers varied along the lines of grade level differences. Chief among the requirements for the lower grade levels K-1 were literacy concerns and the requirement that the software have an audio component to assist the children who could not read words. Conversely, the requirements enumerated by the teachers of the higher grade levels in 4-5 were primarily the difficulty of the problems and the ability for the software to self-monitor the students and provide varying levels of challenge to meet the needs of students with different skill levels. Ranked by emphasis and repetition below are two sets of quality attributes desired for the software, separated into younger and older elementary grade levels.

Younger Grades (K-3)

1. *Accuracy* – The primary requirement for educational software is that the problems given have verifiably correct solutions; similarly, students must be correctly notified of correct and incorrect responses.
2. *Usability/Accessibility* – The students must be able to successfully access the software at any time during the school day and navigate the different skills without significant assistance, at least once they have previously been exposed to the software. This is further addressed in the student requirement analysis.
3. *Curriculum Relevance* – Similar to the administrative requirements, teachers cannot afford to use valuable class time on software that does not support the skills of the curriculum in more than a tangential manner.

Copyright © 2005, American Society for Engineering Education

4. *Performance* – Due primarily to limited classroom computer time and short attention spans, performance speed is a concern for elementary software, especially in the K-3 grades.
5. *Engaging Content* – Again due to limited attention spans, the content and environment of the software must be engaging to the lower level students to assure that it holds their interest and encourages them to continue solving the problems.
6. *Reliability* – This is the requirement that the software maintain its data and integrity even if a failure should occur such that students will not be given incorrect problems and teachers can retain any tracking information even if the system crashes.

Older Grades (4-5)

1. *Accuracy* (see above)
2. *Curriculum Relevance* (see above)
3. *Varying Difficulty/User Independence* – The students operate at different levels of understanding within any class; the software must therefore accommodate students who are advanced as well as those who are slower to comprehend the concepts presented. An overlapping quality attribute is user independence, which can refer to the ability of the student to pace themselves in terms of difficulty or the ability of the software to adjust difficulty based on student responses.
4. *Usability/Accessibility* (see above)
5. *Progress Tracking/Benchmarking* – The teachers are interested in software that can track the performance of students within their classes; the ideal information provided by the software would meet or exceed what the teachers can gain from hand grading student papers. Benchmarking is the ability of the software to diagnose the level of understanding for each student and track their progress in terms of improvement or decline in skill level.
6. *User Anonymity/Security* – This attribute mainly applies to the MMO system because it is available online at all times. Student information is highly sensitive and should not be accessible in any form over the internet; the actual request made by the teachers for MMO was that the students not provide any information except a class number unrelated to any personal or identifiable information whatsoever except by the teacher within the classroom.

The MMO software was designed and constructed based on the previous incarnation of the software and the requirements set forth by the administrators and teachers in the first set of interviews. The architecture was chosen as a client-server web-based architecture which lends itself to availability and modifiability. This is a departure from the initial CD-based single software package distribution that was found to be ineffective for organic classroom integration. Requirements analysis was also used for the teacher stakeholders as seen in the administrative section Figure 4.

MMO addresses the quality attributes desired by the teachers in several ways. Accuracy was listed as the primary attribute for both the younger and older grade levels; this was assured within each module by generating a mathematics problem instance and then solving the problem internally to make sure the solution is valid and adheres to the prescribed skill level. The modular design is the strategy used to control the difficulty level as well as support the

Copyright © 2005, American Society for Engineering Education

development need of modifiability; individual modules with specific degrees of difficulty are transmitted to the client's browser via the internet. In this way, a student can select a difficulty level from the menu and receive a software module that will construct problems geared to that level; this process works independently of the other users, allowing each student within the class to select his or her own appropriate level of difficulty. It should be noted, however, that this system does not completely fulfill the requirement of self-monitoring. In the ideal case, this would mean that the software adapts internally based on the success or failure of the student's responses to the questions; this is an example of a tradeoff where this feature was sacrificed for the increased speed of loading smaller problem generation modules.

Progress tracking was accomplished by remote procedure calls within the modules that track the user interaction with the software and call back to the server PERL script to record the data to a central database; this data can then be accessed by the teacher using a secure password. The students are able to enter the system anonymously using a class number; this differentiation of the individual students using the system is required for accurate data tracking and reporting to the teacher. The anonymous student login and the encryption on the database and password access to the system comprise the strategy for addressing the security concerns of the teacher. Reliability, as mentioned above, could not be completely addressed; however, the data is copied to both the main database and a redundant database. If a failure of the main database occurs, the redundant database can be used to preserve the student activity for teacher performance tracking. Additionally, corruption of the PERL script prevents access to the software, ensuring that students do not receive incorrect problem instances.

The usability and engaging content are interrelated in terms of MMO. Usability is addressed by forcing the client browser modules to read each word on the screen to the students; for higher grade levels, this option can be disabled by the user. In the second set of teacher interviews, it was also found that the menu needs to be read to children who are not yet literate. To prevent the menu from being read whenever the software is loaded, the compromise was to cause an audio response whenever the student rolls the mouse over a word on the menu to simplify use. The engaging content was required to maintain student interest. Having experience with the results of the first release of the software aided the development of this solution; the students responded positively to the vibrant colors and animations that make the characters in the software seem to be alive in constant randomized organic motion.

3.3 Student Testing

As previously mentioned, administrators and teachers serve as the main customers of this kind of software, but they are not the end users. In the unique case of educational software, the end users are students who often do not select the software themselves but are expected to use it. For the software to meaningfully impact the students and fulfill the goal of effectively teaching and/or reinforcing a topic, real learning must occur in an environment that fosters interest and comfort. MMO was designed to meet these needs by adapting the requirements gathered from the administrators and teachers to suit the unarticulated requirements of the students using the software. Primary emphasis was given to the quality attributes of usability, speed, and engaging, developmentally appropriate content.

Copyright © 2005, American Society for Engineering Education

One important idea came from the second set of teacher interviews with the mostly completed product, the idea that the software should be tested in an organic integration into the classroom environment. Typically, testing occurs in a developer prescribed manner for certain durations of time per day or week. However, once adopted, the software will be used on a very different schedule than what the developer prescribes for testing. It is therefore much more relevant to test the software under the normal classroom usage. This usage unfortunately varies from class to class, making uniform evaluation and comparison more taxing. The duration of the testing was two weeks, but it is highly advisable to prolong the student testing for better analysis of skills improvement. It was found in the case of MMO, however, that this organic testing was far more informative as far as student interest and actual usage.

A beta group of four classrooms volunteered to test the software through organic classroom usage; a larger group would be advisable, especially having multiple classrooms within the same grade level for comparison. However, with the school year underway when the request was made, most of the teachers could not spare the necessary class time before the date required by the study. Since this is a pilot study of the software and the engineering process used to construct it, the four classes provided significant data to process and evaluate the effectiveness of the use of MMO. The beta group consisted of a first grade class, a third grade class, a fourth grade class, and a fifth grade class from two of South Carolina's Richland County District One schools where the schools had strikingly different demographics and poverty statistics. Surprisingly, the demographics of the four classrooms did not seem to have a large effect on the results of the software incorporation. Student age was the most important factor determining software response in terms of enthusiasm. The data recorded by the software in the main database is a list of actions taken by the user such as requesting a software module, the difficulty selected, the number of problems answered, the number of problems answered correctly, the total time spent on the page, and the average time spent per problem. The results of the beta classroom testing can be found in the figure below.

Copyright © 2005, American Society for Engineering Education

	Grade 1 Week 1	Grade 1 Week 2	Grade 3 Week 1	Grade 3 Week 2	Grade 4 Week 1	Grade 4 Week 2	Grade 5 Week 1	Grade 5 Week 2
Total Number of Student Respondents per Week	12	14	18	18	16	3	21	21
Total Class Time Spent per Week (Internal Estimate)	1h 32m	2h 10m	54m	1h 10m	30m	10m	1h 15m	1h 17m
Average Amount of Time Spent per Student	22m	27m	53m	1h 8m	10m	9m	1h 12m	1h 14m
Skill	-	-	-	-	-	-	-	-
Addition	-	-	-	-	-	-	-	-
Single Digit w/o Remainder	-	-	-	-	-	-	-	-
Student Respondents	12	14	-	-	-	-	-	-
Average Correctness (%)	47	49	-	-	-	-	-	-
Average Response Time per Problem (s)	23	24	-	-	-	-	-	-
Single Digit w/ Remainder	-	-	-	-	-	-	-	-
Student Respondents	12	13	-	-	-	-	-	-
Average Correctness (%)	39	44	-	-	-	-	-	-
Average Response Time per Problem (s)	27	29	-	-	-	-	-	-
Double Digit w/o Remainder	-	-	-	-	-	-	-	-
Student Respondents	5	6	-	-	-	-		-
Average Correctness (%)	67	59	-	-	-	-		-
Average Response Time per Problem (s)	34	33	-	-	-	-		-
Money	-	-	-	-	-	-	-	-
Count Money	-	-	-	-	-	-	-	-
Student Respondents	6	8	18	18	-	-	-	-
Average Correctness (%)	17	16	68	84	-	-	-	-
Average Response Time per Problem (s)	73	69	54	57	-	-	-	-
Add Money	-	-	-	-	-	-	-	-
Student Respondents	3	2	16	17	-	-	-	-
Average Correctness (%)	25	30	75	78	-	-	-	-
Average Response Time per Problem (s)	86	88	63	64	-	-	-	-
Subtract Money	-	-	-	-	-	-	-	-
Student Respondents	-	-	7	7	-	1	-	-
Average Correctness (%)	-	-	75	95	-	50	-	-
Average Response Time per Problem (s)	-	-	68	76	-	87	-	-
Multiplication	-	-	-	-	-	-	-	-
Single Digit	-	-	-	-	-	-	-	-
Student Respondents	-	-	10	9	16	3	-	-
Average Correctness (%)	-	-	58	64	15	76	-	-
Average Response Time per Problem (s)	-	-	27	32	93	74	-	-
Double Digit	-	-	-	-	-	-	-	-
Student Respondents	-	-	3	3	7	2	21	21
Average Correctness (%)	-	-	94	94	10	68	86	88
Average Response Time per Problem (s)	-	-	28	27	112	74	64	56
Division	-	-	-	-	-	-	-	-
Single Digit w/o Remainder	-	-	-	-	-	-	-	-
Student Respondents	-	-	-	-	-	1	21	20
Average Correctness (%)	-	-	-	-	-	0	89	93
Average Response Time per Problem (s)	-	-	-	-	-	65	57	45
Single Digit w/ Remainder	-	-	-	-	-	-	-	-
Student Respondents	-	-	-	-	-	-	20	20
Average Correctness (%)	-	-	-	-	-	-	79	83
Average Response Time per Problem (s)	-	-	-	-	-	-	63	52
Double Digit w/o Remainder	-	-	-	-	-	-	-	-
Student Respondents	-	-	-	-	-	-	12	9
Average Correctness (%)	-	-	-	-	-	-	23	26
Average Response Time per Problem (s)	-	-	-	-	-	-	123	112

Copyright © 2005, American Society for Engineering Education

Figure 6. Abbreviated beta classroom testing data results: the most used modules for each grade level are shown along with the tracking data recorded by the MMO software. Individual results varied by student.

The teachers were also asked to complete an observation form each time the class used the software documenting the estimated level of involvement and enthusiasm of their students. They were also asked to interview a random sampling of their students for feedback on the MMO software as well as add comments of their own. The data reported on the observation forms can be found below. Only three of the four classes returned observation sheets for the study.

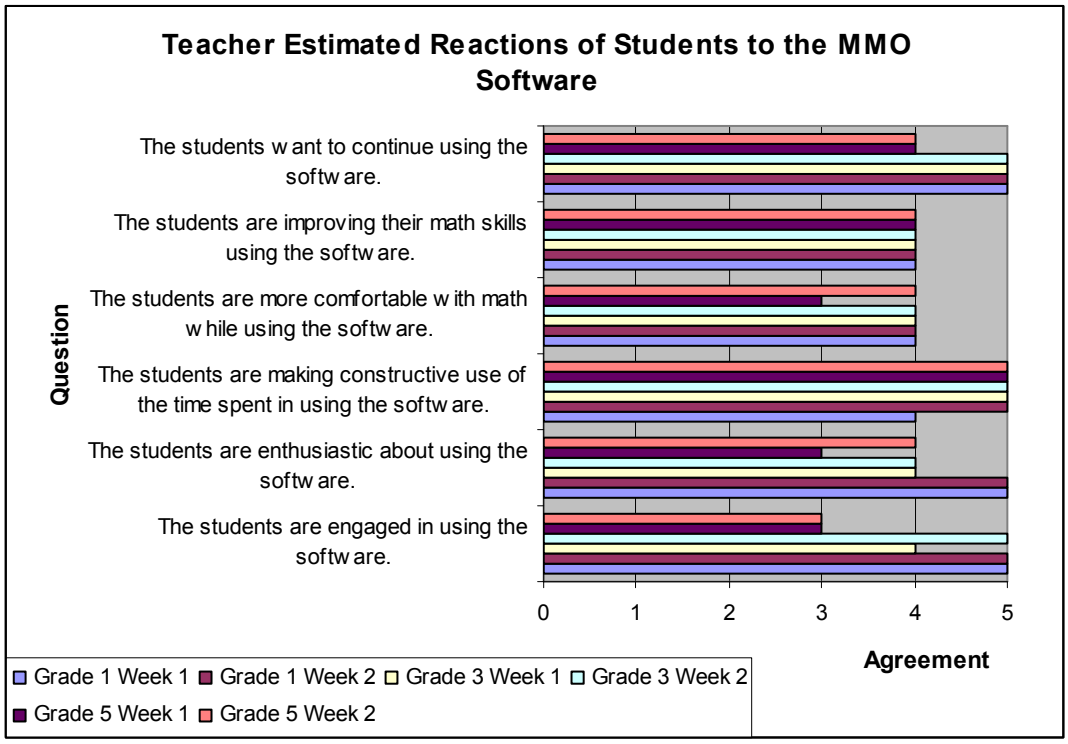

Figure 7. Classroom observation sheet results as reported by the classroom teachers; higher agreement numbers indicate stronger agreement (0 - strongly disagree, 5 – strongly agree).

Below are some of the student and teacher comments listed on the observation form completed by the classroom teachers. These comments were used as part of the overall software assessment and reflect both the positive and negative aspects of the software and its use. The fourth grade teacher returned only a single email response to the software instead of the standardized observation sheets.

Grade 1 Teacher Comments:

- "The software is very engaging. The vivid colors and animation make it an environment that the students really want to explore. They like the ability to turn the music on and off since some students work better with noise and others work better in quiet."
- "Some of my students had trouble logging on to the software when they first started. Once I showed them what to do, they seemed to be able to reproduce the necessary navigation to get to the addition skill we are practicing in the classroom. Most of them only forgot to logout before leaving the computer."

Copyright © 2005, American Society for Engineering Education

Grade 1 Student Comments:

- "I like the bugs a lot. The math is easier with the bugs showing you what to do."
- "It was kind of hard to … get around. The adding was fun when I got there."

Grade 3 Teacher Comments:

- "My students are simply ecstatic about using this software. It was used as a reward for finishing lessons, and they all tried to get on the computers for it. Maybe it was because it was something new or they knew they were the first ones to use it, but they really enjoyed it."
- "The children are still using it happily and even making a competition out of who can get the most problems done and get the tokens. I have two girls who are tied at fifteen tokens each." The tokens are a built-in reward system for correct problem solutions; the tokens can then be used at the MMO arcade to play games like tic-tac-toe.

Grade 3 Student Comments:

- "I really like Math with Montague. It makes me think that bugs aren't so bad. I like the way it lets me fix mistakes on the problems and stuff."
- "I like trying the harder level of the skills like trying the hardest division. With the honeycombs, I see what I need to do so I can do the problems." The honeycombs are the visual aid for the multiplication and division as seen in the screen captures above.

Grade 4 Teacher Comments:

- "The students grew impatient with the slow loading of the software and most of them gave up using it after the first skill. I decided this was not an effective use of class time but allowed those interested to continue using it if they finished more relevant work. I think they were expecting more and were disappointed."

Grade 5 Teacher Comments:

- "The children are really blessed to have this opportunity and someone who values their opinion on this kind of project. They do really well at the problems and they prefer using the software to a worksheet for practice, so I may just let them do a certain number of problems on the computer. I will know if they did them because of the tracking put into it. This is a well constructed piece of technology, something that we don't see a lot of."
- "I think the only suggestion I may have for improving the product is to let it automatically direct the difficulty of the problems the students are given. They may also want a more adult environment for the higher grades instead of the bugs, even though I think it's cute."

Grade 5 Student Comments:

- "I think the multiplication and division problems are hard enough and I like choosing how hard they will be. I just don't know about some of the lower ones like counting and adding. It just seems like we don't need that anymore."
- "I think it's pretty neat that we get to use the computer for math class and I like how the program helps you out with what you need to do to get the right answer."
- "I don't know about all those bugs and stuff, but I like getting tokens for the arcade."

Copyright © 2005, American Society for Engineering Education

- "It helps you think to have something you can see and interact with on the screen. It's almost like its trying to help you learn."

The teacher comments, with the exception of the fourth grade class, were overall very positive toward the software and the varying difficulty levels. The first grade teacher had problems with the students navigating the software the first time they used it, but the data and comments support the fact that they were able to use it successfully after some initial assistance. The third grade class demonstrated the highest success in using the software meaningfully and enthusiastically. For the fifth grade, there were no general problems with the use of the software and most of the students experienced a decrease in the average time it took to successfully complete problems as they used the software more. However, it should be noted that they were not as engaged in the software environment as the younger students according to the teacher assessment. It should also be noted that the problems experienced with the speed of the software in the fourth grade class may have been due to the network connection for the school as none of the other classes reported slow response time; network connections are a serious consideration when choosing client-server web-based architecture.

3.4 Retrospective Analysis

The tradeoff analysis is a continual effort throughout the requirements gathering phase as well as during the architecture decision and development; issues can arise at any time that require tradeoffs for the sake of realistic development. Retrospective tradeoff analysis is useful in determining whether the strategies accomplished the overall needs of the software efficiently and identifying places in which the software must be revised before a larger release. Using the requirements gathered from the administrators and teachers, several tradeoffs had to be made within the MMO software to accommodate the limitations of the implementation and the conflicts among the quality attributes. For example, in MMO, the attribute of performance had to be resolved with the accuracy requirement. For the best performance, the smallest amount of data should be transferred over the network. For accuracy, larger software modules are necessary to insure correctness of the problem instances presented. Accuracy has the higher priority and therefore performance, specifically speed, suffered from the use of larger software modules.

Tradeoff analysis, as briefly mentioned above, is the process by which some quality attributes are only partially implemented to accommodate the need for higher priority attributes. Extensive methods for tradeoff analysis can be found such as the Architecture Tradeoff Analysis Method (ATAM) developed by the SEI group[6]. The ATAM method usually requires extensive documentation and a complete architectural evaluation guided by outside evaluators over three days; this will certainly elicit any conflicts within the system, but it is also an exhaustive process. To simplify this method of analysis and eliminate the need for outside evaluation and extensive documentation, the following chart has been developed specifically for the purpose of discovering conflicts among the quality attributes and requirements. This chart will not take the place of an architectural evaluation, but it will identify conflicting quality attributes. When a conflict is identified, a resolution strategy should be listed with it along with the relative priorities of the attributes. In general, the lowest priority attributes are partially sacrificed for the sake of higher priority attributes.

Copyright © 2005, American Society for Engineering Education

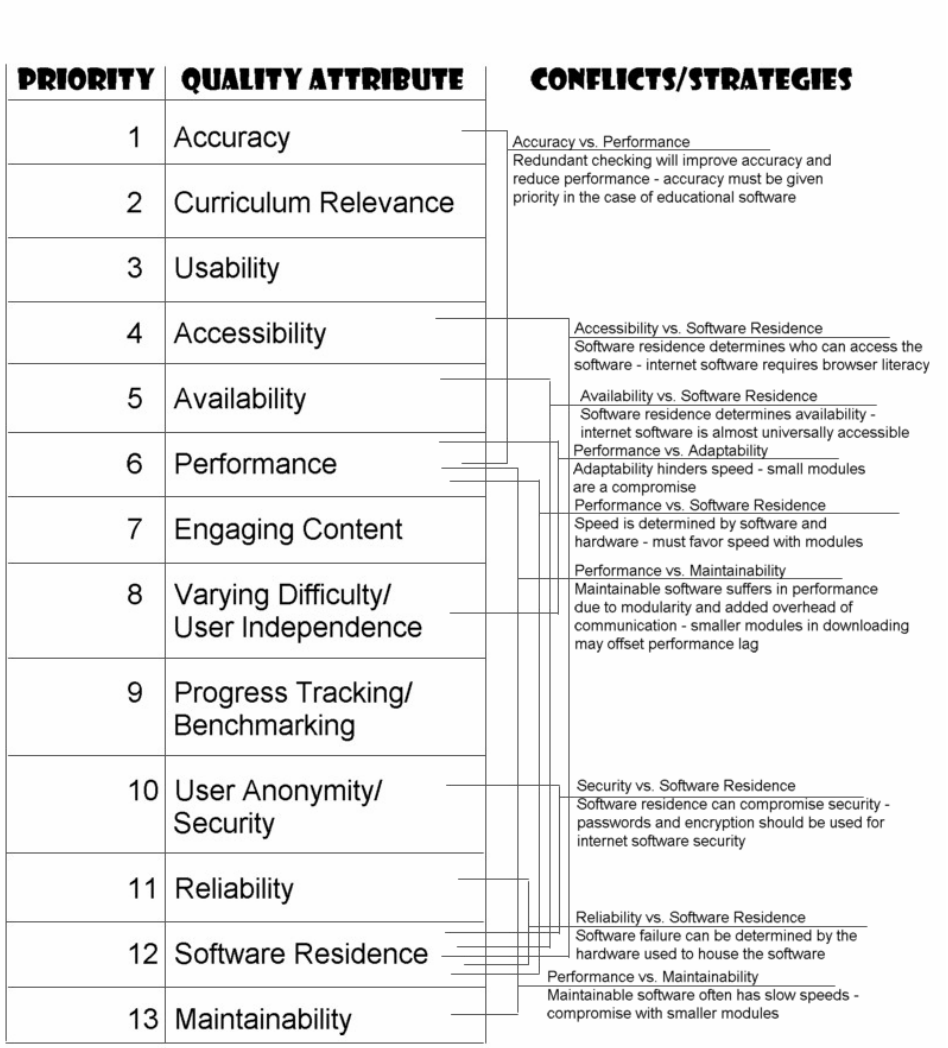

Figure 8. Tradeoff analysis chart showing the conflicts and strategies for resolution in MMO.

Reviewing the testing results from the beta classrooms and the implementation compromises in the retrospective tradeoff analysis, the MMO software can be termed a moderate success. Before a larger release, it would definitely require its own dedicated server and increased redundancy. Any lingering concerns with the security of the system could be solved by porting the existing software to a secure socket layer (SSL) connection that encrypts all data passed across the network. This would solve the problems with availability as well. Considering the problems encountered by the fourth grade class in loading the software through the internet browser, it may be advisable to construct a downloadable version of the software that connects to a backend of the PERL script to record the tracking data while giving the benefit of localizing the large software modules to increase speed and performance.

For a small scale release, there were few problems with the actual software aside from the fourth grade connectivity issues, something that could not be immediately resolved. The changeover of the architecture from a localized single module to the client-server architecture was highly successful despite the extensive recoding required. The software was changed from a standalone

Copyright © 2005, American Society for Engineering Education

Windows PC application to an internet browser application, which allows for the software to be used on multiple platforms and improves accessibility. Any future incarnations of the software will definitely retain the client-server web-based architecture that should have been chosen in the first Math with Montague software. Overall, the software development was highly successful, especially since it was conducted within a three month period.

The process used to design the software was also found to be highly effective for use in educational software development, geared specifically towards addressing the needs of the main stakeholders for a classroom environment. The complete reusable set of modified software design and analysis tools can be found below with further discussion on their general use. Based upon the leading techniques and modified for simplified and specific use, the tools used herein for the various stages of the software life cycle can be used for future educational software development. This collection has been termed the Educational Software Development and Analysis Toolkit (ESDAT), which is detailed below.

4. Educational Software Development and Analysis Toolkit (ESDAT)

The modified educational software life cycle used for the development of MMO is shown in Figure 2. The main departure from the traditional software life cycle is the breakout of verification and validation testing with the inclusion of a small release. These steps are highly relevant to software developed for the classroom and should result in refined requirements and more successful software with greater acceptance on all stakeholder levels.

(I) The main tool used for requirements analysis was the educational software requirements shell as seen in Figure 3. For larger systems, the complete Volere shell[2] may be more suitable, especially in companies and environments with a large number of stakeholders on the development side. This includes financial sponsors, who were not a consideration in the development of MMO. The modified shell presented within is well adapted for small scale development. The main adaptation is the simultaneous consideration of the needs of the administrators, teachers, and students within the same individual requirement shell.

(II) The tradeoff analysis begins within the requirements gathering phase. This involves examining potential conflicts among the general quality attributes and specific requirements of the system. When a conflict is discovered, it is necessary to employ a strategy for handling the tradeoff, or compromise, which usually favors the needs of a higher priority requirement. This process can be an extensive endeavor; however, the tradeoff analysis chart in Figure 8 can be used for a brief overview of the conflicts and strategies. Even if a larger analysis is required, the chart is still recommended for a clear and concise descriptive tool.

(III) The decision for the type of architecture a software system employs is critical to its success. A poor architectural decision, such as the architecture used for the original Math with Montague, can cause a software system to fail in the market because another type of architecture would address the requirements more naturally. One way to get a better idea of which architecture serves the needs of the proposed system is to construct the views into the architecture prescribed by Kruchten[5]; examples of these views can be seen in Figure 5. The requirements gathering should be mostly complete when the architecture is chosen, but some refinements may occur as a

Copyright © 2005, American Society for Engineering Education

result of the architecture choice. With each new requirement added or refined, conflicts within the tradeoff analysis must also be re-examined.

(IV) After the software is written using whatever tools are necessary and proper for the architecture, verification testing must take place. (V) Verification testing is the assurance that the software system under construction is being built correctly with the functionality promised by the requirements. This is the testing that occurs within the development organization before the release of the software. Verification testing can often reveal the weaknesses of the system and, hopefully, any errors within the code or hardware configuration.

(VI) After refining the software system to accommodate any errors found in the verification testing, the system should be installed as if it were being prepared for full-scale release. (VII) Any operation support that is necessary for a release should also be put into action at this point. (VIII) These are preparatory steps for a limited release, which will employ a beta testing group of actual stakeholders and end users such as a classroom full of students. The limited release will result in validation testing, which assures that the correct system was constructed. This testing should be extensive enough to determine any refinements that need to be made; the testing used for MMO was significant for the scope of this study, but a larger beta group and time period would be necessary for a definitive analysis.

(IX) The purpose of the retrospective analysis is to provide documentation of any new requirements or modifications to existing requirements found in validation testing. A retrospective tradeoff analysis should be conducted to assure that all conflicts were considered and addressed, a process that becomes easier when a full product is available for study. (X) At this point, the software should undergo any modification or maintenance necessary for a successful release. Should the retrospective analysis be largely negative, it may unfortunately be necessary to reconsider the architecture chosen and redevelop the system. (XI) Once the software satisfies the requirements with the necessary quality attributes, it will be fit for full release, at which point the traditional cycle of maintenance and eventual retirement, phases (XII) and phase (XIII), takes place.

5. Conclusions

The goal of this study was the creation of a standard method for developing educational software and elevating the standards of such software, considering all levels on which the software must operate. To simultaneously construct and validate the software engineering process, an actual software system, Math with Montague Online, was constructed and subsequently analyzed. The purpose of the analysis was to examine not only the software but the engineering process as well. The software was successful in three of the four classroom trials, but the process elicited better results in yielding important requirements and subjecting the software to validation testing that made its future needs more apparent. The reusable portions of this engineering and analysis method have been collected herein in Section 4 as the Educational Software Development and Analysis Toolkit (ESDAT) which can yield software better suited to the elementary classroom. The ultimate goal of this research is the creation of higher standards to which classroom software must adhere before it is put to use.

Copyright © 2005, American Society for Engineering Education

6. Copyright Information

Math with Montague and Math with Montague Online are Copyright © 2002-2004 Equinox Empires, all rights reserved.

7. Acknowledgements

The support provided by the National Science Foundation's Graduate Research Fellowship Program is gratefully acknowledged.

8. Bibliography

1. PFLEEGER, SHARI LAWRENCE. *Software Engineering Theory and Practice* (2nd ed.). (2001). New Jersey: Prentice Hall.
2. ROBERTSON, SUSAN and JAMES ROBERTSON. *Mastering the Requirements Process*. (1999). Harlow, England: Addison-Wesley.
3. SOUTH CAROLINA DEPARTMENT OF EDUCATION. "Mathematics: Curriculum Standards." Grades K-2, 3-5. Retrieved on September 12, 2004, from http://www.myscschools.com/offices/cso/mathematics/standards.htm.
4. GARLAN, DAVID, J. P. SOUSA. "Documenting Software Architectures: Recommendations for Industrial Practice." CMU-CS-00-169. October, 2000.
5. KRUCHTEN, PHILLIPE. "Architectural Blueprints – The '4 + 1' View Model of Software Architecture." *IEEE Software* (12). November, 1995. pp. 42 – 50.
6. CARNEGIE MELLON SOFTWARE ENGINEERING INSTITUTE. "The Architectural Tradeoff Analysis Method (ATAM)." Retrieved on October 10, 2004, from http://www.sei.cmu.edu/ata/ata_method.html.
7. BASS, LEN, PAUL CLEMENTS, et al. *Software Architecture in Practice* (2nd ed.). (2003). Boston, MA: Addison-Wesley.

9. Biographical Information

THEODOR RICHARDSON is a Ph.D. student at the University of South Carolina (USC) with an M.E. in Computer Science. Ted has worked for two years in the elementary and middle school classrooms as a Fellow in the NSF GK-12 program at USC. Ted is the author of both Math with Montague and Math with Montague Online and the sole proprietor of the Equinox Empires software company in South Carolina.

JED LYONS is an Associate Professor of Mechanical Engineering at the University of South Carolina and the Director of the South Carolina Center for Engineering and Computing Education. He teaches laboratory, design, and materials science to undergraduates, graduate students and K-12 teachers. He researches engineering education, plastics and composites. Jed is the GK-12 PI.