

# Project description

## 1 Project context

Computer software systems now pervade every aspect of human endeavor, and our lives are ever more dependent on their correctness and usability. Activities such as business organization and management that were once the exclusive domain of human workers are now supported or entirely automated through software. Mechanical control of sophisticated devices like medical instrumentation and automobiles is steadily being replaced by software control. The domains in which such software systems operate are highly complex. Consequently, among experts in these domains there is typically a rich history of “know-how”: artifacts and techniques for working in such domains [Johnson98].

Developing software of such complexity requires communication with a wide variety of stakeholders [Poole03]. Customers have a vision not only of the software product but also its place within a larger problem context. End users often have well developed knowledge of the domain area and a keen sense of the usability of a product. Marketing and legal experts also have unique and valuable perspectives. It is crucial for software developers to interact with these and other stakeholders, to get a clear picture of the software’s context and goals, and to glean the expertise of those who actually work in the domain. Software developers themselves form a rich assortment of stakeholder groups, including designers, quality assurance experts, technical writers, and managers.

While communication about software is important, it is complicated for several reasons. First, software engineering is a nascent field, without a time-honored, universal lexicon. The variety of uses for software makes it difficult to find common terms and patterns<sup>1</sup>. The wide range of application areas draws together stakeholders with very different backgrounds and little in the way of a common vocabulary. Finally, the sheer intangibility of software presents a challenge; there are no visible parts that can be easily labeled. These factors contribute to negative feelings among developers regarding software documentation. A recent survey reports that software engineers tend to believe that “[d]ocumentation is often poorly written” and “[f]inding useful content in documentation can be so challenging that people might not try to do so” [LSF03:36].

Perhaps most critically, the process of communicating about software is largely ignored in undergraduate CISE education. Students get little or no practice in communicating with other stakeholders and documenting the software they create, and they are not given direction in how to do this. The CISE curriculum is strongly slanted toward the implementation details of computers. The precision of programming languages and computer hardware is comforting to Computer Science students, who know how to manipulate them well, but it can prevent them from looking at the human problems that motivate software development in the first place. The process of interacting with human stakeholders is seen as “soft” material, not worthy of serious attention. Ferguson notes a similar tendency in mechanical and structural engineering education: “the implicit acceptance of the notion that high-status analytical courses are superior to those that encourage the student to develop an intuitive ‘feel’ for the incalculable complexity of engineering practice in the real world” [Ferguson92:168]. Our project aims to bridge this gap in current software engineering pedagogy by helping students to solve not only the analytical problems, but the *communication* problems that also reside at the heart of software development.

---

<sup>1</sup> Research in software architectural styles and design patterns [Shaw94, GHJV95, ShaGar96, MKMG97] strives to find such commonalities.

The new Software Engineering degree program at Michigan Technological University (MTU) [OttWal03] has inspired efforts to integrate training in communication and documentation into the curriculum. The proposed project seeks to accomplish this by drawing from current research in software engineering and technical communication. This section provides background on the particular approaches to be used in this project (§1.1 and §1.2), related curricular development (§1.3), and the academic context in which the materials will first be used and evaluated (§1.4).

## 1.1 Requirements documentation

Communication is required to understand what software must do in order to be useful: that is, what problem it must solve. This sort of activity falls under the general heading of *requirements elicitation and analysis*. As requirements are constructed, users must be able to relate their expertise in the problem domain effectively and easily. Developers must be able to construct a view of the software problem context and bounds. Descriptions of complex problems must be structured in a way that eases decomposition during design.

Several studies point to deficiencies in requirements as the primary cause of large-scale project failures [CKI88, Davis90, Glass98]. This can be traced to a lack of commitment and trust between customer and developer. According to a recent study, developers consider risks such as “failure to gain user commitment” and “lack of adequate user involvement” more important than such serious risks as “introduction of new technology” and “insufficient/inappropriate staffing” [KCLS98]. This evidence indicates the need for improved communication during requirements elicitation and analysis.

This project will instruct students in the use of *scenarios* and *Problem Frames* for the purpose of requirements documentation. The two approaches are described below.

### 1.1.1 Scenarios

Scenarios are user-centered, narrative-based representations of software functionality. The goal of the scenario approach is to provide a flexible, rich means of discourse that is available to all stakeholders. While scenarios vary in format and level of formality [Sutcliffe03], they are all stories, told from a user’s perspective, of how a tool is used. By placing the user at the center and avoiding undue formalism, scenarios help to bridge the communication gap between developers and other stakeholders.

Scenarios are used widely in the field of technical communication, for software as well as other technological artifacts.<sup>2</sup> In technical communication, scenarios are developed through a “rhetorical approach”: an approach that weds the arts of solving technical problems with the arts of creating effective communications [JohSim90]. Further, this approach strongly advocates for the development of scenarios that allow students to better prepare themselves for situations they will confront outside the classroom [CouGol85, KynSto99].

There are many places in the software process where scenarios can play a role:

- They can serve as a medium for requirements elicitation. In cases where software is being developed to automate a process currently performed by other means, users can create scenarios set in the current, software-free environment to which they are accustomed. They tell their stories using the vocabulary of the domain they know well, and in doing so they convey important contextual information that might otherwise be overlooked by other stakeholders. The focus on realistic tasks, performed by experienced users, helps to ensure the usability of the software product.

---

<sup>2</sup> Guidelines for constructing scenarios, taken from the technical communication field, are included as a supplemental document.

- Scenarios are also a simple common means of representation that can be understood by all stakeholders. They can serve as the basis of an “inquiry method” [RosCar02] that enriches stakeholders’ understanding of one another’s needs and constraints. Since they are expressed in plain language and mention specifics, they encourage the kind of constructive questioning that fleshes out important details. An abstract model written in a generic vocabulary may not provoke such questioning [Sutcliffe03].
- Even developers can benefit from the loose format of scenarios when exploring possible design directions. In this case, use of scenarios is closely related to that of prototypes or mock-ups. In addition to its benefit in inspiring design decisions, scenarios are test cases that can be used to validate the prototype. The *use case* view in the Unified Modeling Language (UML) [RJB99] is an attempt to formalize scenarios for design documentation.

While scenarios are valuable, they can be overly labor-intensive without proper management methods [SMMM98]. Recently, scenario-based requirements engineering methods have emerged to guide the creation and development of scenarios. Some of the principles behind these methods will be integrated into the proposed curriculum.

- The *ScenIC* method [Potts99] views documentation as the “memory of the project”. Scenarios provide *episodic* memories, which instantiate and enhance *semantic* memories (general principles found in traditional requirements documentation). Taking cues from psychology, the *Inquiry Cycle* of ScenIC provides a form of memory management, involving *expression* of requirement goals through episodic or semantic memories; *criticism* of the requirements documentation through issue-raising guidelines, and *refinement* of the documentation. ScenIC provides a means of mapping scenarios to semantic memory material. It also establishes important coverage criteria, to ensure that the scenarios chosen are extensive enough to represent the full software problem.
- The *SCRAM* method [SMMM98, SutRya98] extends the Inquiry Cycle with further detail on how to generate and use scenarios for requirements validation. Scenarios elicited from stakeholders form use cases, generic descriptions of system use. These use cases are compared against a library of abstract models, representing generic software applications. Each model has a set of requirements, which are applied to any appropriate use cases; in this way, requirements can be reused. SCRAM provides heuristics for scenario generation from use cases. Particularly important are the heuristics for generating “abnormal” cases that might be overlooked by developers and other stakeholders.

### 1.1.2 Documenting, structuring, and analyzing software problems through Problem Frames<sup>3</sup>

The Problem Frames approach [Jackson01] views software problems as complex interactions between the target system (the *machine*) and its environment. It concentrates exclusively on the problem to be solved, as represented through *shared phenomena* between the machine and various environmental domains. A premium is placed on carefully *locating* and *bounding* the problem, identifying the roles of the machine and of environmental factors. This encourages precise description of the problem, in a stakeholder-friendly way that requires no machine-specific knowledge.

Each Problem Frame is a generic class of simple, commonly occurring problem, similar to a design pattern [GHJV95]<sup>4</sup>. Associated with a frame is a set of characteristic *concerns*: complicating issues that must be

---

<sup>3</sup> An brief example of Problem Frames in use is included as a supplemental document.

addressed in documentation and communicated to stakeholders. Frame concerns are evolving entities, growing richer with the experience of solving ever more software problems. Realistic software problems are *multiframe* problems: combinations of interdependent frame instances. Problems may be analyzed by decomposing them into their constituent frames.

By focusing on problems, developers maintain a middle road between the formality of computing and the informality of human needs and desires. The identification of environmental domains and shared phenomena creates a clear, precise vocabulary that all stakeholders can understand. Problem Frames therefore provide a vital link between the human focus of scenario development and the computing concerns of software design.

## 1.2 Design documentation

Communication also plays an important role in understanding software design: how a software product solves a problem. Developers working concurrently on components of a product must understand how their components interact. This is best done early in the development process, before there is fully elaborated code to examine. Furthermore, software components pass from developer to developer over time, particularly with the high employee turnover of today's software firms. New developers must be brought up to speed on design. Reading code is an inefficient way of doing this; more literate means of conveying meaning are needed.

Abstract State Machines (ASMs) and the related AsmL tool have emerged as highly effective ways of documenting design. The proposed project seeks to implement ASMs and AsmL in the undergraduate curriculum. They are described below.

### 1.2.1 Documenting software design through Abstract State Machines<sup>5</sup>

The ASM project [ASM] is an attempt to create a practical design documentation methodology from mathematical logic and traditional formal models of computation [Turing36, CooRec73, Schönhage80]. ASMs take a fully abstract view of both data and operations [Börger95a]. An ASM state is an *algebra*, defined in terms of a *vocabulary* of developer-defined mathematical functions. As with scenarios and Problem Frames, the vocabulary of an ASM reflects a user-centered view of the modeled software product; system states can be described in terms accessible to all stakeholders.

State changes are sets of *updates*, each update changing the interpretation of a single function name at a single location. When a program is executed at a state, it gives rise to a set of updates. An ASM program consists of *rules*, or guarded assignment statements, that transform the state by updating various functions. Special rules exist for nondeterminism and parallelism. An ASM model may be *sequential*, with a single *agent* executing its program, or *distributed*, with a dynamically changing set of agents, each executing its own program.

When creating an ASM model, one chooses a set of functions interpreted at each state, and a program that manipulates those functions. In doing so, the specification's level of abstraction is implicitly selected. The very general notions of algebras as states and guarded assignments as operations allows maximal freedom in choosing the abstraction level. The idea is simple but powerful: the ASM thesis is that any algorithm can be modeled at its natural abstraction level by an appropriate ASM [Gurevich93, Gurevich00]. This allows developers to explore a high-level design and anticipate how its different features will interact. At the same time, an ASM model is quite different from a simple prototype or a reference implementation, since it is a

---

<sup>4</sup> Jackson describes design patterns as “look[ing] inwards towards the computer and its software and Problem Frames as “look[ing] outwards to the world where the problem is found” [Jackson01:xii].

<sup>5</sup> A short introduction to Abstract State Machines is included as a supplemental document.

fully general representation of the software product at the chosen level of design detail. There is no need to make premature implementation decisions to produce a complete document.

A crucial aspect of the ASM methodology is its simple mathematical foundations [Gurevich95]. The ASM language has only a handful of syntactic constructs and straightforward semantics. Although simple syntax does not by itself lead to simple specifications, an appropriate modular design results in an elegant, manageable specification [Börger95]. The simplicity of ASMs makes them palatable to software developers, as shown by its successful application in industrial settings [GlaKar97, BPS00, BGL03]. Most notably, Microsoft has chosen ASM as a new method for documenting, prototyping, and testing its software [FSE, Gurevich01, BCSV01, BarSch03]. Familiarity with advanced mathematical concepts is not required. In fact, an ASM program closely resembles a program written in an imperative programming language with a Pascal-like syntax. Personal experience has shown that undergraduate and graduate students can pick up the essentials of ASM quickly and produce substantial design models of their own in the span of a one-semester course.

### 1.2.2 Software simulation and testing using AsmL

The ASM language is essentially a high-level programming language, so in theory each ASM program is directly executable. The promise of executable ASM models is fulfilled by the *AsmL (Abstract State Machine Language)* tool developed at Microsoft Research [AsmL]. Through its automated support environment, AsmL treats ASM programs as executable components. AsmL extends the basic ASM language with a flexible type system, including type parameterization and type inference. AsmL implementations can target Microsoft's COM and .NET platforms, and AsmL components are fully interoperable with components written in other .NET-based languages. AsmL may be used to execute prototypes of preliminary designs, for the purpose of customer validation. It can also serve as a tool for designers to explore the consequences of various implementation decisions.

While ASMs do not have an associated verification methodology, they are well suited as input for testing, and the execution environment of AsmL provides facilities for verification. First, a high-level AsmL model can be run in parallel with a lower-level model, to check that the behaviors of the two models agree. Second, the rigor of ASMs provides a foundation for algorithmic test case generation. AsmL features an integrated test generation environment called *asmIt*. It can be used to automatically generate test cases (sequences of method calls and parameter value choices) from an AsmL model. A finite state machine is generated [GGSV02] and test cases produced from traversals of the machine's states.

## 1.3 Related work

In the software engineering education literature, there has been very little published material regarding the use of scenarios, Problem Frames, or ASMs. The textbook by Rosson and Carroll [RosCar02] is a good introduction to scenario-based requirements analysis, but there are no reports in the educational literature on developing a curriculum around scenarios. The current introductory texts on Problem Frames [Kovitz99, Jackson01] are written for professionals or graduate students rather than undergraduates. A recent book on ASMs by Börger and Stärk [BörStä03] gives a thorough treatment directed to a general audience, but its pace and scope seem rather overwhelming for a typical undergraduate student. Apart from an earlier paper by some of the participants in this project [HMW03], there appears to be no published work on placing either ASMs or Problem Frames in a wider context in the undergraduate curriculum.

There have been reports on efforts to build stakeholder awareness into the CISE curriculum. Dean and Hinchey [DeaHin95] describe a formal methods course in which the instructor acts as a customer. There are periodic interactions between students and the instructor, in which ambiguities are resolved and the formal documentation is annotated with customer-friendly natural language. The interaction is based on a single

“case study”: that of a “metropolitan railway system”. It is unclear whether this case study is based on a real railway system. Wahl’s course on usability testing [Wahl00] involves human participants; here the users are volunteers from another Computer Science course. The interaction with users is limited and static in nature, however, consisting primarily of an exit interview. Zowghi and Paryani [ZowPar03] have developed a requirements engineering course involving role playing. In this course, each student participates in two software interactions: one in the role of developer and the other in the role of customer. This has the advantage of giving students empathy for other stakeholder roles. However, it seems difficult to develop realistic scenarios this way; students as pseudo-users lack the domain-specific knowledge and the intuitive sense of usability that real users possess.

## 1.4 Project infrastructure

The proposed project brings together researchers and educators from multiple disciplines and institutions. This section briefly describes the academic settings in which the new curriculum is to be developed and implemented.

- **Michigan Technological University**

**Computer Science and Software Engineering.** The Computer Science department at MTU has a long history in training undergraduates for careers in software development. Currently, there are approximately 400 undergraduate Computer Science majors and 60 M.S. and Ph.D. students. The department recently started an undergraduate degree program in Software Engineering, one of the first in the nation [OttWal03]. This program will serve as the initial test bed for the materials developed in this project.

The Software Engineering program puts a premium on experience with realistic software development situations. Students build educational-themed applications of their own design in the junior-level CS3141 “Team Software Project”. The senior-level CS4791-CS4792 “Senior Design Project” courses involve development of a software product intended for actual use. In the past, this course has had approximately 6-10 students per term, though this is sure to increase with the new Software Engineering degree program. Ideas for past projects have come from within MTU and from contacts in industry.

The courses CS4711 “Introduction of Software Engineering” and CS4712 “Software Quality Assurance”, each offered once a year, provide in-depth treatments of fundamental software engineering techniques. Each course has approximately forty students. The latter course concentrates on requirements documentation and analysis, formal design documentation, and black-box (specification-based) testing. It will be the first major venue for our new materials.

**Scientific & Technical Communication (STC).** With approximately 100 students in its major, Michigan Tech’s undergraduate Scientific and Technical Communication (STC) Program is one of the largest degree-granting programs in technical communication in the country. Within the program, there are two available specializations—Scientific and Technical Communication (STC) and Scientific and Technical Arts (STA)—the first of which offers a concentration in a technical area and the second in writing, media, or modern languages.

The program’s philosophy is threefold—to improve the practice of technical communication, to examine and teach technical communication as an art and science, and to provide educational contexts that stimulate expertise and leadership in technical communication theory and professional application.

The program's intent is to equip students for positions as writers, media developers, and independent contractors and consultants in industry, or to prepare them for study at the graduate level. To these ends, MTU students gain communication and computer skills with strong technical backgrounds. They also gain experience working with clients, in co-ops, project management classes, or as consultants in our state-of-the-art computing center.

MTU's graduate program, Rhetoric and Technical Communication (RTC) offers both master's and doctoral degrees that engage faculty and students in interdisciplinary work across a range of fields, focusing on the roles of technology and communication within social contexts. The RTC program draws on the scholarly expertise of approximately thirty distinguished graduate faculty whose teaching and research are mirrored in the program's three streams of graduate study: Rhetoric, Composition, and Literacy, Technical Communication and Technology Studies, and Communication in Cultural Contexts.

- **Kettering University.** Kettering offers the only undergraduate degree program in computer science with a full cooperative education component. Beginning in their first year, students alternate three month academic terms with three month work terms, in which students gain real-world experience in paid, professional settings. At the end of five years, students graduate with a bachelor's degree and the equivalent of up to 2.5 years of full-time work experience. This unique blending of academic study and practical work experience allows students to put their newly acquired knowledge to practical use, as well as to bring their life experiences back into the classroom to enhance learning. Kettering's degree program in computer science is relatively young; its first degrees in computer science were awarded in 1999, and it graduates approximately 20 students per year.

The course CS-471 "Software Engineering" is a classical undergraduate course in fundamental software engineering techniques. Typically taken at the junior level, CS-471 gives students the opportunity to work in teams of 3-4 students on a large software project which spans the entire three-month term, while learning about software engineering techniques through classroom instruction. The course is offered twice a year and has approximately 20 students per term.

## 2 Proposed activities

The proposed project will draw the research topics described in §1.1 and §1.2 into the undergraduate curriculum. The new educational materials will be used and evaluated first at MTU. Their general relevance and benefit will then be assessed through their implementation and evaluation at Kettering. In addition, the materials and the evaluation results will be disseminated to the national Technical Communication and Computer Science communities. This section describes the development, implementation, evaluation, and dissemination of the new curriculum, and it lists the anticipated benefits of this work at several levels.

### 2.1 Educational materials

This project will develop educational materials for use within the MTU Software Engineering curriculum. These materials will cover:

- elicitation and analysis of scenarios;
- documentation and analysis of problems using Problem Frames;
- documentation of design using Abstract State Machines;
- prototyping and testing of design using AsmL.

The materials to be developed are detailed below.

### 2.1.1 Scenarios

In the new curriculum, students in the MTU Senior Design Project will get experience in real requirements elicitation and analysis using scenarios. In this year-long course, students are expected to develop a practical software product in accordance with requirements from real stakeholders. The proposed project will develop a framework that students can follow to construct scenarios, using ideas from the ScenIC and SCRAM methodologies.

The semester-long Software Quality Assurance course has more students and less time than the yearlong Senior Design Project. Getting Software Quality Assurance students to interact with real users in the same way as the Senior Design Project is impractical, but the active inquiry and user focus of scenarios is something very important to impart to these students. This proposal will leverage the experiences of the Senior Design Project students to create rich, realistic case studies for the Software Quality Assurance course.

To create these case studies, Senior Design students will build written histories of their experiences with customers and users. Each student team will maintain a log of questions. Based on these logs and on their case study work, students will be required to submit one-page progress reports on a weekly basis. These documents may range from rhetorical analyses of customers and end users to reports on students' communication with other designers and domain knowledge experts. Progress reports will be read, evaluated, and returned to students, who will collect them in a file or portfolio. At the end of the semester, students will be asked to write a final one-page assessment of their written communication skills and to submit this with their portfolio of written work. Using a rubric that identifies the characteristics of precise and effective communication, portfolios will be assessed for their rhetorical, mechanical, and organizational strengths and weaknesses. The aim here is to evaluate students' abilities to write documents that communicate effectively with their audiences.

The portfolios from the Senior Design Projects will provide the raw material for the Software Quality Assurance curriculum. During the summer after each Senior Design Project, graduate students will compile the portfolios into case studies. In Software Quality Assurance, the interaction with customers and users will be simulated by hiding some of the case study data. Students will be presented with a skeletal version of the requirements derived in the Senior Design Projects. In order to produce a precise, unambiguous requirements document, they will have to uncover the missing information. The instructor will act as the proxy user and release information only when it is explicitly asked for. Each Software Quality Assurance team will have scheduled interviews with the instructor, during which they can ask the questions that will clarify the software requirements.

### 2.1.2 Problem Frames

Students in Software Quality Assurance and Senior Design Project will get additional support for using Problem Frames, through the following materials:

- **PFEEdit: a graphical editor tool for Problem Frame diagrams.** The project will develop a tool, tentatively named *PFEEdit*, for the creation and editing of Problem Frame diagrams. Problem Frames lend themselves well to graphical representation, with nodes (the machine domain and various environmental domains) connected by edges (labeled with shared phenomena). Initially, however, students have trouble with creating diagrams. The difficulties are due not to arcane details of notation but rather to deeper misunderstandings about Problem Frames.

Here the simple and evocative diagram format can act as a tool for learning, if implemented in an interactive computer-based way. A Problem Frame editor tool with a graphical user interface will help students by preventing common mistakes and providing informative dialogues. It will also appeal to Software Engineering students, who typically find computer interaction enlightening and empowering.

- **Integration with scenarios.** Problem Frames mesh gracefully with scenario-based descriptions. While scenarios are rich with user-centered information, they may be expressed in vague language that is open to misinterpretation by developers. Problem Frames define a precise language of environmental domains and shared phenomena, and they direct developers toward a decomposition, while maintaining the focus on the problem and the user.

The proposed curriculum will include development of multiframe problem descriptions, based on problems elicited from stakeholders. The rich complexity of real world problems will motivate the use of Problem Frames and decomposition.

### 2.1.3 Abstract State Machines and AsmL

While students in past offerings of the Software Quality Assurance course have studied ASMs, with a good deal of success, there has so far been no in-class exposure to AsmL or any other ASM execution environment.<sup>6</sup> It is clear that the full benefit of ASMs cannot be felt unless they can be used for prototyping and testing. Furthermore, students feel frustrated about having to write programs in a new programming language that they cannot even run. This project aims to integrate AsmL into the curriculum and give students hands-on experience with formal design documentation.

- **Improving and extending the ASM primer.** An introductory text on Abstract State Machines, the ASM Primer [HugWal02], has been written for use in the Software Quality Assurance course. The tone and pace of this document are chosen for an undergraduate Computer Science audience, in contrast to the terser, more mathematically rigorous descriptions found elsewhere [Gurevich91, Gurevich95, SSB01]. The primer places ASMs in the software development process and provides a variety of examples. It also addresses many of the questions that commonly surface in the Software Quality Assurance course.

The proposed project will improve on this initial work in the following two ways:

- **AsmL support.** All the examples in the primer will have executable AsmL versions in the expanded version. Students will be able to interact with the examples, rather than simply reading them as static documents on the page.
- **Coverage of an industrial-strength case study.** A secondary tutorial document will cover the database recovery problem addressed in an earlier ASM paper [GSW97]. The system to be designed is a centralized database management system that services operation requests from multiple users, maintaining information in a log so that the database can be rolled back to a consistent state in the event of a failure. This problem has several attractive properties. Unlike the examples in the first primer, the environment interacts with the database manager in subtle ways. Moreover, there are many levels of detail in a solution to this problem.

---

<sup>6</sup> This is due to the current lack of support for the Windows operating system in the student Computer Science labs at MTU. Thanks to the new interdepartmental cooperation, however, Software Quality Assurance students will now be able to use AsmL on Windows-based computers in the STC labs.

- **Using AsmL to execute and test ASM programs.** Currently, students in Software Quality Assurance develop ASM documentation for projects of their choosing. These typically come from industrial or academic software projects in which they have been involved. In the new curriculum, students will develop this documentation using AsmL. This will have many benefits:
  - Many of the typical mistakes that students make in drawing up ASM documentation can be caught through the AsmL compiler.
  - The ASM documents can be directly executed, providing prototypes for usability analysis.
  - AsmL can serve as a vehicle for black-box testing, in which the ASM documentation serves as the requirements against which the implementation is verified.

## 2.2 Dissemination of results

The educational needs addressed in this project have wide-ranging applicability: in graduate and professional settings, as well as the intended undergraduate setting. The proposed materials will be used and evaluated initially at MTU but designed to transfer easily into other educational programs. The following activities will promote the project as a national model of excellence, exposing its results to the larger Software Engineering community and providing feedback for the future improvement of project materials.

- **Implementation at Kettering University.** The materials used in the Software Quality Assurance course at MTU will also be used in the Software Engineering course at Kettering. This will help to determine their effectiveness in a variety of academic settings.
- **Software Communication & Documentation Workshop.** This project will provide the PIs with useful experience in teaching the material, which will be worth sharing with the larger CISE educational community. Moreover, input from other experts in requirements elicitation and analysis and software documentation will help the project greatly. To promote this exchange of ideas, a week-long workshop based on these topics will be held at MTU, with a focus on drawing them into the undergraduate curriculum. It will feature at least one guest speaker. Participants will be encouraged to submit position papers; during the workshop, these will be developed into full papers, which will be compiled into workshop proceedings. The PIs will seek to publish the proceedings.
- **Educators' tutorial.** After the project has developed sufficiently in terms of course material and experience, it will be beneficial to introduce the material to instructors in other institutions. To this end, a week-long tutorial session for CISE educators will be held at MTU. Participants will receive short, intensive introductions to scenarios, Problem Frames, and ASMs, and will engage in some of the case studies created earlier in the project. The PIs will receive feedback from the participants on how to improve the material for wider use.
- **Public distribution of materials.** All the curricular material developed in this project will be made publicly available on the World Wide Web. The online material will include:
  - **Integrated case studies**, in HTML format. These will draw from the real-world software problems encountered in the Senior Design Projects, and will include annotated documentation in the form of scenarios, Problem Frames, and ASMs.
  - The expanded **ASM primer**, with executable AsmL examples.
  - The **PFEdit** tool.

- **Publications.** The results of the project will be reported at CISE educational conferences such as SIGCSE and CSEET. The PIs will also seek to submit the ASM Primer for publication as a textbook.

## 2.3 Evaluation

This evaluation seeks to determine the degree to which engineering students instructed in rhetorical analysis<sup>7</sup> can communicate with stakeholders in a way that “accommodates technology to the user” [Dobrin83:242], thus bridging the gap between technical imperatives and human needs. To do so, three major project goals are identified; evaluation instruments and data analysis methods are described for each.

### 2.3.1 Goal One: Enlighten students about the needs and knowledge of other stakeholders.

**Evaluation Instrument:** A questionnaire that indicates the extent to which students view project stakeholders from a rhetorical perspective, to be used before and after case study instruction. The questionnaire will function as a summative evaluation, assessing the progress students have made in considering stakeholders’ needs and knowledge.

**Data Analysis:** Contrastive content analysis will be used to analyze student answers before and after case study instruction. Examining the contrast between answers before case study work begins and after it is complete will help determine how effective scenarios have been in developing students’ rhetorical awareness of audiences. Questions will be contextualized using (1) a description of a customer with specific views of the software product and its application as well as (2) end users with robust domain knowledge.

- (1) Based on the description of the customer and end user, how would you explain what a stakeholder is?
- (2) Using the description, explain what you, as a software designer, would need to know, if anything, about this project’s stakeholders.
- (3) Using the description, explain how you would find out what you need to know about your customer and end user.
- (4) Given the expectations and knowledge of the customer and end users, when would you find out what you need to know about them?
- (5) Given the expectations and knowledge of these stakeholders, explain how often would you plan to be in contact with them, if at all.
- (6) Why would you want to be in contact with these stakeholders, if you think you might want to do so?
- (7) What kind of knowledge, if any, would these stakeholders have that might be useful to you as a designer?
- (8) What sorts of needs, if any, would these stakeholders have that might be useful for you to know about?

### 2.3.2 Goal Two: Empower students to be in active communication with stakeholders.

**Evaluation Instrument:** Videotaped interviews simulating discussions among designers, customers, and clients to identify stakeholders’ knowledge, needs, and goals. Using questions two, three, seven, and eight from the questionnaire described previously, students will develop a set of questions that they will use to engage in active communication with their customers and end users.

---

<sup>7</sup> Here, “rhetoric” or “rhetorical analysis” means an awareness of or a method for identifying the most effective ways to make technical information accessible to those who will use it or whose lives will be affected by it.

Videotapes will be used as both formative and summative measures and thus will be recorded before, during, and after case study instruction in rhetorical analysis.

*Formative.* Investigators and instructors will tape and evaluate the first set of videotapes in order to identify lapses in students' audience awareness and adjust instructional goals for Software Quality Assurance accordingly. Using the same information, instructors will meet with students before case study work begins to set individual learning goals in rhetorical analysis.

*Formative.* At mid-semester, investigators and instructors will tape and evaluate the second set of videotapes in order to identify gains and lapses in students' audience awareness. They will adjust instructional goals for Software Quality Assurance accordingly. Using the same information, instructors will then meet with students to modify individual learning goals for subsequent instruction in rhetorical analysis.

*Summative.* After case study instruction concludes, investigators and instructors will tape and evaluate the third set of videotapes in order to measure students' overall progress in carrying out conversations with clients and end users that produce robust profiles of these stakeholders and thus lead to "communicative effectivity" [Ornatowski98:35].

**Data Analysis:** Thematic coding will be used to identify recurring patterns that emerge from videotape data. Themes might focus on stakeholder involvement, attitudes about expert/novice exchange, or domain-specific knowledge. Examining these patterns before, during, and at the close of case study work will help determine how effective scenarios have been in developing students' rhetorical awareness of audiences.

### **2.3.3 Goal Three: Enable students to communicate precisely and effectively.**

**Evaluation Instrument:** A portfolio of written documents, to be maintained throughout instruction and then submitted for evaluation. For the past decade, programs in Rhetoric/Composition have used portfolios as a way to assess student writing. A similar method will be used to determine engineering students' abilities to write documents that meet their audiences' needs for clarity, precision, and accuracy. Portfolios will be used as formative and summative evaluation instruments, assessing first the ongoing and then the overall progress students have made in communicating precisely and effectively with a variety of stakeholders as they develop software for them.

*Formative.* On a weekly basis, students will be required to submit one-page progress reports, which may range from rhetorical analyses of customers and end users to reports on their own communication with other designers and domain knowledge experts. Using a rubric that identifies the rhetorical, mechanical, and organizational characteristics of precise and effective communication, instructors will read and evaluate these weekly reports in order to identify gains and lapses in students' audience awareness and adjust instructional goals for Software Quality Assurance accordingly. Instructors will return evaluated reports to students who will collect them in a file or portfolio.

*Summative.* At the close of instruction, students will be asked to write a final one-page assessment of their written communication skills and to submit this with their portfolio of written work. Using a rubric that identifies the characteristics of precise and effective communication, instructors will score portfolios for their rhetorical, mechanical, and organizational strengths and weaknesses.

**Data Analysis:** Resulting data will be coded and entered for computer analysis using statistical programs appropriate to analyze either continuous (formative) data or categorical (summative) data. Analysis and reanalysis cycles will determine conclusions about the overall effectiveness of scenarios in developing students' rhetorical awareness of audiences.

### 2.3.4 Additional measures

**Evaluation Instruments:** (1) students' ratings of scenario instruction; (2) independent appraisals by outside observers of the quality of student performance as they use new strategies of communicating with shareholders; and (3) students' opinions about the impact of scenario instruction on their abilities to develop software for users. These ratings, appraisals, and opinions will be used as summative evaluation instruments, indicating attitudes toward scenario instruction.

**Data Analysis:** Resulting narrative evidence will be examined using content analysis.

### 2.3.5 Evaluation supervision and assistance

Prof. Johnson will be the PI serving as the program evaluation coordinator. He has ten years of experience as a program evaluator at Miami University and at MTU. Prof. Linda Ott, the chair of the MTU Computer Science department, will serve an advisory role in managing the evaluation plan. She is an experienced program evaluator, and her expertise in software engineering, particularly in software measurement, will be of great use.

## 2.4 Local and national benefits

The primary goal of the proposed project is the education of literate, actively inquisitive software engineers. This will have wide-ranging effects at institutional and national levels.

- **National benefits.** By enriching the software engineering curriculum, this project will help to improve the national software engineering culture. Training future software developers in communication and documentation techniques will enable them to produce higher quality software, better attuned to the needs of users and less prone to bugs. With a tradition of careful, literate documentation, maintenance of software will be easier. Active communication with customers and users will lead to higher customer satisfaction and will avoid the "requirements churn" that leads so frequently to project cost overruns [HutKno95]. The special skills developed in this curriculum will give the nation's software engineering workforce a competitive advantage and provide job security in the increasingly globalized world of software development.

Of course, the project will also benefit students in CISE disciplines, the software developers of the future. Students with the skills exercised in the proposed curriculum will be happier software engineers, better able to express their doubts and concerns and therefore less susceptible to frustration and burnout. The focus on real-world problems will attract those who would otherwise find CISE uninteresting. This in turn will result in a larger pool of talented developers for tomorrow's software needs. The intensive communication with users and customers will prepare students particularly well for software teams employing the new "agile methods" [HigCoc01, Boehm02] such as Extreme Programming [XP, Beck00]. These methods are characterized by a high degree of interaction among stakeholders throughout a project's lifespan.

- **Benefits to women.** One of the most exciting aspects of this project is the strong likelihood that it will increase interest and retention of women in Software Engineering. The near absence of women in computing is a phenomenon that is evident locally at MTU and nationally in both academia and industry [Camp97, AAUW00]. Women account for only 7.1% of undergraduate Computer Science students at MTU [EnrSta], even lower than the 18.8% result of a recent CRA survey [VFH03].

In a recent book, Margolis and Fisher have explored why women reject computing disciplines and propose some ways to attract them [MarFis02]. One of their principal suggestions is "computing with a

purpose”. The authors claim that “[a] curriculum that places technology in the context of its real-world uses and impact is appealing to female students” [MarFis02:131]. In conjunction with the mandatory training in the technical details of computing, they propose educational experiences “that situate the technology in realistic settings” [MarFis02:131]. This is exactly what the project aims to provide, through its emphasis on scenarios and interaction with real stakeholders.<sup>8</sup>

- **Benefits to the CISE educational community.** This project will put into practice new ideas involving student interaction with real users. It will also explore the use of new computer-assisted software engineering tools in the classroom. Through the workshop, tutorial, and other dissemination activities, it will inspire further work in teaching requirements elicitation and formal software documentation. There are many promising directions for future curricular development, based on this work; for instance, while this project is centered on upper-level undergraduate courses, the concepts can and should be integrated into earlier stages in the CISE curriculum. The letters of support from Virginia Tech and North Dakota State University demonstrate that there is enthusiasm for using the proposed materials.
- **Benefits to MTU and Kettering.** A Software Engineering degree program that serves as a national model of excellence will increase the prominence of MTU and the Computer Science department. This will surely result in higher application and admission rates for undergraduate students, one of the primary goals of the University’s Strategic Plan [StrPla]. In addition, the project will increase mutual recognition, appreciation, and collaboration between the Computer Science department and the Technical Communication program. Given the great deal of common interest uncovered already in the preparation of this proposal, further interdisciplinary research efforts are anticipated. Finding and exploiting synergy like this within the University is another Strategic Plan goal. The interdisciplinary nature of the project will also strengthen bonds between MTU and Kettering. Faculty at the two institutions will work together intensively and come to learn about each other’s education and research efforts.

The project will provide opportunities for both graduate and undergraduate students to work with faculty in a multitude of areas, ranging from the production of scholarly presentations and pedagogical designs to programmatic development and conference planning. The workshop and tutorial will draw educators to MTU and strengthen its prominence in CISE education. The high degree of interaction with industrial partners will provoke their interest in the program.

## 2.5 Project plan

### Academic year 2004-2005

- Create an evaluation plan and perform an initial evaluation of students in Software Quality Assurance and Senior Design Project. At this time, none of the proposed materials will be available yet. Evaluation at this time will give us a baseline against which the benefits of the new material can be measured. The evaluation instruments will be created by the PIs.
- Develop the PFEedit tool. This will be done by a CS graduate student, under the supervision of Prof. Wallace. The tool should be ready for use by the end of the academic year.
- Create initial scenario case studies. These will be used the following academic year in Software Quality Assurance. The case studies will be based on real software problems, located at MTU or local businesses. The work will be performed by an RTC graduate student, under the supervision of Prof. Brady and Prof. Johnson.

---

<sup>8</sup> It should be noted that Margolis and Fisher advocate *early* exposure to such experiences in particular. While this project is focused on senior-level courses at MTU and Kettering, it seems likely to stimulate development of related material for the introductory Computer Science curriculum.

- Install PFEdit on CS lab computers; install AsmL on STC lab computers. This work will be done by lab staff, with direction by Prof. Wallace.

#### **Summer 2005**

- Enhance the current ASM primer with AsmL support. All examples found in the current primer will be implemented in AsmL and documented appropriately. This will be done by a CS graduate student, under the supervision of Prof. Wallace.
- Extend the primer with a real case study in software design: the database recovery example [GSW97]. The models will be implemented in AsmL. This work will be performed by Prof. Wallace and Prof. Huggins.
- Prepare scenario case studies for use in Software Quality Assurance, and identify users available for interviews in Senior Design Project. This will be done by Prof. Brady and Prof. Johnson.

#### **Academic year 2005-2006**

- Use and evaluate the new materials in Software Quality Assurance. Students will explore the scenario case studies, create Problem Frame decompositions using PFEdit, and create formal, executable ASM specifications with AsmL.
- Conduct student-led user observation and scenario construction in Senior Design Project. This will be supervised by Prof. Wallace.
- Perform an initial evaluation of Kettering students in the software engineering course. This will be conducted by Prof. Geske, the course instructor.
- Schedule and advertise the Software Communication & Documentation Workshop for summer 2006. A call for participation will be made to experts in requirements elicitation and analysis, usability, technical communication, and formal methods. This work will be done by the PIs and a CS or RTC graduate student.

#### **Summer 2006**

- Given the documentation from the Senior Design Projects of 2005-2006, construct scenario case studies for use in Software Quality Assurance. This will be done by an RTC graduate student, under the supervision of Prof. Brady.
- From the Senior Design Projects of 2005-2006, develop Problem Frame and ASM material for use in Software Quality Assurance. This will be done by a CS graduate student, under the supervision of Prof. Wallace.
- Review the portfolios from the Senior Design Projects of 2005-2006, for evaluation of this project. The review will be done by a committee of CS and RTC graduate students.
- Prepare materials for use at Kettering, including installation of AsmL and PFEdit. This will be done by Prof. Geske, with assistance from Prof. Huggins.
- Hold the Software Communication & Documentation Workshop. Participants' submissions will be collected, with the intent of publishing them as a book. Organization of the workshop will be the responsibility of the PIs, with help from a CS or RTC graduate student.

#### **Academic year 2006-2007**

- Begin use and evaluation of the project materials at Kettering. This will be done by Prof. Geske.
- Continue use and evaluation of the materials at MTU.
- Schedule and advertise the Educators' Tutorial for summer 2007. This work will be done by the PIs and a CS or RTC graduate student.

#### **Summer 2007**

- Construct scenario, Problem Frame and ASM material for Software Quality Assurance, based on the Senior Design Projects of 2006-2007.
- Review the portfolios from the Senior Design Projects of 2006-2007.
- Hold the Educators' Tutorial. The PIs will organize the tutorial, with help from a CS or RTC graduate student.