A STRATEGY FOR IMPROVING CURRICULAR COVERAGE OF PROBLEM SOLVING THROUGH EMPHASIS ON REQUIREMENTS ANALYSIS

Thomas J. Marlowe, Seton Hall University, 400 South Orange Avenue, South Orange, NJ 07079, USA, thomas.marlowe@shu.edu, 973-761-9784
Vassilka Kirova, Alcatel-Lucent, 600-700 Mountain Avenue, Murray Hill, NJ 07974, USA, vassilka.kirova@alcatel-lucent.com, 908-582-0421
Cyril S. Ku, William Paterson University, 300 Pompton Road, Wayne, NJ 07470, USA, kuc@wpunj.edu, 973-720-2649
Mojgan Mohtashami, Advanced Infrastructure Design, 10 Richardson Lane, Hightstown, NJ 08520, USA, mojgan@aidpe.com, 609-371-5288

ABSTRACT

Approaches for teaching problem-solving, such as the case study method, across a variety of disciplines typically begin with a well-formulated if perhaps incomplete problem. These approaches can be complemented by teaching requirements analysis techniques, largely developed and refined for systems analysis and software engineering, which take a partial problem description and seek a full, consistent and unambiguous problem definition. A novel approach is presented, integrating the methods taught and employed in multiple courses across the curriculum, which can provide students with the additional flexibility and critical thinking ability needed to tackle complex problems, and offer clear benefits for current and future academic programs and potential employers.

Keywords: requirements analysis, problem solving, critical thinking.

INTRODUCTION

In modern practice, students in technical, problem-solving disciplines currently learn both critical thinking and career-related skills by solving well-formulated problems, though perhaps designed with missing or incomplete information, for which they are expected to ask the instructor, use standard resources, or follow specified guidelines, or do some limited independent investigation, specifically in graduate level courses.

In many computer science classes, for example, programming assignments require specified input/output (and error) interfaces and behavior, and/or use of a particular program skeleton or data structure, which together sufficiently define the assignment, although these may be specified at a high level of abstraction (Felleisen et al, 2013). Likewise, in management science or engineering, the case study method (Ellet, 2007) presents a problem definition, often together with a set of alternatives or questions, and requires a recommended solution, with a justification. And similarly, in research methods in social science and related areas, a research topic may be presented, often in terms of a real-world description of a problem, from which students need to
extract a set of variables and hypotheses, possibly followed by designing a survey instrument and analysis, and presentation of results and conclusions.

But in the real world, people are often presented with problem descriptions that are incomplete and even partially erroneous (in addition to including extraneous, redundant or irrelevant information). In a client-contractor situation (as discussed in software engineering or systems analysis courses (Kendall & Kendall, 2010; Summerville, 2010)), this can arise because of the client’s misunderstanding of the problem, the level of the contractor’s domain knowledge, the nature of the contractor’s discipline, or the background knowledge necessary to solve the problem. It can also be a problem of communication, due perhaps to the use of “technical jargon” or localized domain dialects, to complex and not widely-accepted representations, or to overloaded or ambiguous terminology. Finally, it might be a failure (on either side) to consider unlikely or infrequent “corner cases”, or an imposition of infeasible or even contradictory objectives. For example, failure to address “rainy day” scenarios is a common problem with outsourcing or using a new contractor or development teams with no deep understanding of, or limited experience with the domain at hand (Cockburn, 2000). (There are also enterprise-level business, cultural and legal issues, but we do not consider these here.)

These problems are addressed by Requirements Analysis (Kendall & Kendall, 2010; Laplante, 2009; Larman, 2004; Summerville, 2010; Whitten & Bentley, 2007; Wikipedia-3, 2013). This is typically covered late in undergraduate majors, often in a single course such as Systems Analysis or Software Engineering. It is also addressed, but as a secondary focus, in courses such as an upper-level Systems or Network Administration course for selecting new systems, or in Community or Library Services Planning courses, which are widely ranging and challenging to begin with. Techniques and good practices for requirements elicitation, negotiation, and representation in support of business and requirements analysis often get more systematic treatment, but only in graduate courses intended specifically for information science and technology majors (Calefato et al, 2012; Pressman, 2010; Summerville, 2010).

From experience, and from discussions with managers and faculty in subsequent courses, this does not seem to be enough. A single or limited exposure to requirements elicitation and analysis, after several years of problem solving without that emphasis, does not change most students’ analytical capabilities, patterns of thought, or capacity for dealing with real-world under-defined problems. We suggest that students should be repeatedly challenged with the need to define problems, and learn not to limit themselves to the problem description provided by an authority. They should learn that a problem description might be incomplete, or erroneous, ambiguous, or badly explained or suffering from overloaded terminology or unneeded jargon. These experiences should be integrated with consideration of real or realistic scenarios, such as case studies; with experience with analysis, design and/or implementation, as appropriate for discipline and course level; and with strategies for testing, review, revision and reflection into a single unified approach.

This combined approach will benefit all parties: not only the student, but also current (and future) academic programs and prospective employers and managers, and even society as a whole. The student will gain independence and flexibility, critical thinking and problem solving
ability, and communication skills, leading to improved performance in future courses, interviews and careers. It also appears that working with this unified approach facilitates a student’s sense of future directions and possibilities for extensions for the application. As a result, an academic program will benefit from a population of better prepared and more confident students who are not afraid to ask questions or to take assignments to a higher level.

Importantly, this preparation can substantially increase the value of early internships to both students and managers, with students more attuned to context and discussions, and able to better fulfill the responsibilities they have been assigned. This will in turn again improve academic programs, since those students should have a richer fund of real-world practical context and experience on which to draw.

As importantly, employers and managers benefit from both interns and employees who will be up-to-speed more quickly, learning more rapidly and more thoroughly, and with improved initiative, communication skills and critical thinking ability. These characteristics directly counter the most frequent manager complaints: underprepared employees who require a lot of hand-holding and training, poor communication skills, and lack of flexibility and critical thinking.

According to the president of a small engineering firm in New Jersey, small companies cannot afford to hire or retain such underprepared employees. With minimal company staff and tightly constrained resources, a small company cannot afford to invest the money or the staff time to develop those capabilities, and to provide precise, comprehensive instructions for every project. While most employers do have an employee training and development plan, these typically need to be devoted to acquisition of specialized domain knowledge, adapting to changing tools and technology, achieving certifications, or advanced education for a few key personnel. In contrast, under this approach, the employer can presume that a qualified individual with this background and the habits of thought it inculcates will be more readily able to learn new ideas, and take their training and responsibilities to the next level. Of course, just presenting this material will not guarantee these results in every case. What a student takes out of this approach will in large measure depend on the student’s preparation, skills and receptivity, and on the quality of instruction.

**PROPOSED FRAMEWORK: INNOVATING THE CURRICULUM WITH RENEWED FOCUS ON PROBLEM ANALYSIS**

We propose that students in an undergraduate program in a technical, problem-solving discipline have a minimum of three course experiences with requirements analysis, and that, in later courses, the question of adequacy of project documentation at least be raised. Each phase will layer upon the concerns addressed in previous phases, while addressing earlier concerns in a deeper, more sophisticated or less obvious way. Each phase may consist of multiple class discussions or assignments, extending or reinforcing prior assignments; at least one assignment in each phase should be a team project. As appropriate to the project, discipline/course and student backgrounds, different students may be assigned different roles and perspectives (for example, manager, domain expert, standards, safety, customer perspective). This would allow for
individual exploration, followed by brainstorming and prioritization. And while such projects and role-based learning are a common feature of software engineering and systems analysis courses (both undergraduate and graduate), we perceive the need and benefit of this approach being more generally, iteratively and systematically used. The proposed framework uses elements of the classical “case study method” (Ellet, 2007), as commonly used in engineering and business schools. This method, broadly based on original “case method” (Case Method, 2013; Hammond, 1976), can provide a bridge to applying requirements analysis techniques over a much wider range of disciplines.

Phase I: Determine whether a problem description is incomplete, inconsistent, or ambiguous, and whether any terminology might be overloaded or unclear. Provide and document a more complete, more correct problem definition, by identifying objectives, formulating proper questions, and determining and consulting potential sources of information (including the instructor acting as the client). A simple example is discussed in the next section.

Phase II: Identify stakeholders and interests. Use this to determine additional information, constraints or results needed. Formulate the problem definition in a form conducive to design and implementation of the solution. This analysis may, for example, reveal the need for privacy and security, for disability access, or for auditing functions. It may also consider what aspects of the solution are likely to be faulty or where requirements are likely to be changed (the “Principle of Protected Variation” (Larman, 2004)).

In some cases, the result may be a recommendation to consult, hire, or subcontract a specialist or expert in a particular domain, discipline, or problem dimension, such as community outreach or data visualization. In this phase and the next, problems should include (often hypothetical) projects with a known client and projects in which a product or proposal is developed “on spec”, to meet a perceived marketplace, economic, technical, or social need or want.

Phase III: Focus and expand on issues of risks and risk management, domain and discipline knowledge and knowledge management, and the proposed dissemination of results. Consider more fully how requirements, constraints and objectives will be verified or tested. Prioritize objectives and functionality, and identify tradeoffs where possible. Where possible, consider likely future trends, and identify where the solution needs to be modified to support future evolution.

As far as we are aware, programs with multiple exposures to requirements analysis techniques are at best rare, and are not specifically structured as a sequence of experiences. In addition, recurring coverage of requirements and this unified approach also supports consideration of such multiple formats and scenarios.

In computer science, Phase I could occur in a freshman or sophomore data structures course, Phase II in a sophomore/junior database course, and Phase III in junior/senior software engineering. The demands of all phases will increase in progression and often in parallel. In this example program, Phase I would be accompanied by coding of classes and algorithms; Phase II by design, implementation, and testing of a database and set of queries, plus user forms and
reports; and Phase III by implementation of a relatively complex software application, together with reflection on “lessons learned”. The latter phases could be accompanied, for example, by assignments to extract from a problem description recommendations for non-functional issues such as policies and controls for access, security, or configurability.

In management science or decision science, these phases would typically occur in courses that use case studies (or statistical modeling), such as Quantitative Modeling, Decision Analysis, or Management of Operations. Phase III could be introduced and all phases reinforced in a senior-level systems analysis course or seminar.

There may also be an historical or legacy systems dimension to the analysis, whose importance may vary with discipline and problem. Prior related projects, both successes and failures, can reveal requirements dimensions and constraints that may not have been sufficiently considered, such as communication infrastructure, interfaces to and interaction with existing system components, or coordination with partners and subcontractors.

It will be more difficult to impose a sequential structure in programs without prerequisite structure, and in most graduate programs, especially those in which many students will come from disciplines without an emphasis on problem solving or modeling, such as Library Science or Urban Planning. In such programs, requirements analysis should still be revisited in multiple courses, but in the absence of a default sequence for courses, the three phases will need to be covered, as far as possible, in each course.

**EXPERIENCE AND QUALITATIVE RESULTS**

The authors and their research colleagues have had success with assignments in which students are given an incomplete problem specification. In some cases, students are asked if they can develop a solution from the information given, and after (as usually happens) they agree, are confronted with a series of questions. Alternatively, students are asked to develop a solution, and after they run into problems (or submit an unsatisfactory program), get a review of requirements analysis.

A research colleague has used a Phase I exercise in undergraduate Data Structures (Marlowe et al, 2013). The problem description is “Read in a list of positive integers I will provide, and sort them in ascending order. The list is read from the keyboard and the result displayed to the screen.” Students are then asked, “Do you understand enough to write the program?” After the students agree, examples are presented with invalid input (negative numbers, floating point numbers, characters, and so on) and duplicates. It also becomes clear to the students that there was no agreed way for the user to end the list, and no understanding of how to handle display of the empty list.

Marlowe has used exercises of both kinds corresponding to Phase I, with elements of Phase II, in undergraduate Database and Software Engineering classes, and gives a written risk analysis assignment related to Phase III, but with substantially more guidance, in Software Engineering; and Ku has also used an exercise mixing Phase I and Phase II in his Software Engineering class.
Kirova has used an exercise blending elements of all three phases in a graduate system analysis and design class, finding that many students remain unaware of, and inattentive to, the serious implications of vaguely defined and incomplete problem descriptions as undergraduates were—indicating that a greater and repeated emphasis is needed in the undergraduate curriculum. Finally, another colleague, in an interdisciplinary community planning/sustainability program, is using a blend of requirements analysis, case studies, application and reflection, and finds that students on the whole are comfortable with case studies and application of knowledge, but much less so with requirements and reflection (Hall et al, 2009; Marlowe et al, 2013; Nousala, et al, 2009).

The importance of requirements as a complement to problem solving across the range of problem-solving disciplines has been endorsed by Matthias Felleisen (Felleisen, 2013), a distinguished computer scientist and educator, and a strong advocate of the similarity of analysis and design methods across those disciplines. We have also conducted a number of interviews and discussions with professionals across a wide range of disciplines, including several librarians, medical professionals, science educators, a number of engineers, a software development entrepreneur, executives at a small engineering firm, and a manager at a large software development firm. All have agreed that this is a valuable addition to preparation in their fields, with some wishing that it had been included in their own curriculum. Managers also comment that many of the problems we have identified recur in their new hires, and in many cases are difficult for these individuals to overcome in a workplace setting.

STUDY PLAN

The authors and their colleagues plan to continue using such exercises, extending from Phase I to other phases, in undergraduate and graduate courses in computer science at several academic institutions, including William Paterson University, Seton Hall University, New Jersey Institute of Technology, and Montclair State University. We are also seeking partners in other disciplines who will include similar, discipline-relevant exercises in their own courses. Ku, with the assistance of departmental colleagues, will seek to implement the full three-phase approach in the undergraduate computer science major at William Paterson University.

In each case, we plan to provide students with questionnaires at the start and the end of each course. The initial questionnaire will be designed to measure students’ perceived views of their ability to read problems carefully, to work on complex problems independently or in teams, and to find missing segments of an incomplete problem description and resolve them. The final questionnaire, in addition to asking the same questions, will also ask students to assess changes in their perceptions and abilities, and on whether they feel that the attention to requirements in the course has improved their learning experience, their skills, or their confidence with the discipline (e.g., software development).

Where the three-phase approach is used, we would also plan to ask students about to graduate to evaluate the contributions of the requirements emphasis made to their education. In the long run, we will also seek reports from alumni, initially a year or two after graduation, and also five years after, on the effect this concentration had on seeking and succeeding in employment, or in
graduate work in their original discipline or in other areas. Finally, we hope to place students in internships with managers aware of this project, and survey those managers to determine if this preparation has had an effect on intern performance.

**DISCUSSION AND RELATED WORK**

The individual aspects of this approach are well-known. Requirements analysis is covered in most works in software engineering and for systems analysis, such as those cited above, as is testing and review, and teaching case studies in (Elle, 2007; Hammond, 1976; Herreid, 2005; Introduction to the case study method, 2013; Wikipedia-1, 2013). Problem solving in various contexts is discussed in many works, including (Ackoff, 1978; Hayes, 1981; Wickelgren, 1974) and the classic (Polya, 1957). On the other hand, techniques for analysis, design, implementation or solution, and evaluation and testing are very much varied and discipline-specific.

We compare the requirements analysis approach outlined above to the case-study method in particular. Requirements analysis is both similar to, and complementary to, the usual approach to the case-study method. On the one hand, each begins with a problem description, and each must deal with the possibility of missing or incomplete information. On the other, requirements analysis aims at extracting a consistent problem definition, where a case study asks for a solution or a recommendation. For that reason, discarding extraneous information is desirable in requirements analysis, but may be crucial in a case study.

In the former, incomplete, inconsistent or ambiguous information must be resolved, possibly by repeated contact with the client; in the latter, although seeking information is a (sometimes necessary) option, students are encouraged to make assumptions, and solve the problem conditional on the correctness of that information. In more sophisticated situations, the recommendation in a case study can be made conditional on information that may no longer be available—“If the ground is not saturated at the time construction begins, …”

The two approaches are of course complementary. Analysis of a complex problem will often begin in the real world with only a business vision and possibly a product case, which is then elaborated by initial requirements processes (elicitation/acquisition) into an incomplete problem description, which is then elaborated into a full and complete, coherent problem definition (subject to the usual real-world uncertainties), which can then be investigated, or designed and implemented, to provide an acceptable solution. (Actually, a problem definition does not have to be completed to allow progress on its solution, but does need a correct and consistent core definition, and must include any elements that must be addressed throughout the solution, such as software security. Modern agile techniques (Larman, 2004; Martin, 2002) in software development and elsewhere begin with such a definition, adding features and addressing unexpected or deferred issues as they arise. This is responsive to near ubiquitous time and budget constraints, and affords early discovery of flaws in requirements, design and implementation—it being a truism of software engineering that the cost of fixing a bug rises exponentially as development phases are completed (Pressman, 2010).)
The two approaches are also complementary in another sense. After the initial exposure to requirements, and in particular in later exercises in the same course, requirements problems can be made more interesting and immediate if stated as a case study, with problem definition as the required solution.

The “case study method” is also used for a related technique in research (The Case Study, 2013; Gill, 2011; Kohn, 1997), possibly quantitative but more usually either qualitative or a mix. As a method for qualitative research, the case study method also includes determining and defining the research questions, selecting cases and specifying data collection approaches, which in turn requires survey design. While these additional activities have more of a flavor of requirements analysis, they typically “emphasize detailed contextual analysis of a [well-understood] limited number of events or conditions and their relationships” (The Case Study, 2013), and develop a “logic model” which connects conditions or inferences rather than actions to be tested (Kohn, 1997).

Statistical modeling, particularly multivariate statistical analysis, for that matter, has elements of requirements analysis in attribute and variable selection (Wikipedia-2, 2013), as it seeks to define appropriate stimulus, response, intermediate and control variables, plus confounders and distractors, and to eliminate redundancy in the variables measured. But by comparison it also has more of a “closed-world” flavor, in which the answers (attributes and variables) already live in a universe of discourse, while requirements analysis, used broadly, begins by looking for that universe and its limits. In summary, one can think of contextual analysis as finding a path to explore on a map, while requirements analysis also looks to develop or refine the map.

There are also connections to theories and reform movements in education, in which stress student acquisition of deep understanding, critical perspective, and long-term learning (Lemke, 2013). The four element approach (requirements, case studies, analysis and development, reflection) is in most disciplines limited to the cognitive domain (rather than affective or psychomotor), but requires all six elements of the revised Bloom taxonomy (Clark, 2013; Forehand, 2013): remembering, understanding, applying, analyzing, evaluating and creating, and stresses the higher-level skills. Considering Gardner’s multiple intelligence theory (Wikipedia-4, 2013), the approach needs and combines logical-mathematical, linguistic and interpersonal intelligence, the last with both intramural (teams and enterprise) and extramural (client, customer, subcontractors, society) focus. Some problems will also call for kinesthetic or naturalistic intelligence, and reflection may in addition have an intrapersonal dimension.

The process, specialized to a student’s discipline and interests, supports these objectives (other than the philosophical and perhaps social dimensions of (Lemke, 2013)), adding depth, exposure to real-world scenarios, consideration of multiple perspectives and interests, fosters lifelong learning and critical thinking, and can be designed to encourage interaction with a wide range of information sources and communities of interest and of practice. Where interdisciplinary seminars and capstone courses exist, students will acquire a richness of understanding through interaction with these elements.
Finally, it is important for students (and instructors) to realize that, while requirements analysis is an important—often critical—initial activity in analyzing and solving problems, it is equally important for them to realize its limitations. First, requirements analysis of complex problems will rarely if ever be perfect. In particular, initial analysis will need to be complemented by ongoing, periodic review of requirements, together with review, testing, and evaluation of the results of analysis. New information emerges, the environment (market, technical, regulatory, …) changes, prototypes or proposed solutions reveal invalid or incomplete assumptions or additional objectives, or client/customer feedback results in revised or additional demands. Second, cost-benefit analysis and scheduling apply to requirements analysis as well as other problem-solving activities. One can spend unbounded amounts of money, time, and personnel hours on requirements, without ever making progress on addressing the problem—one of the problems with the classical approach to software manufacturing (Pressman, 2010). Thus interleaved requirements and contextual analysis/solution activities, and prioritization of objectives, are issues that should also be addressed repeatedly.

CONCLUSIONS

We have argued that further stress on requirements analysis in computer science, management science and other programs in problem-solving disciplines increases the value of those programs, leading to benefits for students, their faculty and academic programs, and their employers and managers. Integrating requirements analysis with other analytical techniques, application development, and/or the case study method, as appropriate to the academic discipline, provides a synthesis that increases the benefits of each component. For requirements analysis, we propose a novel three-phase approach wherein students are exposed to more and more complicated and sophisticated requirements issues across a sequence of at least three courses. This will be integrated as appropriate with case studies, discipline-appropriate solution activities (such as analysis, design or implementation), and reflection, and a sense of the inherently incremental and prioritized nature of problem-solving strategies and activities.

ACKNOWLEDGEMENTS

We would like to thank Dr. Susu Nousala, and our interviewees, especially Kathy Malanga of William Paterson University, Marta Deyrup, Mark Couch and Martha Loesch of Seton Hall University, Garett Chang, president of Highstep Technologies, and Przemek Bogdanowicz of National Louis University, whose comments have aided in strengthening this paper.

REFERENCES


671876-10


