Instilling a Software Engineering Mindset through Freshman Seminar

Michael J. Lutz, James R. Vallino, Kenn Martinez, Daniel E. Krutz
Department of Software Engineering
Rochester Institute of Technology
Rochester, NY, USA

***Abstract*—Student retention is a challenge faced by all engineering programs. Our first year software engineering students have schedules filled with computer science, mathematics, science and humanities. The lack of any exposure to engineering meant some students, expressing a dislike for software engineering, left the program before they had any exposure to the discipline.**

To address this issue, we created a one credit Software Engineering Freshman Seminar, which all entering students take in their first term at RIT. This lets us insure student/faculty contact early in the program, as well as providing an opportunity to introduce engineering concepts and practices early in each student's program of study.

This paper discusses the seminar's current incarnation. In particular, we focus on those aspects of the course which help students identify with software engineering as a profession. The challenge we face is achieving this goal with students whose technical knowledge and skills are modest. We have settled on an approach that provides experience with teamwork, requirements elicitation, and the effects of change, and addressing professional ethics. These in-class activities are complemented by an assignment to interview a practicing software engineer and to write an interview summary for discussion.

This activity ensemble serves to disabuse students of the notion that software engineering is little more than programming, or that the discipline is identical to computer science. Should a student exit the program at this point, at least he or she knows a bit about what they are leaving behind.

Keywords-first-year seminar; software engineering; teamwork

I. INTRODUCTION

In 1996, the Rochester Institute of Technology (RIT) launched the first undergraduate software engineering program in the United States [1][2]. From an initial class of 15, the undergraduate program has expanded to a current enrollment of approximately 375 students. While the Department of Software Engineering has grown to encompass a masters program, the undergraduate program remains the focal point of the department's identity.

Engineering programs seek to prepare engineers who can define, design, develop and deploy useful, cost-effective, and maintainable systems; this is no less true of software engineering than of more traditional disciplines. Traditional engineering builds upon (but is distinct from) natural sciences such as physics and chemistry. Similarly, in our view software engineering has its foundations in computer science, but it is no more the case that computer science encompasses software engineering than it is that chemistry encompasses chemical engineering.

The proof of our philosophy is in its results. First, software engineering is recognized as an engineering discipline by ABET, and the first graduates of an ABET-accredited baccalaureate software engineering program came from RIT. In addition, software engineering students have great success both on co-op and after graduation. Across the broad range of undergraduate computing programs at RIT, our students have the highest median co-op wages and the highest median salary upon graduation. Students and graduates work for firms large and small, and in domains spanning embedded systems, as at Goodrich Aerospace and Harris RF Communications, through end-user focused firms such as Microsoft, Apple and Google. All in all, our program provides a solid foundation for entry into and continual growth within the world of professional software development.

II. THE FIRST YEAR CHALLENGE

While our program is successful overall, we face the challenge of instilling a sense of engineering practice and professionalism, along with the distinctive perspective of software engineering, in our first year students. The challenge is made more difficult by the fact that entering students are often confused as to where software engineering fits within the larger framework of computational studies. Software engineering's recent emergence as a discipline distinct from computer science, combined with a first year program of study that is heavy in math, natural science, computer science, and the liberal arts, serves to exacerbate this confusion. In the end, first year students who leave software engineering often did so as a result of this confusion – they had little knowledge of or appreciation for the distinctive nature of software engineering.

To address the challenges and confusion outlined above, our program includes a one credit hour course providing a broad perspective on the discipline for entering students; over time, this Software Engineering Freshman Seminar[3] has evolved into the one we present in this paper. While tuning and tweaking is regularly taking place, the general structure, topics, and flow of material has stabilized. As a consequence, students completing the course have a grasp of some key software engineering concepts and practices, and can appreciate the role of their foundation studies as preparation for the software engineering courses that follow.

III. Course Goals

The overriding course goal is reducing first year attrition in our program. We know that attrition after the second year is less than 10%, especially after students complete our Engineering of Software Subsystems course, where software design principles and patterns are first discussed in detail. We decided a bridge was needed to see students through the first year of the program, where math, science and liberal arts courses predominate, into the second year where software engineering *per se* becomes the curriculum's focus. Our hope was to forestall student departures because "I don't like software engineering" or "I want to do something besides just programming." Certainly students should be free to pursue interests elsewhere, but we'd prefer they do so based on an accurate perception of the discipline. Misperceptions, whatever their cause, should be eliminated.

Our bridge comprises two courses, the seminar described in this paper and a Personal Software Engineering [4] course taken at the beginning of the second year. Both serve to forge bonds between software engineering majors and the faculty in the department, and to provide distinctive engineering experiences in conjunction with foundational studies. In this our program is in line with many similar efforts in other engineering programs [5][6][7].

For the seminar, we decided to focus on three elements of software engineering practice that undergird our program and that can be imparted to students with modest technical background: product requirements and design, teamwork, and professional communications. We also saw this as an opportunity to introduce students to ethics and ethical professional behavior. Activities throughout the course illustrate and reinforce these concepts.

With one exception, requirements and design activities do not involve software. In part this is a concession to the fact that many students are software development naïf's, and requiring them to do development in both the seminar and introductory computer science would create an unreasonable load. As important, we wanted students to realize that many engineering problems they will encounter (as well as their resolution) require more than the creation of executable code. Thus the requirements and design activities involve paper mockups and Lego based exercises rather than executable programs. Even the one exception, requiring teams to provide enhancements to Java classes, is small enough so as not to obscure the concepts being taught.

Our program, in contrast to many computing programs, is heavily team-based. Indeed, with only two exceptions, all courses in the program have at least one (and usually several) team-based projects. In light of this, an early exposure to both the benefits and costs of working in teams gives first year students a leg-up on what they will be doing later on in their studies. It also serves to counter the "lone hacker in a cubicle" perception of software development so prevalent in the culture at large. In any event, by the end of the seminar most students are comfortable working on teams, solving problems larger than they could address on their own.

The ability to clearly and concisely communicate ideas is as critical to a software engineer's career as his or her technical skills. Indeed, one impetus for developing the curriculum was recognition of the divergence between the preparation of graduates from previous computing programs and the needs of industrial software development firms; poor communication skills were at the core of the problem. Thus our courses require significant written documentation and frequent oral presentations. It is natural to reflect these demands in the seminar course, though at somewhat reduced formality. Once again, such activities serve to distinguish software engineering from other computing disciplines.

Finally, as we are educating future professional engineers, it is incumbent on us to reinforce the professional responsibilities and ethical demands of the discipline; there is no reason why this education cannot begin at the outset of each student's studies. Of course professional ethics is an area that is notoriously difficult (and dull) to teach via lecture; such approaches often come off as special pleading. In our approach, we try to balance the need for teaching specific ethical principles with experiences in which students explore the ethical ramifications of specific technical decisions. Unintended consequences, such as epileptic seizures due to rapidly flashing game screens, provide a rich environment in which to discuss professionalism.

To provide this view of the breadth of software engineering, we defined the following learning outcomes for the Software Engineering Freshman Seminar.

A student will be able to:

1. Identify the principles of the Software Engineering ethics (e.g. Code of Ethics as recommended by the IEEE Computer Society and ACM)
2. Identify the major activities of Software Engineering
3. Identify strategies to address issues that can arise in a team project.
4. Identify the difference between Software Engineering and other computing disciplines.
5. Apply general concepts to a specific process step, namely execute a project test plan and, acting as the quality assurance group, assess the effectiveness of the test plan for the development team.
6. Explore and describe the responsibilities, working environment, skills and technologies of a software engineering professional.

With this background information on our course goals, it is appropriate to turn next to the specific topics and activities – the tactics, if you will – by which we strive to achieve these goals.

IV. Topics and Activities

The seminar is heavily oriented towards active learning, as we've found that this engages students in the material for each session. This engagement leads to insight and interest in software engineering, with the consequence that students acquire a broad perspective on the discipline.

Given the team-based nature of the curriculum, it is no surprise that most of the work in the seminar requires working in teams of 3-5 students; to help students become acquainted, we usually vary the teams from one exercise to another. While most exercises are completed during class, a few span multiple meetings, and require the team to meet outside the class period. Once again, this reflects a situation they will encounter throughout their software engineering studies.

One issue we frequently have to address is the lack of (graded) group work in high school. Even in situations where collaboration is encouraged, at the end of the day most assignments entering students are familiar with had to be completed individually. To counter this mindset, we purposefully focus on the degree of collaboration within teams, and downplay individual assessments based on project outcomes. It is true, of course, that peers and instructors evaluate individual contributions when grading projects in later team-based courses, but in the seminar it is more important to have students feel comfortable being assessed as a group. We seek to encourage open, collaborative approaches to problem solving as the norm across all the exercises.

The learning outcomes from the previous section are addressed across the 10-week course, with the emphasis varying among the activities pursued in any given week. Initially the focus is on perceptions of the software engineering discipline, followed by activities related to various phases in a software development process (e.g., lifecycle activities), capped by interview sessions with students returning from co-op and a sample of software engineers from local industry. For convenience, we classify activities as non-software (applicable in any engineering context), software (focusing on issues of particular importance in software engineering), and the "real world" (via interaction with practicing professionals). The following subsections expand on activities in each of these classes.

A. Non-Software Activities

1) Planning and Team Collaboration: In this activity, while we stress the importance of team collaboration and communication, we also want the students to recognize the importance of careful planning. Such planning can mitigate the unexpected consequences that often prove costly at later stages of development.

The stated objective is the construction of the tallest Lego tower, using the kit provided. Often teams immediately begin assembly, following the first idea from an outspoken team member, and paying little attention to the future. As the towers grow, a rule change is introduced: Team members are only allowed to use their left hands. Students quickly realize that they can longer function autonomously; rules and protocols are required for the team's work to continue.

Finally, the requirements are changed - each team is given a set of wheels and told their tower must move across a table prior to its height being measured. Naturally, those with flimsy, unstable towers must rush to reinforce their constructions - this is a valuable lesson on premature optimization in the face of continuing changes.

At the conclusion of this activity, all students observe the performance of each tower - whether it passes the acceptance test and only then whether it is the tallest. We often hear comments such as “we did not think of doing it that way” or “we probably should have spent more time planning what we were going to do”. Once the tests are completed we have a class discussion in which the students are asked to reflect on the activity as it relates to software products with which they are familiar.

2) Disruptive Teamwork: In this activity, teams of 4-5 students are set to the task of preparing a short presentation on a software engineering topic. Unbeknownst to most of them, one member of teach team is a "mole," selected by the instructor. The moles are pulled aside on some pretext, and told to be disruptive during their team's meeting. Typically instructors give a variety of disruptive roles from which to choose. The goal is to introduce common team problems and see how the team as a whole reacts.

We are continually impressed by the thespian abilities of our moles and the energy they put into their roles. In the past, we've had moles who demand to be in charge of the team, who tried to get their colleagues to watch YouTube videos with them, who challenged each and every idea other team members proposed, and who even entered the team space, put their head on the table, and went to sleep. As one might expect, by the end of the 30 minute team session the other team members are annoyed (to put it mildly).

At the conclusion of the exercise, the teams reassemble in their classroom. The instructor then exposes the mole on each team, as well as the particular disruptive behavior each mole was assigned. Groups then work on the *real* presentation: How they handled their "problem" member and what they might do in a similar situation in the future. Presentations are followed by a class-wide discussion of the advantages and disadvantages offered by teams.

3) Professional and Ethical Responsibilities: The session on professional and ethical issues is preceded by an out-of-class assignment to read and comment upon the Software Engineering Code of Ethics developed by the ACM and the IEEE Computer Society [8]. In the class session, this is expanded upon via a short lecture on related topics, including health and safety, moral and government guidelines, and ethical behavior in the workplace.

Students then form teams and develop a short skit on one of the specific topics covered in the Code of Ethics or in class. Students generally have fun with this assignment and their skits usually illustrate their topic effectively (and often humorously). Following each skit, a focused discussion takes place related to the skit's topic. In particular, students are asked to reflect on real world situations related to the topic and how they might deal with the resulting situations.

B. Software Activities

1) Defining and Describing Software Engineering: One of the earliest activities centers on the meaning of software engineering as a discipline. Students are asked to work in teams to create a short presentation or brochure to be delivered to their former high school. Teams are given an example presentation (one used during open houses, and familiar to many of the students), along with an outline of topics to consider for inclusion. The topics include the differences between software engineering and other computing disciplines, the relationship between software and our rapidly changing world, and the range of career opportunities within the field.

We know that students are largely ignorant of the ethical issues involved in plagiarism [9]. The presentation provides an excellent venue, early in their studies, to present these issues, along with advice on how to avoid plagiarism. As part of the activity, students are required to provide proper citations, and we also offer guidance on abiding by rules of ethical academic behavior.

Given the proliferation of internet resources of widely varying quality, we also introduce students to ways of identifying credible, trustworthy, and informative sources. Key points include verifying the original source of a work, determining the work's publisher, and finding the last revision date.

Having spent a week on this activity, several are selected to make a formal presentation. A resulting class discussion on the merits of the content, style and sources increases student appreciation of the software engineering profession's place in the context of an expanding and ever changing technical world.

2) Challenges of Requirements Elicitation: At the midpoint of the term, students participate in a second Lego project, this one to build a house to a customer's specification. The goal is to expose students to the difficulties of both eliciting and conforming to customer expectations.

At the outset, each team is provided with the basic requirement that the team must build a Lego house satisfying a customer's requirements. Instructors and course assistants prepare by identifying a simple set of requirements such as a minimum of two rooms, a window in each room, a roof, etc. Elicitation spans three iterations; during an iteration, each team has the opportunity to ask three specific questions (queries like "what do you want the house to look like?" receive short, direct and ambiguous replies).

Teams soon realize that they must carefully consider which questions to ask and how to frame them in order to maximize the useful information received. Course assistants enjoy acting as customers during this activity, and revel in truthfully answering questions so as to reveal as little information as possible. Question: "Does the house have windows?" Answer: "Yes, the house has windows."

At the end of the 30 minutes devoted to the activity, each team shows the house they built and how it meets the requirements as they understood them. Normally, no two houses are even vaguely similar, reinforcing the problems of obtaining accurate and useful information from customers. Students see firsthand how requirements can be interpreted in radically different ways. Overall, the exercise reinforces the importance of maintaining a continuing conversation with the customer, while working to ensure the real requirements are understood and the customer's desires are satisfied.

3) Software Process Methodology: We introduce the notion of a development process via a version of the Extreme Programming (XP) game, using a variant of Joe Bergin's coffee machine planning game[10]. XP is used because it allows teams to make headway in the face of changing requirements; in particular, it emphasizes evolutionary development with small, incremental releases.

Each student team designs a vending machine on paper, where the machine must conform to prioritized, predefined user stories. Students in each team assume one of three distinct roles: customer, developer, or monitor. Customers establish machine feature priorities using their own opinions combined with an estimate from developers as to the effort needed to include the feature. Developers add features to the vending machine from most to least important; in the second and later iterations, this includes integration and refactoring of what was done previously. Moderators are part coach, part referee, monitoring communications between customers and developers, and ensuring the process stages are properly time boxed.

At the conclusion of the exercise, student teams compare the resulting systems, recognizing that they all started with the same user story set. The class ends with discussion of the process, what went well and what caused problems, and the effect of the different roles on estimation and prioritization.

4) Team Design and Implementation: The longest activity, spanning two weeks, is the Robocode[11] project. Robocode itself provides a framework for simulated battles between programmable, robotic tanks. In addition to the battlefield, automated scoring, and various graphic and sound effects, Robocode provides an API for creating new tanks.

In this activity, we introduce pair-programming, and have pairs of students develop their own unique robot from a skeleton we provide. In the first class, each team sketches the behavior they want to implement, and then works on its tank for the remaining time. Near the end of class, all the teams' tanks (as well as a few from the Robocode library) are placed in the arena and the battle begins. After all the flashes, sounds, and mayhem subside, students see where their robot ranks.

Between class sessions, and at the start of the second week, each pair hones its robot based on the first week's results. At the end of the second week, another section-wide battle is waged, with each section's winning robot submitted to the final battle royale at the end of the term. The team that emerges victorious from the final battle has its robot memorialized by a small trophy with a toy tank on top.

The goal of this activity is less about improving student programming skills than it is about working in pairs to develop a software system over several iterations. In addition, planning an adaptive strategy to exploit other tanks' weaknesses reinforces the need for a well-considered and flexible design.

5) Cross-Team Testing: Near the end of the term, teams of students from the seminar pair with teams from our second year Introduction to Software Engineering course in a cross-team testing exercise. Each seminar team acts as an independent test team, performing acceptance tests from a test plan prepared by their paired second year development team. We encourage the test teams to take initiative, and expand testing of functionality and usability beyond the boundaries of the test plan.

Both the first year and second year students gain valuable lessons from this exercise. The first year students gain testing experience, and see for themselves the significance of usability design and the importance of validation. In particular, the acceptance test drives home the connection between requirements and black box testing; students realize effective testing does not require access to the source code. The second year teams, in addition to the obvious benefit of having their test plan exercised, see actual users struggle with or delight in the systems they produce.

C. Exposure to the Real World of Software Engineering

Near the end of the term, we turn our attention to software engineering in the world outside of RIT. The primary vehicles for this are a panel discussion with upper-division students and local software developers, as well as a paper summarizing an in-depth interview with a software engineer in industry. These activities are designed to expose students to the real life activities of software professionals, to foster increased interest in and curiosity about the profession, and to clarify any remaining misconceptions regarding work in the field.

As preparation for the panel session, each student submits a set of questions they would like answered. They know that the panel will consist of upper-division students who have completed several co-op blocks, as well as practicing professionals, many of whom are alumni of the program. As a consequence, the questions range from the mundane ("how do I find housing while on co-op?") to the profound ("what non-technical skills do you find most useful?"). On the whole, our experience has been that students appreciate the opportunity to learn from those with experience, and as a result have a better idea as to whether or not software engineering is the career for them.

The interview paper requires students to find a software developer to interview, to arrange an interview by email or (preferably) over the phone, and to summarize the interview and their analysis of it in a written document. We place a few restrictions on the selection of an interviewee: the person must be engaged in software development or management, may not be a close relative of the student, and may not be selected by more than one student. Occasionally students are unable to find anyone; if the instructor is persuaded that the student made an honest effort, the instructor may tap into his or her professional network for a colleague who will agree to be interviewed.

To overcome student inertia, we provide an initial set of generic questions to help frame the discussion. However, students must supplement what is provided by specific questions of their own. In the best of all worlds, the students have a conversation with the person they select, and such conversations frequently lead to wide ranging discussions that enhance the student's understanding of software engineering.

Each student submits a transcript of the conversation and a document reflecting on what he or she learned. Students often describe their preparation, thoughts, and expectations prior to the interview, along with unexpected discoveries as a result of the interview.

The "final exam" is actually a class-wide, collaborative reflection on the course as a whole and the interviews in particular. Once again, students are divided into groups, where they compare experiences and compile a list of observations as to what they have learned during the term. When the class gets together again, teams share these observations. Occasionally instructors impose some structure on the proceedings, such as having teams create mind maps for the course; other instructors take a more freewheeling approach in the interest of spontaneity. Whatever the approach, the class usually ends on a high note with students able to articulate what makes software engineering different.

V. EVALUATION

Part of our curriculum internal assessment is based on student feedback. The opportunity to provide course-specific feedback is afforded to students at the end of each school term via an online anonymous survey. Table I contains student course evaluation data from the last two years when we ran the version of Software Engineering Freshman Seminar discussed in this paper. The columns range from Strongly Agree (SA) to Strongly Disagree (SD) from left to right.

The feedback received in the course's latest incarnation helps validate both the individual class activities as well as overall student learning. Recurring themes are an appreciation for hands on team activities in a software engineering context, the understanding of how software development benefits from process, and the importance of communication between teams and customers.

TABLE I. STUDENT COURSE EVALUATION DATA

Question	SA	A	N	D	SD
I learned a lot in this course.	17.7	54.4	19.7	7.5	0.7
In general, the out-of-class assignments were relevant to the course.	38.8	49.0	8.8	2.7	0.7
In general, the in-class activities were relevant to the course.	53.1	37.4	5.4	4.1	0.0
Overall, I would recommend this course.	43.7	33.8	16.9	3.5	2.1

Following are representative samples of written feedback we have received:

> *As I understand it, what I just took in this course was already a rework. Keep it!!! This "intro" course to software engineering was fantastic! It was really great, my favorite course by far!*

> *I enjoyed Software Engineering Seminar and I think it made me feel certain that I'm in the right major, and that I have a thorough understanding of*

what Software Engineering is like outside of the classroom.

This course was very informative, and fun. This is a great course and it gives you a great idea about what Software Engineering is like and starts to prepare you for what lies ahead.

We continually struggle with how to impress upon our entering students that software engineering is much broader than the programming they may have done through high school, and even the material they will see in introductory computer science courses. This is a reprise of the concern expressed at the start of the paper that students leave the program before they have seen any engineering. In particular, they need to see that software engineering will not relegate them to a lifetime of low-level coding.

For most of our students, this realization does not really set in until after they have been out on co-op. Our hope is that Software Engineering Freshman Seminar helps students move towards that realization at the outset of their college studies. Based on the following comment, our hope for the course was met, at least for one student:

I felt this interview was very useful. It opened my eyes to how great SE-101 [Software Engineering Freshman Seminar] was. I'll admit I didn't feel like some of the projects were that useful. Until this interview I was a little disappointed we didn't do more coding in class. However, I found that we actually covered the most important topics, like communication, and working with others. It really opened my eyes to what software engineering is really like.

For many students, this first introductory experience reaffirms their desire to pursue a career in software engineering.

VI. Future Evolution

We expect the seminar course to continue evolving in the future. In particular, RIT's impending switch from academic quarters to semesters has forced us to reconsider some of the pedagogy. Our present 10-week course with one two hour meeting per week will expand to a 15 week course with one or two meetings per week. Activities which currently require a full two hours to complete will have to be rewritten or replaced in light of these constraints.

A particular challenge will be process activities, such as the XP game, that require extended time to be effective. One possibility, inspired by field trips in biology and zoology, is to schedule one or two long activities for a weekend, with, of course, sufficient food and refreshments to entice students to participate. The feasibility of such an approach is up for discussion.

We also see a role for expanded coverage of intellectual property issues. Currently we only address these in the context of proper citation of other's work, but much more could be included that is accessible to first year students. Certainly software engineers need be cognizant of the role played and restrictions imposed by copyrights, trade secrets, and patents.

VII. Conclusion

When the term software engineering was first coined, it was mostly an aspiration and a metaphor. Over the past 45 years the term has come to signify a new and exciting engineering discipline. The seminar course we've described is one way in which that excitement can be communicated to the next generation of professional software developers.

Acknowledgments

We would like to acknowledge the many student course assistants who have helped run this course. They have not only engaged students during class time, but also were responsible for creating content for two of the class sessions. Another invaluable service that they provide is to aid our freshmen students in their adjustment to college life. We regularly note interactions between the course assistants and students in the seminar that are more about how to navigate the ins and outs of the RIT campus and systems than how to set the color of the team's Robocode tank turret using Java.

References

[1] M. J. Lutz and J. F. Naveda, "The Road Less Traveled: A Baccalaureate Degree in Software Engineering", *Proc. 28th SIGCSE Technical Symp. on Computer Science Education*, March 1997.

[2] F. Naveda, M. J. Lutz, J. R. Vallino, T. J Reichlmayr, and S. A. Ludi, "The Road We've Traveled: 12 Years of Undergraduate Software Engineering at the Rochester Institute of Technology," *Proc. Int. Conf. on Information Technology: Next Generations*, Las Vegas, NV, April 2009.

[3] K. Martínez, (2011, September 8). *SE 101 Freshman Seminar* [Online]. Available: http://www.se.rit.edu/~se101

[4] M. J. Lutz and T. J. Reichlmayr, "A Course for Developing Personal Software Engineering Competencies", *Proc. ASEE Annual Conf. and Exhibition.*, June, 2012.

[5] S. S. Courter and K. J. B. Anderson, "First-year Students as Interviewers: Uncovering What It Means to Be an Engineer," *Proc. 39th ASEE/IEEE Frontiers in Education Conf.*, October 2009.

[6] D. Troy, D. S. Keller, J. Kiper and L. Kerr, "First year engineering: Exploring engineering through the engineering design loop," *Proc. 38th ASEE/IEEE Frontiers in Education Conf.*, October 2008.

[7] R. L. Porter and H. Fuller, "A new 'contact-based' first year engineering course," *Proc. 27th Annual Frontiers in Education Conf.*, October 1997.

[8] D. Gotterbarn, et. al., "Computer Society and ACM Approve Software Engineering Code of Ethics," *IEEE Computer*, October 1999.

[9] M. A. Fitzgerald, "Making the Leap from High School to College," *Knowledge Quest 32* (March/April 2004).

[10] J. Bergin, (2001, July 27). Learning the Planning Game: An Extreme Exercise [Online].
Available: http://csis.pace.edu/~bergin/xp/planninggame.html

[11] Robocode version 1.7.3.5 (2012, March 11). [Online].
Available: http://robocode.sourceforge.net