

Experiences using SPARK in an Undergraduate CS Course

Dr. Anthony S. Ruocco
Roger Williams University
School of Engineering, Computing and Construction Management
Bristol, RI 02809
(401) 254-3334
aruocco@rwu.edu

ABSTRACT

This paper describes experiences garnered while teaching a course on high integrity software using SPARK to a mix of junior and senior level undergraduates. The paper describes the impact of pre-requisites, course layout and execution, and lessons learned by students and the instructor. The course used the SPARK toolset provide by Praxis High Integrity Systems, and the Gnat Programming System (GPS) provided by AdaCore Technologies (ACT) under the Ada Academic Initiative Program. Details about using these tools is integrated through out the paper.

Categories and Subject Descriptors

D.3.3 [Programming Languages]: Language Constructs and Features – *abstract data types, constraints, control structures.*

General Terms

Algorithms, Documentation, Design, Reliability, Languages, Theory, Verification.

Keywords

Computer Science Education, Computer Science Curriculum, SPARK.

1. INTRODUCTION

Roger Williams University is an undergraduate institution with a liberal arts history. In 2000, the computer science program was moved from the school of arts and sciences to the school of engineering. This consolidated the Accreditation Board for Engineering and Technology (ABET) programs under a single Dean. One critical aspect of developing and assessing a curriculum is to gather needs from the program's constituency [1]. In many cases, this means contacting local industries where graduates from the program tend to be employed. Beginning in spring 2004, several local firms were queried concerning graduates with Ada95 programming skills. While only one representative specifically mentioned the use of Ada95 [Stevens,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGAda'05, November 13–17, 2005, Atlanta, Georgia, USA.
Copyright 2005 ACM 1-59593-185-6/05/0011...\$5.00.

Bruce, Naval Undersea Warfare Center, personal conversations], several mentioned the idea of high-quality software programming. The original intent was to utilize Ada95 as a means of demonstrating high integrity software programming. However, during the SIGCSE 2004 conference, the instructor became aware of the SPARK tool set and the High-Integrity Software textbook. Subsequent research indicated other programs were using the SPARK toolset [4]. Through correspondence with the author's of SPARK, contact was made with AdaCore Technologies. Roger Williams University became affiliated with the AdaCore Technologies Academic Alliance initiative [3] which provided the Gnat Programming System (GPS) IDE. With SPARK as the language, and GPS as the IDE, development of COMSC450 High Integrity Software began in earnest. This paper will discuss how the course was designed, how it was executed in the Windows Operating System environment, and most importantly, lessons learned.

2. Course Design

2.1 Student Background

This course is a computer science elective. It was opened to all computer science majors in their junior or senior year. Of the seven students enrolled, 2 were juniors, 4 were seniors and 1 was a December graduate. There was no formal pre-requisite list for this course. The computer science courses students had taken were as follows:

Table 1: Courses completed by students

Course	Completed	Concurrent
Intro programming	7	
Data structures	7	
Computer Org.	7	
Programming languages	7	
Theory of Computation	5	2
Analysis of Algorithms	5	
Operating Systems	2	3
Compiler Design	2	3
Senior Design I	2	3
Senior Design II	2	

The purpose of the course was not to learn Ada95, but focus on high-integrity software. A portion of the courses was also oriented on social and ethical issues of computing. This purpose is clearly evident in the course description and objectives.

Description: This course focuses on programming techniques for computer systems found in safety critical environments such as avionics, power plant and/or transportation systems. The course uses a specialized language (SPARK) and its tools to write and examine high-integrity code segments. Students become familiar with some of the differences between general programming languages and specialized computing languages. Use of the risks.comp news group also highlights non software-specific risks in large systems.

- Objectives:**
1. Understand the safety/risk implications inherent in high integrity software systems.
 2. Use specialized software tools in the production of high integrity code segments.
 3. Use a specialized programming language to produce high integrity code segments.
 4. Research an issue in high integrity software and present a possible solution.

The primary text was [High Integrity Software](#). John Barnes, Addison Wesley 2004. Students also purchased [Ada95 as a Second Language](#), Norman Cohen, McGraw Hill 1996 to use as their Ada95-specific reference.

2.2 Course Layout

The course met twice weekly, for a total of 28 lessons. The 85 minute session was sufficient for spending some time during each lesson in discussion-mode and then doing hands-on work. The hands-on work was especially important early in the course as students worked their way through the GPS tutorials and some small SPARK coding drills. The intent of the course was to discuss SPARK independent of Ada95. One reason for this approach was so the students would not develop a habit of writing code in Ada95 the retrofitting SPARK constraints to the code.

After a quick overview of Chapter 1, the course jumped to Chapter 4 for Language Basics. Ten lessons were devoted to language basics, including file IO operations. During these lessons, the Examiner tool was extensively utilized. The course covered SPARK basics and constructs. Types, subtypes and aggregates were explored and utilized in small code segments. Control structures were discussed with the RETURN statement receiving special attention. Packages and visibility were covered. The intent of this portion of the course was to have each student develop a mastery of the basics of the language and to provide a comfort-level for moving on to more advanced concepts. There would be one group project culminating this section.

The next phase of the course provided a detailed coverage of the examiner, flow analysis, and the INFORMED design methodology. By this phase of the course, the intent was for the students to work in groups to develop a significant product by having sub-teams work on small sections of a larger project, and then consolidate their sub-team product into a single artifact.

The final lessons of the course covered gtkAda95. This portion was done while students were working on their group project and an individual research paper on some aspect of high-integrity software. The intent was to show the students some of the tools which make Ada95 an extremely robust language

3. COURSE EXECUTION

3.1 Lessons Learned

The first weeks of the course consisted of a discussion of SPARK, and then utilizing small code segments under the GPS environment. Although the text indicated no knowledge of Ada95 was required to use SPARK, it was clear the students would need at least a minimal level of Ada95 understanding. Trying to use the SPARK text as a reference was of limited value, but the text by Cohen was well utilized. The course was executed using almost parallel developments of Ada95 topics as we covered that language construct in SPARK. The makers of SPARK provide several files which modify the GPS IDE in such a way that many of the SPARK tools are listed as drop-down items on the GPS menu. This proved to be invaluable. Students were able to produce code in the IDE then execute the SPARK tools as an option (comparable to the compile/build options). Once code was run through the SPARK tools, students were instructed in using the SPARK reports as the basis for analysis of code prior to the compilation process. The SPARK report was analyzed so that code changes were based on SPARK errors. Following (typically) several iterations of the SPARK report, the code was compiled through the IDE.

Trying to cover SPARK independent of Ada95 was not possible. The SPARK text was adequate in terms of representing the differences between Ada95 and SPARK. However, when students did not know Ada95, the references had little to no meaning. Doing parallel development was very doable, and beneficial. Future iterations of the course will follow this basic approach.

The SPARK toolset is designed to be command-line based. However, using an IDE to produce code, leaving the IDE to run SPARK, then returning to the IDE for correction was awkward. Students tended to by-pass using SPARK tools, instead going straight to compilation. This tended to find all the syntax errors, but bypassed SPARK requirements. Students then discovered that trying to SPARK any files they had already compiled to be annoying and in the case of advanced programs, very frustrating. Usually, trying to correct SPARK errors lead to other compilation errors, giving the students the feeling they were going over the same code much more than they should.

Once it was possible to integrate SPARK into the IDE, it was much more intuitive to write code, run SPARK, read the report, and then correct the code to SPARK specifications before compilation. The report was also available through the IDE. This made the process of using SPARK much more intuitive. When all operations were done through the IDE, it was much more likely the students considered the SPARK tools as part of the development process. The number times students by-passed SPARK during small code segments in class dropped considerably with the integrated SPARK tools. Unfortunately, for the larger projects with less direct instructor supervision, students did revert to the process of compiling in Ada95 to remove syntax

errors, and then ran SPARK after the fact. In discussions, students said they did this as a conscience effort to save time, though admitted that it frequently backfired in terms of reducing SPARK errors.

There were four projects during the course. The first was a confidence builder. Students were required to program several matrix operations as part of a matrix package. While relatively easy from a program perspective, it was a good vehicle to demonstrate SPARK rules regarding defining all index ranges. A further strength of using defined ranges was taking student packages and mix/matching them to show how package specifications and package bodies can be independent. About half of the students did the project following the intent of coding, using SPARK, debugging SPARK, then the usual compilation process. Others used the compiler to find Ada95 syntax errors, then trying to make their work SPARK-compliant. This was a good 'learning' opportunity in showing how the Ada95 compiler accepted code that SPARK rejected. Several students claimed they would have been more likely to follow the intent of using SPARK prior to compilation if they were more confident in their Ada95 skills. As the course progressed, confidence in basic programming did increase.

The second project was an expansion on the first, adding some matrix functionality. The intent was for students to create a child package off their first project. Unfortunately, creating the child package was not specifically stated as a requirement. None of the students used child packages, instead they simply added new procedures and functions to the code from project one. This created a need to re-examine all the code. When showed how a child package alleviated the need to re-examine all previous code, they indicated the child package should have been expressly required. If students had had previous experience with Ada95, they would have been more likely to use child packages as intended.

The third project proved to be difficult, but with beneficial side effects. The students were to determine maximum flow in a water system. A coded solution was provided. Students were required to take the Pascal solution [2], and convert it to SPARK. The students were divided into two groups for this project. The structured nature of Pascal should have translated fairly well into Ada95. There were some global variables which needed to be accounted for, and where Pascal requires loop control variables to be declared, Ada95 does not. Overall, while the code translation was expected to be tedious, it was not intended to be as onerous as the students made it out to be. On their own, the two groups melded into a single effort. They discovered fairly early that it is not the number of members that makes a group, but the organization of those members. As the students never organized themselves, their interactions can be described as chaotic at best. Students assigned each other pieces of the code to translate. However, several sections were done by multiple people, and some sections were not done at all, and were hurriedly attempted as the deadline approached. The students used a shared directory in the lab to consolidate work. The shared directory was somewhat protected, in that only students in the course had access to it. The lab directories are not backed up regularly, so there was a high risk of data loss. One student made frequent back-ups to disk to mitigate this risk. However, though they did consider backups, they made no provision for version control. It was

common, especially early on, for each student to do their portion of code, copy it into the directory thereby over-writing what others had previously done. By the time the deadline was reached, the students had managed to compile a version of code, which subsequently did not work. As the students had basically punished themselves, the lab was discounted and used as an experience in group dynamics.

The final project was intended as a total class effort. The class was to simulate a shuttle flight system controlling pitch, roll and yaw. There was a two-person team assigned to each controller. There was also an overall control team, coordinating the pitch, roll, yaw by responding to commands received from a file. The overall control team consisted of a member from the pitch, roll, and yaw control teams. Its purpose was to establish the specs that each controller needed for interaction with each other and the main controller. Based on the experience of project three, the students were specifically forbidden from reforming their teams and from holding super-sized meetings or discussions. Overall, this project went far smoother than project three. One major shortfall was that the three controls (pitch, roll, and yaw) were the same except for some parameters. The idea behind the project was valid, but there should be a greater diversity of the sub-components.

The original course plan developed during the summer was to cover tasking in Ada95, then look at some of the tools as they apply to the Ravenscar (which allows some tasking) profile. However, during course design it seemed that tasking would be too complex to cover in a meaningful way. Instead, thought was given to showing some aspects of the gtkada tools. While utilizing these tools can be complex, there are sufficient demos to give a sense of the flexibility of Ada95 in terms of GUI applications. While students were completing their research papers and project four, the final three formal lessons consisted of demos of various gtkada capabilities. None of these lessons were accounted for in terms of graded events. However, several students thought the availability of the gtk tools made using Ada95 a viable option against other languages touted for their GUI support.

Throughout the semester, examples were taken from the risks.com list. The students were asked to find and explain other instances of software risks, and were subsequently assigned a research requirement in the area of software risk. The risk.com list was used several times during the semester. Usually, this was done by finding interesting examples and using them to stimulate conversations. Many times, these conversations took place during lab-periods, or when students were doing small, non-graded exercises. This approach led to many interesting conversations concerning the social and ethical implications of computing. Through-out the semester, articles from newspapers as well the web were used to highlight general issues in the area of social implications of computing. Students were very good at identifying the implications of faulty software. They were very articulate in discussing recognized implications of software such as music/video copying. They also were well prepared to discuss issues relating to virus attacks. In general, articles directly relating to software glitches were understood. However, students tended to be less able to discuss non-direct social implications of successful software. For example, using face recognition software to identify known criminals was identified as a 'good

thing', but they did not consider any implications concerning saving all the images taken during the search process.

The course presented a good opportunity to present social and ethical implications of software. The topics were discussed in a very informal approach. Generally, the topics were introduced in a way that sought student opinion and thoughts. Their responses were non-attributable. Discussing things as students were doing hands-on work made the discussions even less hostile. This approach worked well as a means of raising awareness and the need to look at a technology issue from several perspectives. None of the information from these discussions was part of any graded event. Student comments indicated they enjoyed the conversations. As a side-effect, many felt the open discussions made the course a more personal interaction with the instructor and fellow students. The students did do an independent research paper on some social/ethical issue which was evaluated..

3.2 Student Reaction

The course was open to junior and senior students as an elective. As part of the course introduction, students were told they would be frequently polled as part of the assessment of the course. Much of the assessment was done through informal conversation with some formal assessment taking place at the conclusion of the semester. Almost every aspect of the course was covered in informal assessment. Some of the major comments applicable to all students were:

- Students felt they would have done better if the first two weeks were dedicated to pure Ada95 and the GPS IDE
- Material was covered at a reasonable pace, but some of the finer points (such as using child packages) were lost due to lack of familiarity with the language constructs.
- Though students had varied backgrounds, all indicated the most important course prior to this course was the Programming Language class (the version taught is a survey course). Upon further comments, the students felt the survey aspect of the course made them comfortable in facing a new language.
- Project three (the max-flow problem) was their worst programming experience, but best group experience of the course
- Gtkada was interesting, but without any course application, students felt they would be unlikely to experiment with it.

- Students felt a course dedicated to Ada95 should be implemented within the curriculum as an elective.

4. CONCLUSION

This course covered the Ada95 language to a sufficient depth to support use of SPARK. The intent was to develop the concept of SPARK as an inherent language in its own right. However, this proved to be very difficult. It is likely that students understand the reasons behind the development of SPARK, but consider it a set of specialized tools for Ada95. Incorporating the SPARK tools into the GPS IDE was critical to the successful use of those tools, at least in the Windows environment. Without the integration, students tended to bypass running the tools prior to compilation. The group work enforced course objectives as well as highlighting the impact of group dynamics. The course offered a good venue to initiate discussions and awareness of social issues of computing. Overall, the course, as executed, fully supported and attained the learning objectives. Most importantly, this course demonstrated that teaching high-integrity software can be accomplished at the undergraduate level.

5. ACKNOWLEDGMENTS

This course could not have been a success without the assistance of several people. Rod Chapman of Praxis Critical Systems provided technical assistance in using the SPARK tools and for integrating SPARK tools with the GPS IDE. Robert Dewar of AdaCore Technology for providing support through the Ada Academic Alliance. And thanks to Karen Mason, also of AdaCore Technologies for managing the GAP listserve.

6. REFERENCES

- [1] Computer Accreditation Commission, *Criteria for Accrediting Computing Programs*. ABET Inc, Baltimore MD, 2003.
- [2] Syslo, M., Deo, N., and Kowalik, J., *Discrete Optimization Algorithms with Pascal programs*. Prentice-Hall, Inc. Englewood Cliffs, NJ, 1983.
- [3] www.gnat.com/academic_overview.php
- [4] www.praxis-his.com/sparkada/universities.asp