The Draw-Bot: A Project for Teaching Software Engineering

Martin v. Mohrenschildt
Computing and Software
Communications Research Laboratory,
McMaster University,
Hamilton, Ontario, Canada L8S 4K1
e-mail: `mohrens@mcmaster.ca`

Dennis K. Peters
Faculty of Engineering & Applied Science
Memorial University of Newfoundland,
St. John's, Newfoundland, Canada A1B 3X5
e-mail: `dpeters@engr.mun.ca`

June 4, 1998

Abstract

We present a course project which was successfully used to teach software design principles to third year computer engineering students. The goal of the project is to program a robot to trace a shortest path through a maze. The students, organized in teams of five, have to follow the classical steps of software development and prepare interface, design and testing documents. Having a project that requires controlling a device to complete a clear task generates enthusiasm in the students and helps them to understand the principles taught in the course.

1 Introduction

Computer Engineering 3VA3 is a third year course in software design taught at McMaster University. A key element of this course is an opportunity to apply software engineering principles such as software specification, design and testing to systems with safety-critical and real-time requirements in a concrete project. In previous years pure software projects i.e., projects that did not involve any hardware other than the computer, were used. Such abstract projects have a number of disadvantages:

- they often have many seemingly arbitrary rules,
- they often fail to inspire enthusiasm in students, and
- it is hard to tell success from failure.

In this paper we present an alternative project: controlling a robot constructed from a commercially available kit to complete a simple task. In the next section we present the educational goals of the course and the criteria for choosing the project. Section 3 describes how we conducted the project to avoid the problems that typically arise in such project based courses. We give an overview of the project in section 4 and evaluate its success in section 5.

2 Educational Goals

This course is intended for students who already know how to program and are familiar with the use of data structures. In this course, students are exposed to the principles of professional software development from an engineering point of view. The major topics are

- software specification and documentation,
- modularity,
- module interface design,
- module internal design,
- real time and safety critical software,
- testing and inspection, and
- software development in a team.

Past experience has shown that students often do not realize some of the issues that arise in multi-person software development projects until they are confronted with them. The project, which is a major component in the course (60% of the final mark), exposes the students to some of these issues and allows them to practice the principles taught in the course.

For the course to be successful, the project must have a concrete and well defined goal. Its requirements must be achievable, and the amount of effort required must be appropriate for the course timetable. The students' level of interest is also a key factor, which we have found is significantly improved by requirements that are based on reality.

We chose to step away from the pure software projects used in previous years and to design an "action" project—a project where there is some physical interaction of a computer with the world. The goal of the project is to control a robot to trace a shortest path through a maze. To stay feasible and safe we could not use real industrial robots, so we constructed the robot using a small educational robot kit that is controlled through a software interface. In this way the students don't have to concern themselves with the hardware aspects of constructing or controlling the robot. The software-hardware interaction offers interesting and challenging problems including real-time and safety critical aspects.

This software engineering course is a major component of the curriculum for computer engineers (and, since 1998, also electrical engineers). It is the last course in a series of three software courses where students are taught the principles of professional software development. This course is a cornerstone of the education of these students since they will be expected to develop software as part of engineered systems during both their studies and their professional careers.

3 Project and Course Organization

The classroom time for the course consisted of three 50 minute lecture periods per week over a 12 week semester. Two periods each week were used for lectures to present the theoretical concepts and principles, and the third was used as a tutorial to discuss the project.

3.1 Phases and Milestones

At the beginning of the course the students are given the *requirements specification*—a document that precisely describes the required behavior of the software. Further it describes all deliverables for each of the project phases. We found that this structure is essential to ensure that each student understands what is expected. The requirements specification can be seen as a contract between the client (instructor) and the software developer (students).

We identified seven project phases, each of which results in a document:

1. Modularization
   The *module guide* presents the modularization chosen by the groups. It contains a list of all of the modules in the system, and, for each module, an informal description of the "service" it provides, and the "secret" it hides. It also describes the interaction between modules by specifying (typically using a figure or a table) the modules each module relies upon in order to provide its service (i.e., the *uses hierarchy*).

2. Interface Design
   The *module interface specification* describes the interface to each module by specifying all interface functions, their input and output arguments, types and exceptions and the declaration of data types if needed. The visible effects of each interface function is described using one of the specification techniques presented in the lectures.

3. Internal design
   The *internal design document* describes how the modules provide their services by presenting and explaining the internal data structures and algorithms of each module.

4. Code Inspection
   Each team must prepare a *code inspection report*, which lists the properties of the code to be checked during the inspection, the method to be used to conduct the inspection, and the rationale for choosing this method. Reports of the actual code inspections list any problems that were found during the inspection.

5. Testing
   The *module testing report* describes how each module was tested. This testing should be conducted by someone other than the primary author of the module and that person must prepare a test report. It list the test cases and expected and actual results.

6. System Safety Analysis
   The *system safety analysis report* shows in a verifiable manner how the software satisfies the safety requirements. It starts from the possible failures and, with reference to the design, shows how such a failure is prevented.

7. Final Presentation
   All of the documents are combined into one final document, and the groups demonstrate their software.

As each phase was completed, the students were required to submit a draft of the corresponding document, which was credited with 20% of the value for that document. These were reviewed by the instructor or TA and feedback was given to the students. Having these milestones, with fixed due dates, kept the students on track: they worked consistently over the term rather than procrastinating until the end of the term. In addition this provided an opportunity to correct poor design decisions before they had too much effect on the project.

3.2 Project Teams and Leagues

The project was conducted in teams of five students. This number was chosen considering the size and amount of documentation and programming required. Past experience has shown that the formation of the teams is crucial: if the students are permitted to form their own teams top students tend to group together, resulting in very strong groups and weak groups. To solve this problem we decided to mandate the most knowledgeable students as group leaders and let the leaders choose their teams. The leaders were chosen using results of a multiple choice test given in the first lecture of the semester.

The tutorials played a major role in the project: they were used to give guidance in design, documentation and other technical matters. Further groups were asked to give small presentations of their own designs in order to expose all students to a variety of design alternatives and to give students an opportunity to practice and improve their presentation skills.

For these tutorials we split the groups into *leagues* which were lead by either an experienced TA or the instructor. Within each league the groups agree on high level design decisions, such as the modular structure and the interfaces to the software components, and submit a common module guide and set of module interface specifications. This helps to emphasize the importance of carefully documenting design decisions and gives practice in developing the system to adhere to the specifications. An added advantage is that groups of one league can interchange project modules if one of the groups fails to complete one or more modules (e.g., due to a student dropping the course or getting sick).

3.3 Instructional Resources

This course required significant preparation effort to develop the project and prepare the requirements document. In addition, several components had to work for it to be successful:

- **League Leaders** The league leaders are the main source of direction for the students with respect to the project, so they must be experienced in software design and have a solid knowledge of the practice of the principles being taught in the course. In addition to an instructor who has industrial experience, we were lucky to have very competent and experienced TAs who were involved in the development of the project and were capable of acting as league leaders.

- **Hardware** (Robots, Computers) The draw-bot is constructed using a Robix^{TM} RCS-6 construction set [1]. It consists of three arms, each of which is controlled by a motor. The first two arms move in the horizontal plane to position the pen and the third arm is used to raise or lower the pen. The modular structure of this set and the simple software interface made constructing the draw-bot and interface software easy. Also it is easy to construct new robots to be used for future projects. We used two kits for our class.

- **Software** (Interface, Simulator) Since this is a course in software, not hardware, we provided the students with a software interface to the hardware. This interface contained only the elements needed to control the robot—accelerations, speeds, and hardware interface details were not shown. In addition we provided a simulator, which has exactly the same software interface as the actual hardware and illustrates the behavior of the robot on the computer screen. This turned out to be essential since it allowed students to test their software without using (and possibly damaging) the actual robot. All students who “mastered” the simulator got the robot running the first time.

4 Project Overview

The goal of the project is to control a robot to trace the shortest path thought a 2-dimensional (paper) maze. Further, some real time and safety critical elements are added: there is an emergency stop button, a home button (return to initial position) and a reverse button (trace backwards).

In the requirements specification the behavior is described in terms of the *monitored* and *controlled* variables. [2][3] These are the quantities that the software can measure, e.g. the status of a button, and the ones that the software can set, e.g. the position of a certain motor. To aid in understanding, and to help expose students to a variety of formats, the requirements are presented in three forms: Informal, Logic, and State Machine, all using the notation of the textbook [4]. These descriptions are intended to describe the same behavior and are in some sense complimentary. In this paper we only show parts of the informal description, the complete description can be found in [5].

4.1 System Interface

Table 1 lists the variables that represent the quantities in the environment to be monitored and/or controlled by the system. All environment variables are functions of time.

Table 1: Environmental Variables

Variable	Type	Description
i_mazeWalls	set of **positionT**	The set of points that make up the walls of the maze. Note that the exterior walls (i.e., the perimeter) are included.
i_mazeStart	**positionT**	Start position for the maze.
i_mazeEnd	**positionT**	Finish position for the maze.
i_stopButton	**buttonT**	The status of the button labeled “stop”.
i_homeButton	**buttonT**	The status of the button labeled “home”.
i_backButton	**buttonT**	The status of the button labeled “back”.
i_mazeFile	**string**	The file name passed on the command line.
o_penPos	**positionT**	The position of the pen relative to the ‘origin’ $\langle 0,0\rangle$, which is the center of the robot base post.
o_penDown	**Boolean**	*true* iff the pen is touching the plane containing the maze. Assumed to be initially *false*.
o_powerOn	**Boolean**	*true* iff the robot power is on. Assumed to be initially *false*.
o_message	**string**	The message displayed on the operator console.

4.1.1 Pen Position

We represent the location of the draw-bot pen tip using a Boolean, o_penDown, to indicate if the pen is touching the maze surface or not, and a pair, $\langle$o_penPos.x, o_penPos.y$\rangle$ of reals, representing the location in the horizontal plane where the pen tip is touching the maze (if o_penDown is *true*) or would touch the maze if lowered. The ‘home’ location of the pen-tip (to which it is returned on initialization of the draw-bot), is $\langle HOME\_X, HOME\_Y\rangle$.

4.1.2 Maze

As illustrated in Figure 1, the maze is contained within a 15×15 cm region of the horizontal plane. The ‘internal walls’ of the maze are segments of the lines forming a square grid with line spacing 10 mm. Figure 2 illustrates a maze, reduced from an actual size of 15×15 cm.

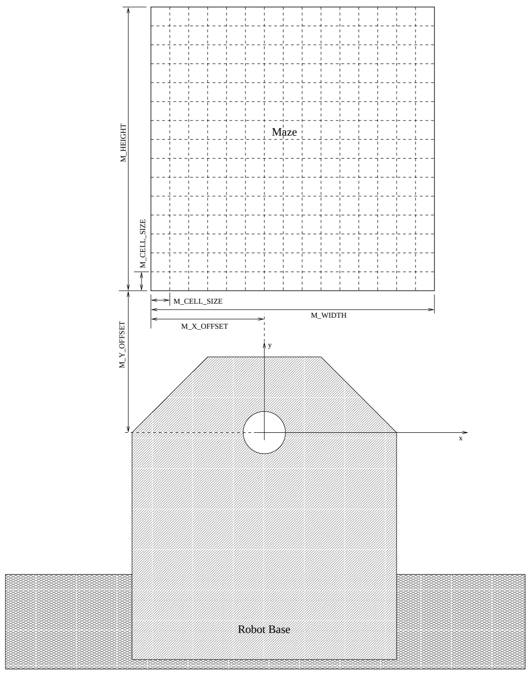

Figure 1: Robot and Maze Parameters

4.2 Behavioral Requirements

This section describes the required behavior of the Maze-Tracing Robot in terms of the environmental quantities described in Table 1.

4.2.1 Safety Requirements

As stated, our project contains safety critical aspects. These are requirements which have to be satisfied no mater what happens:

If at any time the stop button is pressed the robot must stop moving within 0.5 seconds and must remain stationary until the stop button is released.

When the pen is down the pen tip must never come within 2 mm of a wall point.

In the final evaluation we examined the documents and software extensively for these requirements. Failing to fulfill them resulted in a significant reduction in the group’s grade.

4.2.2 Modes of Operation

The operation of the draw-bot can be split into several phases or *modes*. The following is a description of the

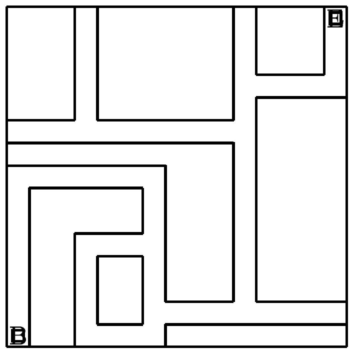

Figure 2: Sample Maze

required behavior of the robot in each of these modes.

Initialization When the program is started i_mazeFile is read. If an error occurs (e.g. file read failure) or if there is no path through the maze, then an appropriate diagnostic message must be output and the control program must exit without powering the robot.

Starting After i_mazeFile has been read, and it has been determined that there is a path through the maze, the robot power must be turned on, which initializes the pen to the home position with the pen up. The pen must then be moved to the start position of the maze.

Forward Once the starting position has been reached the pen must be lowered and a path traced through the maze to the end. When the pen reaches the end of the maze it must be raised and returned to the home position.

Reverse If at any time while the path is being traced the "back" button is pressed the Draw-bot is required to reverse the direction of its tracing within 0.5 seconds and begin to re-trace its path back to the beginning. It should continue to re-trace it's path only as long as the "back" button is held down; when it is released the Draw-bot should continue in the forward direction. If, while reversing, it reaches the start position it should stop there until either the "back" button is released or the "home" button is pressed.

Home If at any time while the path is being traced (in either direction) the "home" button is pressed the Draw-bot is required to stop tracing within 0.5 seconds, raise the pen and return to the home position, without making any further marks.

Done When the pen has been returned to the home position, the power must be turned off and the system must exit.

5 Assessment

The project was very successful from several points of view. We did not encounter large technical nor organizational problems, this is mostly due to the vast amount of preparation and the strong commitment of the TAs.

All but one group completed the project and, thanks to the league structure, the software produced by the one unsuccessful team could be shown to work by using some components from other teams in the same league.

The team selection process (based on our selection of team leaders) was quite successful at quickly forming relatively compatible teams, while still ensuring that the most knowledgeable students were not grouped in the same team. There were relatively few personality conflicts within the teams and no team was excessively stronger or weaker than the others.

The student response was very positive. The anonymous course evaluation contained only positive comments and is reported to have been one of the most positive ever seen in this department. Most students stated that they felt that they solved a real problem and learned a great deal. Our formal approach, to which we encountered scepticism at the beginning, was accepted by the majority of the students at the end of the course. They understood that a well structured approach to software development, even if it initially seems like more work, pays out in the end. The students were excited to test their project using the real robot and were happy that it worked from the beginning most of the time, and was stable even in unexpected situations.

5.1 Problems Encountered

Our group structure helped many weaker students to accomplish a task which they would not have been able to do by themselves. But since the project mark was the same for all members of each team, the individual final marks did not accurately represent the skills or effort of each individual student. A grading scheme that will allow us to give more individual marks would be an improvement, but is difficult to do in a fair manner.

The source code of the simulator was provided to the students, which lead to some confusion. Some groups made some minor changes to it, which they should not have, causing them some problems as they tested using the real robot. However, the lesson that the students learned from this experience—that it is important to respect a module interface no matter how unwieldy it may be—was valuable.

The technical aspects of the draw-bot caused some difficulty in getting the mazes to be traced correctly using the real robot. For example, positioning the pen tip such that the pressure applied was sufficient to make a mark, but not so much that it impeded the draw-bot movements, was quite difficult and required that the pen height be adjusted depending on the distance of the pen from the base. The interface provided to the students did not allow them to adjust the pen height—they could only set it to "up" or "down"—so they were not able to account for this affect. We overcame this by adjusting the configuration such that the pen did not actually make a mark, but just came close to touching the paper. In addition the actual value of o_penPos was quite sensitive to small mechanical variations in the robot construction so it was not always possible to avoid 'touching' the 'walls'. We accepted a demonstration using the ideally behaving simulator as evidence that the student designs behaved correctly in that respect.

6 Conclusions

Software team projects are often a significant component of software design courses similar to Computer Engineering 3VA3. We have found that choosing a project that involves controlling a device to complete a clearly defined task has a number of advantages:

- the students are more enthusiastic,
- it is easier to tell success from failure,
- the project requirements can be based in reality, and
- the importance of interface design is clearly illustrated.

These advantages outweigh, in our opinion, the effort required to develop such a project.

The 'league' structure adopted for this course emphasizes the importance of clear module interface definitions and gives some resilience to certain kinds of failure. This structure can only be used where there is a sufficiently low ratio of students to qualified league leaders.

Acknowledgments

We gratefully acknowledge the efforts of Dr. David L. Parnas, who offered many thoughtful comments to help give us the benefits of his experience with teaching this course in the past. Kavitha Nadarajah was one of the TAs for the course and her assistance is greatly appreciated. We obtained a grant from the Teaching and Leaning Committee of McMaster University, which was matched by the Department of Electrical and Computer Engineering. This enabled us to pay for the robots and for the services of Igor Vulanovic to develop the simulator. Weixiong Lin helped to fine-tune the draw-bot configuration and developed some test software and mazes.

References

[1] "ROBIX^{TM} RCS-6 robot construction set user guide and project book," Advanced Design, Inc., 1101 East Rudasill Road, Tucson, AZ 85718 USA, Sept. 1995. URL `http://www.robix.com/`.

[2] Heninger, K., Parnas, D. L., Shore, E. J., and Kallander, J., "Software requirements for the A-7E aircraft," Tech. Rep. MR 3876, Naval Research Laboratory, 1978.

[3] van Schouwen, A. J., Parnas, D. L., and Madey, J., "Documentation of requirements for computer systems," in *Proc. International Symposium on Requirements Engineering (RE '93)*, pp. 198–207, IEEE, Jan. 1993.

[4] Hoffman, D. and Strooper, P., *Software Design, Automated Testing, and Maintenance: A Practical Approach.* International Thomson Computer Press, 1995.

[5] Peters, D. K. and von Mohrenschildt, M.. Course Handout for Computer Engineering 3VA3, McMaster University, Sept. 1997. Available at `http://ece.eng.mcmaster.ca/faculty/mohrens/robot/robot_rs.html`.