Jonathan D'Andries
Final Research Paper
Ectropic Project
29 April 2004

Student Surveys and ECoDE Use: Research Wrap-Up on Ectropic

I. Table of Contents – Section Headings and Appendixes

I. Table of Contents....1
II. Introduction....1
III. Research Design....2
IV. Findings: Background Questionnaires....2
V. Findings: Exit Surveys....3
VI. Findings: Student Usage and Designs....4
VII. Conclusions....10
VIII. References and Acknowledgements....11
IX. Appendix A: Background Questionnaire Results....12
X. Appendix B: Written Responses from Background Questionnaire....16
XI. Appendix C: Exit Survey Results....18
XII. Appendix D: Written Responses from Exit Survey....23
XIII. Appendix E: Background Questionnaire....36
XIV. Appendix F: Exit Survey....38
XV. Appendix G: Consent Form....41

II. Introduction

This study is a part of the ongoing research effort on Ectropic software development (*ectropy* being the opposite of *entropy*). The vision of the Ectropic project is to provide a less centralized means for open source development on a large scale. The existing Ectropic Design tool is called ECoDE (Ectropic Collaborative Design Environment), which is built in Squeak, and provides support for object-oriented analysis (OOA), object-oriented design (OOD), and direct correspondence with object-oriented programming (OOP). To study use of the ECoDE tool, we chose to target students of an introductory software design course (CS-2340, Objects and Design) because students are easily accessible in large numbers and are generally willing to offer low-cost feedback. Students learning software design have different needs than professional programmers, but they share the goal of quality software development in a team environment.

This semester's research study sought to evaluate how effectively ECoDE supports the process and products of object-oriented software design. Specifically, we examined whether extensive and effective use of ECoDE influences better design process, documentation, and implementation. We looked at designs and ran critics to see how consistent they were in terms of design, documentation, and code. Additionally, we used log files to evaluate students' usage patterns and then to suggest features to improve the tools.

To narrow our focus on the effectiveness of and student response to the ECoDE tool, we have posed two questions:

1. How much and in what way do students use the ECoDE tool?
2. How do students subjectively evaluate the ECoDE tool?

In this paper, I will address these questions, first by explaining the research design, then by analyzing student responses in the background questionnaire and exit survey, next by evaluating student use of ECoDE, and finally by providing conclusions and describing what I have learned from the research.

III. Research Design

In the spring 2004 semester, the CS-2340 Objects and Design course had 110 students of which eighty-one completed the consent form, background questionnaire, and exit survey (see Appendixes E, F, and G). Students received one point of extra-credit on their final average for participating in the study and allowing us to use their log files and designs to assess student use of ECoDE. Of the CS-2340 students, only three were women, and most were traditional college-aged males.

To gather data, a background questionnaire was distributed on March 25, 2004, to obtain participant descriptions, including information about their majors, class standing, grade point average, expected grade in the course, years programming, previous coursework, and experience in software development. On the final day of class on April 20, 2004, I administered the exit survey, which asked students to describe their impressions and experiences using ECoDE. Students reported the modes in which they worked, responded to the design critics, evaluated Ectrospace, and suggested future improvements to the ECoDE tool. As a final method of data collection, I analyzed anonymous ECoDE log files and designs for usage data. I described patterns of student usage and then analyzed in depth few cases of student use. I report on analysis of this data in the following findings sections on background questionnaires, exit surveys, and student usage and designs.

IV. Findings: Background Questionnaires

Appendix A provides graph and table information gathered from the background questionnaires. Of the eighty-one research participants, seventy-nine are computer science majors, all of whom are in their sophomore, junior, or senior years at Georgia Tech. Forty-seven, or more than half, have previously taken the CS-2335 Software Practicum course, but most have very little experience with software design. Overall grade point averages vary widely, with the largest number (34% or twenty-seven students) falling in the 3.0-3.5 range and the second largest (23% or nineteen students) above 3.5. Almost all students expect to earn an A or B in the CS-2340 course. Students' years of programming experience vary widely from less than two to almost twenty years of experience. Most students fall into the three to seven year range, although variation is wide both within and outside this range. Similarly, students have a wide variety of experience with programming languages. Most have a solid grounding (medium or high experience) in Java and C, and many students program in C++, Basic/Visual Basic, Scheme/Lisp, and Perl. Still other students mention the languages of SQL, Pascal, PHP, HTML, Python, TCL, Assembly, and Fortran. The pie charts and bar graphs in Appendix A visualize these findings in more descriptive detail.

Of eighty-one total respondents, fifty-two report software development outside the classroom. They show a mix of personal and professional experience, with many students designing "for

fun" or for their own benefit and still more for co-op, freelance, fulltime, and other employment. Interestingly, return-to-college students reported their experience and even motivations for taking coursework in software development. As one student explained, he learned to program at work, but now seeks a bachelor's degree in computer science: "I have been a professional developer for the past twelve years and returned to GT to complete my BS in 2002." Another student learned to program at work, but through self-initiated study: "While interning at a company a couple of years ago, I wrote several Perl scripts, mostly to update many files at a time on a web server. It was not any sort of structured design process. I was assigned the task and wrote the script on my own." Still, other students report working with computers from the early ages of six and ten and describe their enjoyment working with programs, such as Basic, that today are outdated. Certainly student responses show a range of independent software development, but show commonalities in that students demonstrate personal and professional reasons to learn.

V. Findings: Exit Surveys

Most findings from the exit surveys should be "taken with a grain of salt," so to speak. Many of the exit surveys can be categorized into extreme like/positive reaction or extreme dislike/negative reaction toward ECoDE or Squeak (and, hence, ECoDE for being associated with Squeak). Of the total eighty-one participants, I categorized fourteen into the strong positive category and sixteen into the strong negative category when reviewing student surveys. I based this categorization on the frequency of praise/criticism for ECoDE as well as the student's tone in written responses. While the majority of responses (forty-eight) balance positive and negative comments, surveys in the extremes consistently support or tear down ECoDE in all responses. One respondent, for example, said ECoDE should self-destruct and suggested adding the critic of Ebert, referencing the movie critic. The tone of this survey was not only negative, but also cynical, particularly toward the class. Interestingly, this respondent said he learned nothing from ECoDE or from CS-2340, as he was re-taking the course and had previously earned a D. His comments are indicative of many of the extreme negative surveys, as students who adamantly argued against ECoDE also freely reported *not* using the tool. One person even reported that they liked the tool because "it allowed me to have the other team members do all the work."

Similarly, students reported contradictory responses. Some of the students who had not used Ectrospace made suggestions on how to improve it. Students who reported disliking the critics said it did help them with the design process. Also survey participants reported that they could accomplish on paper everything what ECoDE works to accomplish, but then they listed the auto-generation of code as what they liked the most about ECoDE – clearly indicating functionality that cannot be achieved when designing on paper.

Despite the large number of extreme and contradictory responses, students did supply some very thoughtful reflections on their use with the ECoDE tool. They reported that ECoDE made the design process easier, helped them understand and use CRC cards, illustrated how closely form follows design, facilitated the relationship between responsibilities and methods, made organization easier, and helped bridge the gap between design and implementation. Many students have used Visio or other design and drawing programs and, therefore, were able to compare ECoDE with existing tools. Students generally appreciated having a program specific to their class and programming language and reported benefits of ECoDE, including a clearer

idea of course requirements, design organization, and method sequences. Students said that ECoDE "simplified things," "helped us focus on learning design," and "encouraged a better, more focused design process." One person pointed out that ECoDE served as a model of what could be done in Squeak, explaining that the program benefited their learning in the course, as it "provided an example of a useful program written in Squeak." Because students reported that ECoDE helped them "identify the steps necessary to effectively designing software," the program has accomplished its goals as well as supported the CS-2340 class, at least for some students. Please see Appendixes C and D for visual representations and full student responses.

A majority of students report that CRC cards and scenarios are what they like most about ECoDE, and that the UML layout is what they like least. Therefore, I believe the future researchers should focus on completing the UML functionality to allow students to save their layouts and make significant changes to the arrows in this visual mode. Also of note is the fact that very few students used Ectrospace, but of those who did, most felt that they should be able to save their changes incrementally when working simultaneously with peers. There is need to extend Ectrospace such that we allow students to make changes simultaneously and write critics that make sure these changes are consistent.

VI. Findings: Student Usage and Designs – ECoDE Log Files and Design Analysis

Of twenty-nine groups in the class, I compared the eleven from whom we had signed consent forms for every student in the group. I looked at overall usage patterns to determine who (how many of the group members) used ECoDE, when they used ECoDE, and how much they used ECoDE. The goal was to describe patterns for how students iterated between code and design during the lifecycle of the project and to relate this to the success of ECoDE in general. I am drawing on my experience as a CS-2340 teaching assistants (I was a TA and have graded some of the students in this study) and as a student who took this class before ECoDE to draw conclusions about the patterns observed in the logs and in the designs.

Who Used ECoDE?

For each log event, we record the initials of the students using Squeak at the time it was logged. We can apply this information to determine how many of the group members actually used ECoDE and to what extent each used it. In our eleven cases, I was a little surprised by the distribution of work. Even as a programmer of software intended to help people work together, I fully expect the design aspect of this course to be relegated to one or two members of the group who are either motivated to keep the design consistent or have this job assigned to them by other team members who would rather focus on the code. This strategy is very well explained by one of the responses to the exit survey: when asked how ECoDE helped their team, a student responded that it made it easier for the documentation guy in our group to do the documentation." This response clearly indicates that a single person was assigned the non-programming tasks. Figure 1 is an example demonstrating this phenomenon:

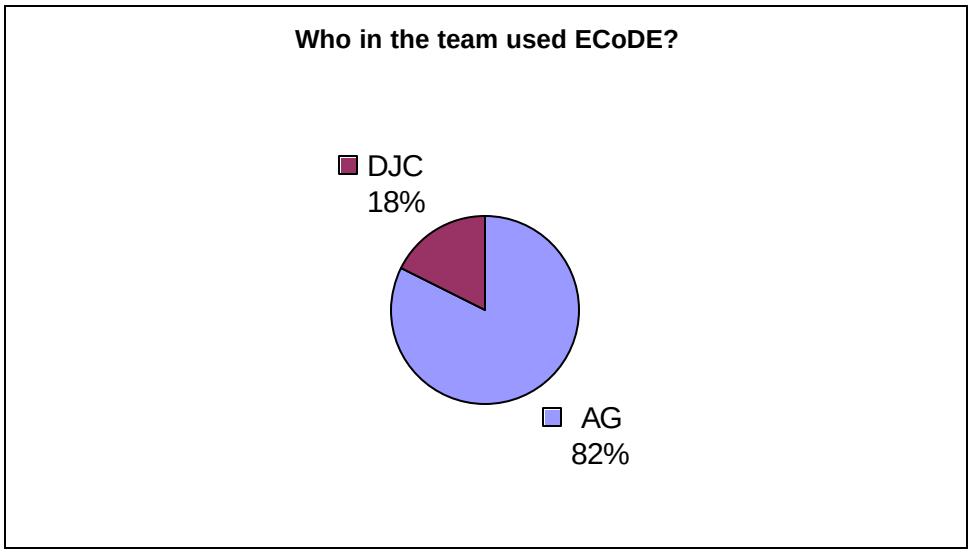


Figure 1: Demonstrating the assignment of a single member to do most (> 80%) of the design documentation. Only four of eleven groups followed this pattern.

Perhaps the most interesting aspect of this data is that only four groups demonstrated this pattern (and in only one group was the work done entirely by one person). More common was the distributed model of work in which at least three members made significant contributions to the work (> 10% contribution). Figure 2 represents a common pattern in who used ECoDE:

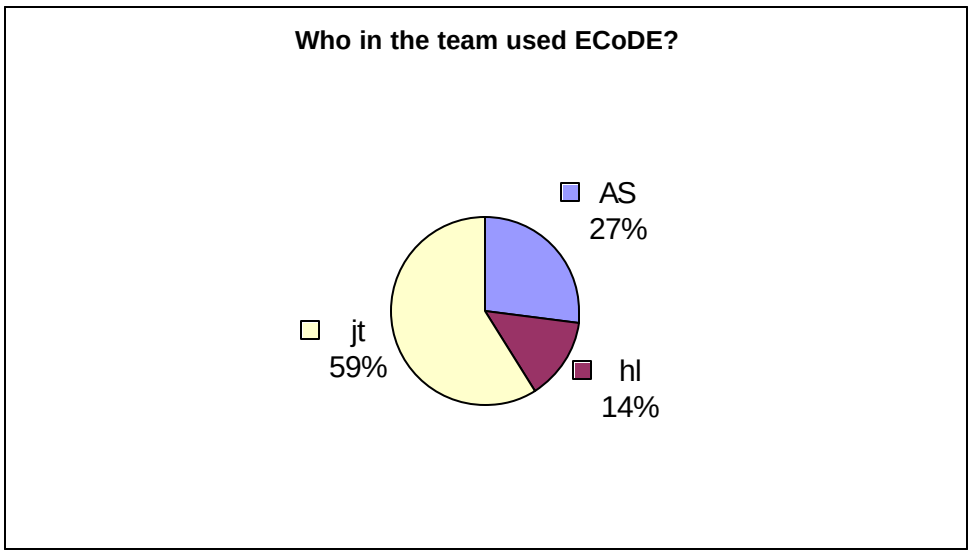


Figure 2: A more distributed model of design work, with at least three members contributing significant (> 10%) portions of the work.

This distribution of work validates efforts for collaboration in ECoDE. If students are willing to pass around the design to different computers where each member makes changes, then they can and will benefit from parallel work with synchronization mechanisms designed to keep this work consistent. Ectrospace can reinforce distributed designing by storing their work in a central place and alerting users to potential conflicts with other team members as they design. More

importantly, however, we can try to convince students that they do not need a "documentation guy" if ECoDE makes understanding and updating the design from the code significantly easier than going back to paper copies. Instead of being a static representation of the code in UML format, ECoDE design documentation can change as students code and facilitate a more iterative design process.

In terms of the current version of ECoDE, we have already made significant contributions for collaborative support. Ectrospace saves work in a central location, though only four groups in the class chose to use it. Critics always reference the current state of the design and code, so students can update in response to a critic, making their work consistent as they make changes instead of all at the end. We also support direct generation of code for any method or attribute in the design. If students have created or added any new features to the design, we can generate stubs of code and comments for them to work from, thus saving time with rote input of methods like accessors and modifiers that can significantly bog down a coding session.

What we are missing in the current version is the ability to update the design from the code. We can import new classes with all their methods and attributes, but we have left off the ability to update classes already in the design with new information from the code. There exists the workaround of deleting the current design class and importing from the code, but users lose analysis-time concepts (CRC card information) this way. Rather, we should add a feature for "update design from code" just as we already have the feature for "update code from design." The more we support the leap from code back to design, the more students are likely to be collaborative when updating the design documentation.

When Did Students Use ECoDE?

For each log event, we record the date to determine when, for each milestone, students are updating their designs. In addition to working collaboratively, we hope to encourage students to work iteratively. For this category, unfortunately, the eleven groups all show discouraging patterns. On the positive side, most of the work is spread out over four or five days for milestone 3, which is when students are required to come up with a complete design for the entire system (no code is due for this milestone). This is representative of working through the requirements of the system and adding support for each significant feature in the initial design. What is negative is the distribution of log events after milestone 3. Once students start working on the code, they generally do not go back to the design until the day before the milestone is due. Figure 3 shows this trend toward single-day updates that is common to all eleven group as the semester progresses:

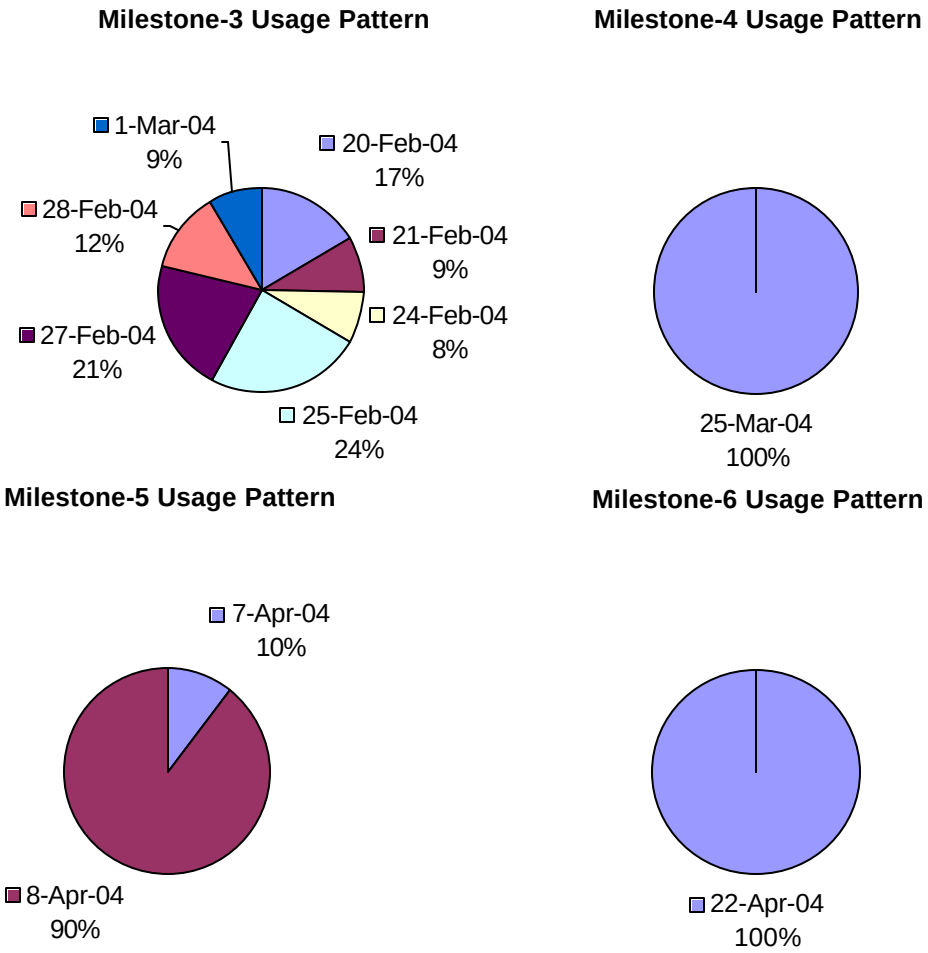


Figure 3: When did students use ECoDE? As the milestones progressed, all the updates in ECoDE were made the day before (or on the day) the milestone was due.

We, as designers of ECoDE, should find more creative ways to make the process of building software more iterative in terms of going back to the design. Some of these methods were discussed when finding ways to support collaborative work (in the section "Who Used ECoDE?"). Most notably, we need to extend the concept of Ectrospace and critics so that students can work on the design in parallel and not worry about keeping the design consistent with other members of the group. The lack of iterative design, however, could also be a bi-product of this educational environment. As a student, I found myself struggling to get the code working to the point of staying up all night before the project was due to get things complete. It is likely students see working code as a higher priority than keeping the design consistent with the code. We, as contributors to the CS-2340 software design class, should find a way to make design more significant in the minds and motivations of the students.

How Much Did Students Use ECoDE?

Milestone by milestone, we studied the log files to determine how much students used ECoDE as the semester progressed. Even if students did not iterate consistently as they coded the project, they may have iterated from one milestone to the next as they changed the way their programs worked. This data was more encouraging, as the tool seemed to support students who made

significant changes to their work. We saw three distinct patterns in usage over the semester. In the first pattern, shown in figure 4, students refined their designs up front and made fewer changes at the end of the semester.

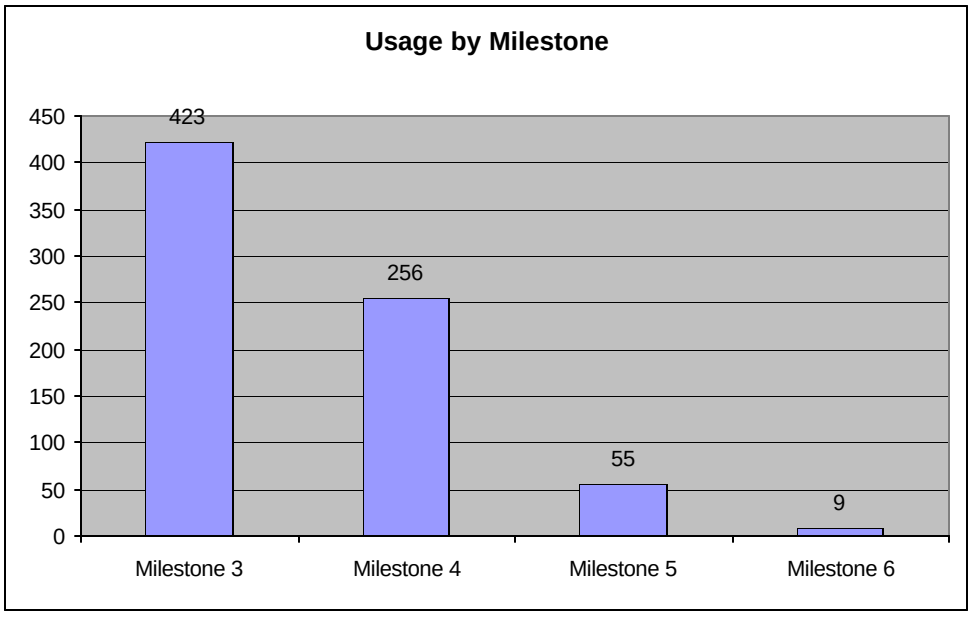


Figure 4: Six of eleven groups studied showed this decreasing pattern of making changes in ECoDE.

We recorded six of eleven groups with almost exactly the pattern as described in figure 4. Looking at the individual designs, there did not appear to be significantly lower quality based on this pattern. Some groups may have found that their early designs worked fairly well and only needed minor changes throughout the semester. These are probably students who did not experience as many problems with their design as they fleshed out the code. It could also be true, however, that students did have problems, but were only making changes where the grading TA commented for each milestone. In the second usage pattern, shown in figure 5, student apathy toward the design has likely produced very minimal effort in the end product.

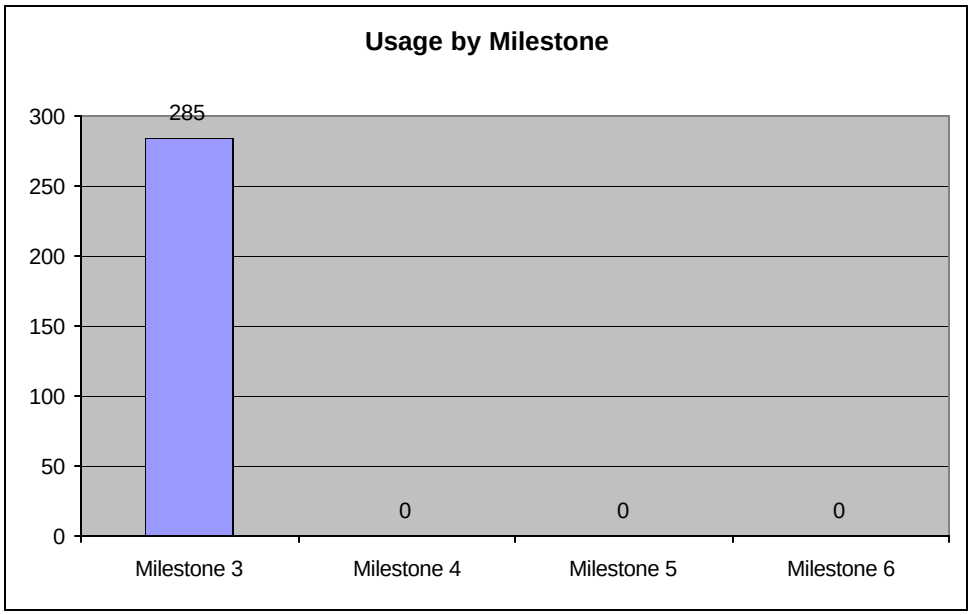


Figure 5: Two of eleven groups showed very minimal changes over time.

This pattern is clearly waterfall design methodology. In this case (two of eleven groups), students never used ECoDE after the design milestone (milestone 3). I find it discouraging that TA's could not find any way to motivate students to change their design even once after the first submission. It is hard to draw conclusions about student use if they do not use it, so I consider this an exceptional case not worth analysis. The third pattern, however, is much more encouraging. Students initially set up a design in milestone 3, made significantly less modifications for milestone 4, but continued to make more modifications with each milestone as the semester progressed. This pattern is shown in figure 6:

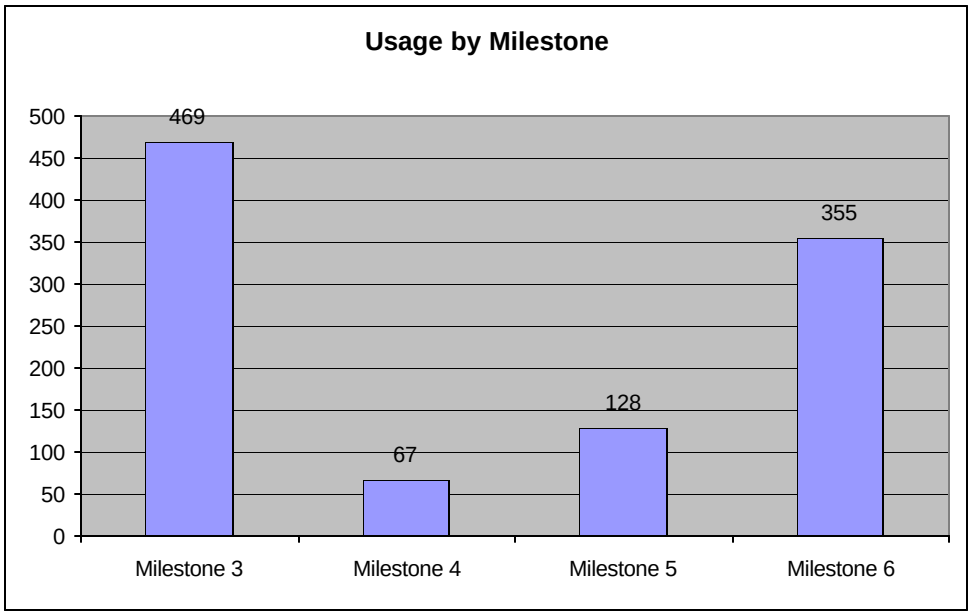


Figure 6: Iterative design in ECoDE, as evidenced in three of eleven groups.

This pattern indicates that students were not satisfied with their original design, but were able to use ECoDE to make changes along the way. These are the students we can help the most. Looking at these designs, we see that students had similar success with those from the first pattern who appeared made less modification as the semester progressed.

It may be that ECoDE supports two types of usage patterns. First, we have students who put a lot of effort into the design up front and make small changes to their design over time. For these students, code generation can save time in setting up the initial code stubs, critics can warn students if their new changes are not consistent with the design, and Ectrospace allows team members to view the design as a reference for the code development process. Second, we have students who also put a lot of effort into the design up front, but make additional significant changes as they learn more about the program from writing it. For these students, the diagrams and click-and-drag style of updating are very powerful mechanisms for making significant changes to their design. The effectiveness of the visual interface was one of the major complements we received in the exit surveys, despite the annoyance that the layout is not maintained once the application is closed.

From this information, we realize there is a need to concentrate on making the diagrams more effective so that students are empowered to make the changes they need to as quickly and easily as possible. Saving the UML layout of the diagram is a needed improvement that students consistently referred to in their exit surveys. Here is why this is so important: making significant changes are easiest with the visual, click-and-drag interface, but spending time to layout the classes/CRC cards in the diagram every time students open the program adds annoyance and overhead for using the tool. Perhaps this is one reason that most changes that were made in ECoDE were made on a single day when the diagram layout could be set once and used to make all the necessary changes in a batch style.

VII. Conclusions

From the exit surveys, we know that twenty-three students (29%) said they would recommend ECoDE to future CS-2340 students, and an additional thirty-four (42%) said they would it (1) if there were changes made to the program, (2) for certain student populations, (3) if it were used in a language other than Squeak, (4) if explained, or taught, at greater length in the course, and (5) if using ECoDE resulted in extra credit. While some of their suggestions are not possible or even advisable, we can make changes to the program to increase its benefit for student users. Perhaps most notably, we can add the ability to save the layout of the UML diagram and make text input save automatically instead of having to press Alt + S or lose their work. I suggest this future work on implementing the UML layout and text input.

Still, the overall feedback on ECoDE is very positive. Our most noticeable contribution is that students who used ECoDE really felt like ECoDE helped them understand how analysis leads into design and design into programming. Students consistently report a better understanding of CRC cards and scenarios after using our product. We have a lot of room for improvement in the design and program modes, but the main obstacle is time and creativity. We have a useful framework in place, which future researchers can take in whatever direction is most interesting to them.

Finally, to end the semester's work, I suggest the following observations about ECoDE and student research:

- Methods map well onto a single responsibility. Every time a student questioned this assumption, we (the student and I) were able to explain either that the method was necessary or that the responsibility was misnamed.
- Classes map awkwardly to CRC cards if inheritance (parent, child, or both) is involved. It is often hard to figure out where to assign responsibilities if both a class and it's subclasses are included as CRC cards.
- We need to differentiate between arrows at analysis time and arrows at design time.
- We need to find a way to separate UI classes in the design without losing the connections back to the model. It is nice to see what parts of the model hook into the UI, but these extra classes can make the diagrams very cluttered. One solution would be to provide view filters on the diagram so the user could choose whether or not to see UI classes.
- We may want to differentiate "design methods" from "program methods". If we can "update design from code," we may add several methods into the code that are not important enough for the design. An example of this is a parser that has many "shift-reduce" type methods that just clutter what is really being decided in a design.
- To make this project successful, we need to focus on the TA's who grade with it. It took a little while for them to come around, and students noticed this: "It isn't a good sign when the TA's and students agree that part of a tool (ECoDE UML) is more trouble than it is worth."
- Keeping the research process anonymous only goes so far. When looking at designs, it was hard NOT to know whose design I am analyzing.
- Students like to complain. See the responses in the appendices.

VIII. References and Acknowledgements

Gray, Kathy Arnold. "ECoDE, Ectropic Collaborative Design Environment: A Study in Helping Students Learn Object Oriented Software Design." Graduate Student Paper. Atlanta, Georgia: College of Computing, Georgia Institute of Technology, 2002.

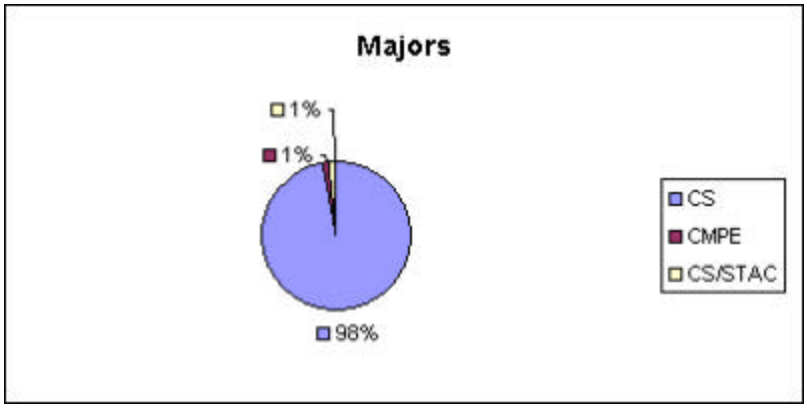
Thanks especially to my adviser and research administrator, Dr. Spencer Rugaber. I have enjoyed working with Dr. Rugaber and hope this report aids in his continued development of the Ectropic design software.

Thanks also to Beth Godbee without whom I would have never completed this project. Thanks for her help with survey compilation, analysis, and final editing.

Appendix A: Background Questionnaire Results

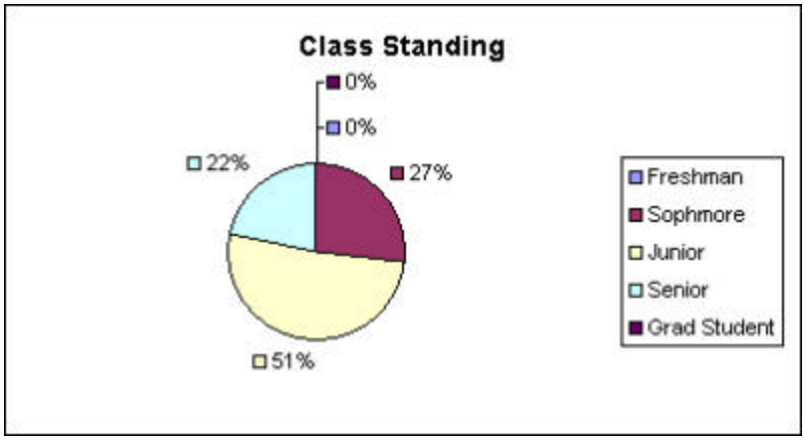
1. Major

CS	79
CMPE	1
CS/STAC	1



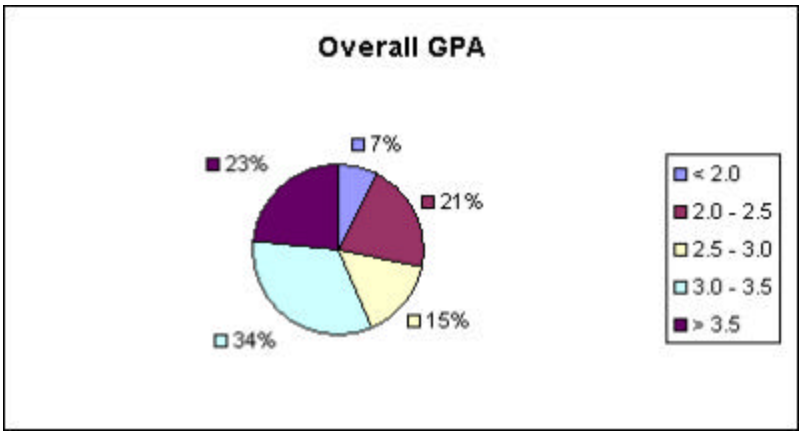
2. Class Standing

Freshman	0
Sophomore	22
Junior	41
Senior	18
Grad Student	0



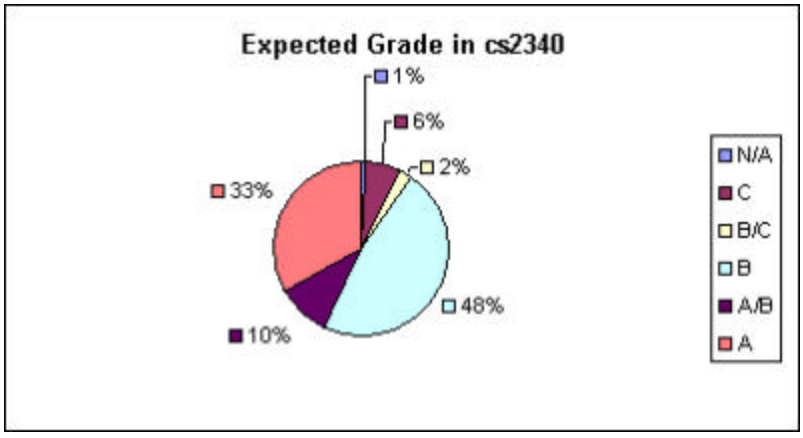
3. Overall GPA

< 2.0	6
2.0 - 2.5	17
2.5 - 3.0	12
3.0 - 3.5	27
> 3.5	19



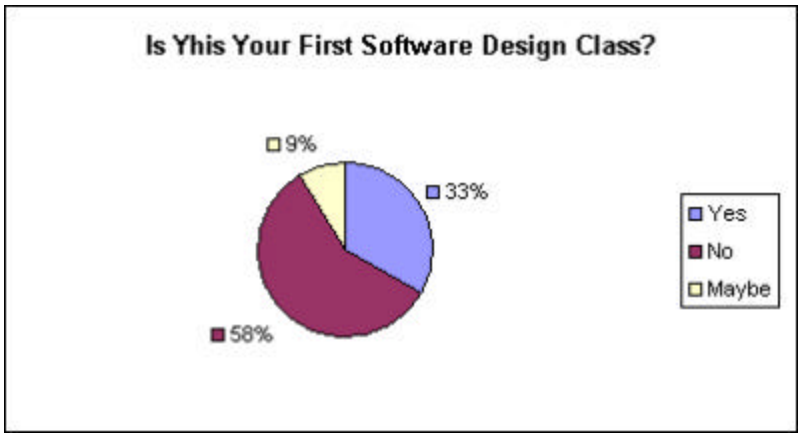
4. Expected Grade

N/A	1
C	5
B/C	2
B	38
A/B	8
A	27



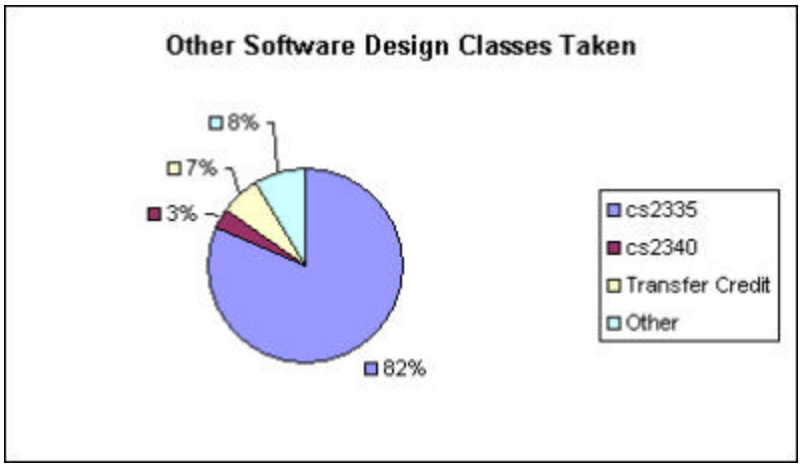
5. Is this your first software design class?

Yes	27
No	47
Maybe	7



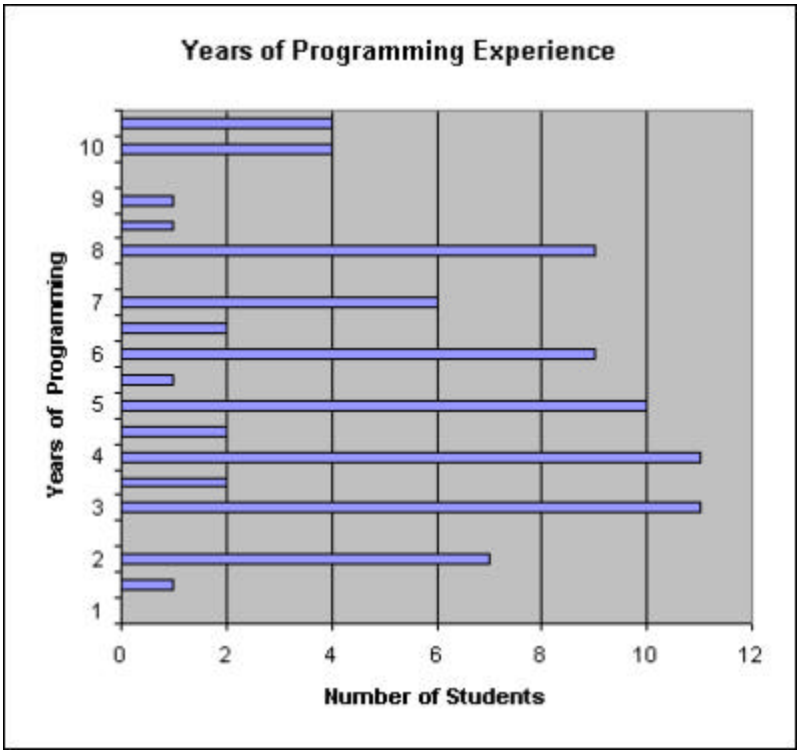
6. Other software design classes

cs2335	48
cs2340	2
Transfer Credit	4
Other	5



7. Years Programming

1	0
1.5	1
2	7
2.5	0
3	11
3.5	2
4	11
4.5	2
5	10
5.5	1
6	9
6.5	2
7	6
7.5	0
8	9
8.5	1
9	1
9.5	0
10	4
> 10	4



8. Programming Languages and experience

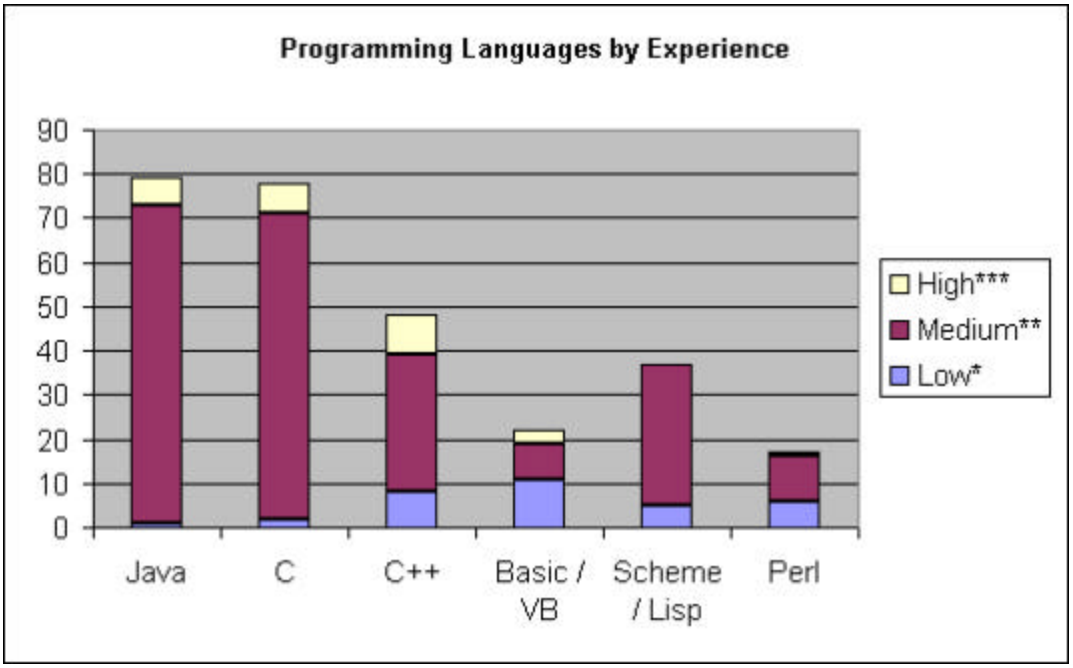
	Low*	Medium**	High***
Java	1	72	6
C	2	69	7
C++	8	31	9
Basic / VB	11	8	3
Scheme / Lisp	5	32	0
Perl	6	10	1

***** Low experience is listing a reading knowledge only.
****** Medium experience is determined by having a class taught with that language and/or work experience on code that is less than 10,000 lines.
******* High experience is determined by work experience and working on code that is 10,000 lines or longer.

Other Languages Mentioned

SQL	9
Pascal	8
PHP	7
HTML	7
Python	4
TCL	4
Assembly	6
Fortran	3

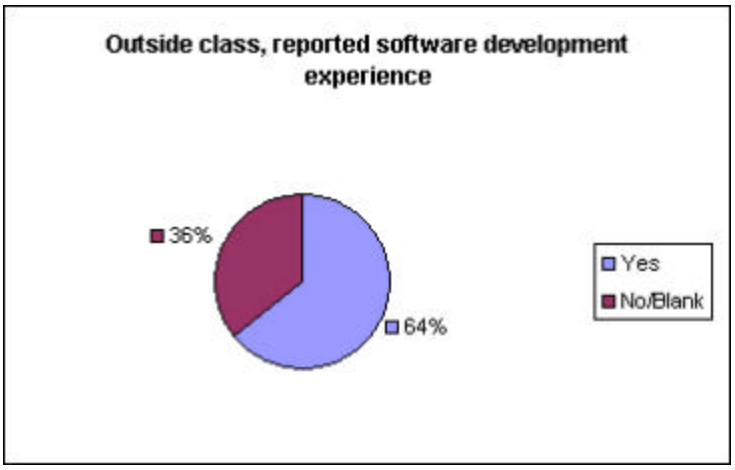
More Languages (<=2 responses)
Lotus Script, Borland Delphi, Fox Pro, ADA, Power Builder, Progress, Logo, and Cobol.



9. Outside class, reported software development experience

Yes	52
No/Blank	29

See Appendix B for individual responses.



Appendix B: Written Responses from Background Questionnaire

Question #9: Briefly describe any software development experience you have had outside of a classroom.

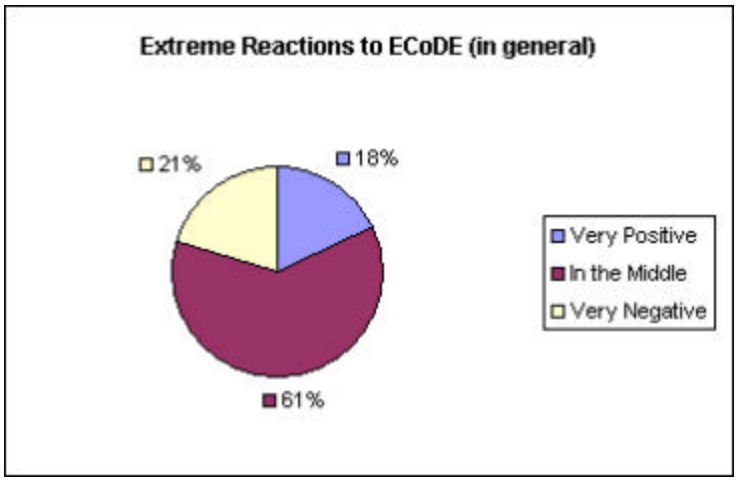
- Co-op with a software development company who is a Microsoft partner
- Co-op position developing Java web applications
- Have worded as a DoD contractor for the last three semesters, developing software for aircraft Navigation systems
- Have four years experience developing C++ object-oriented 3-D games
- Work as in a software development team at GTRI
- Have programmed by own email client and task scheduler in Java
- Have written boring programs for my own amusement, maybe some bad TI-83 games
- Have implemented a test harness for testing different rebuilt tools; also implemented a tool to live sync multiple databases and create one content out of them
- Serve as a project manager for an open source project "Gallery," have extensive object-oriented programming for benchmarking sorting algorithms for high school science project
- Do web programming at my co-op job, as well as some other small programs (mostly scripts)
- Do some C++ and TI-ASM personal programming
- Co-op at a company every other semester and work to develop video-conferencing and remote desktop sharing software
- Currently designing a game engine for simple 2-D games; I've worked on several other projects for fun, but not many large scale ones
- Worked with an SAP contracting firm last semester
- Have co-op experience
- Worked with autograders for CS1322 (as a TA) and do recreational programming projects
- Worked at a commercial software company for three years, designing and writing scripts
- Have done rudimentary high school programs for fun
- Worked on developing image interface in Java as a UROC project
- Co-op and do personal projects
- Have made a pong game in Java by myself (yeah!)
- Have done extensive development of media software (Java) and flight control software (C) for Gulfstream Aerospace Corporation
- Worked as a systems analyst/architect for almost six years
- Worked as a self-employed programmer for five years
- Create some Perl script at the workplace
- Do freelance work in network applications in UNIX, database work, and game development
- Distributed development experience among employees both in the same building and in another location
- Designed a program that would allow project managers to easily print out technical information on air handling units that were installed on their projects
- Co-op for IMB and work in Pearl with ADSI to create an automated web application for division servers, in Java web application to do CD distribution, and in C/C++ to generate automated testing lab networks
- Write hundreds of simple Tel programs for co-op job; planned and developed advanced Java programs for co-op job

- Attempted to learn and use SQL and Perl programs for my co-op job
- Have programmed recreationally since middle school and participate in free software programs
- Works as a paid undergraduate researcher, designs personal applications, and writes scripts for campus organizations
- Works as a palm software developer
- Program full-time, suing TCL, PL/SL, SQL, and Java script
- Have worked as a web developer and implemented small applications
- Have written a chess program in C++ for fun
- Uses Webpro EX (web-based ordering software) and External Lands (MMORPG) at work
- Co-op semesters at construcw@re.com, an emerging solutions company
- Have worked multiple jobs involving Perl and web programming
- Have developed a typing tutor for a one-handed keyboard in Java and a video capture library for Linux in C; also do catch-all programming at home
- Wrote a raytracer in Java
- Do game development on my own and programming as part of my job
- Developed web applications in Lotus Notes / Domino
- Have a co-op job in development and redesign of flagship GUI; also worked a web developer job in high school; do PHP consulting; have published a book
- Have developed image conversion modules for a large software company as a co-op
- Have done C coding since twelve on Internet daemons, as well as an online HTML scripting language, a compiler number library; in assembly, have written a large-size integer library
- Designed and implemented production tracking software in a production line at a mat factory I used to work for; no teamwork, however, so I was alone in it.
- Have been a professional developer for the past twelve years and returned to Georgia Tech to complete my BS in 2002
- Interned at a company a couple of years ago where I wrote several Perl scripts, mostly to update many files at a time on a web server; it was not any sort of structured design process. I was assigned the task and wrote the script on my own
- Taught myself all of the fifteen or so languages I know with the exception of squeak; mostly just searching for code/info. on the Internet
- started programming BASIC when ten; worked last year at Intel writing low level drivers and performance monitoring programs

Appendix C: Exit Survey Results

Many of the exit surveys could be categorized into extreme dislike/apathy toward ECoDE or Squeak, and others could be categorized as really positive toward ECoDE. Here is the breakdown calculated as extremes of opinion.

Very Positive	14
In the Middle	48
Very Negative	16

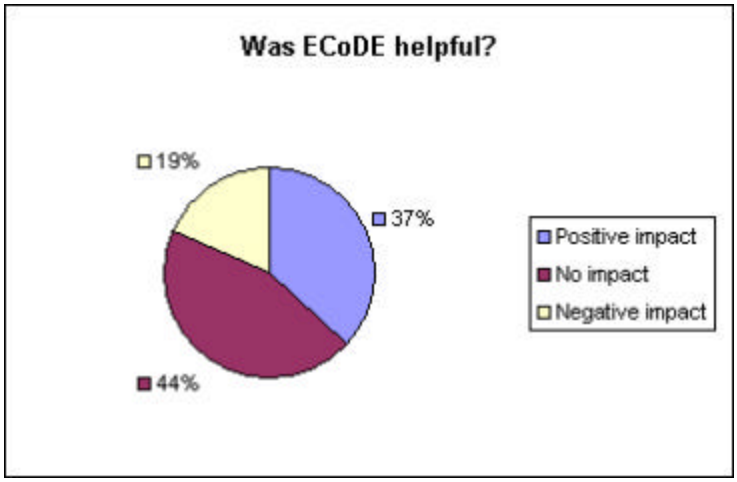


1. How, if at all, did ECoDE aid your learning and understanding of software design?

See Appendix D for written responses.

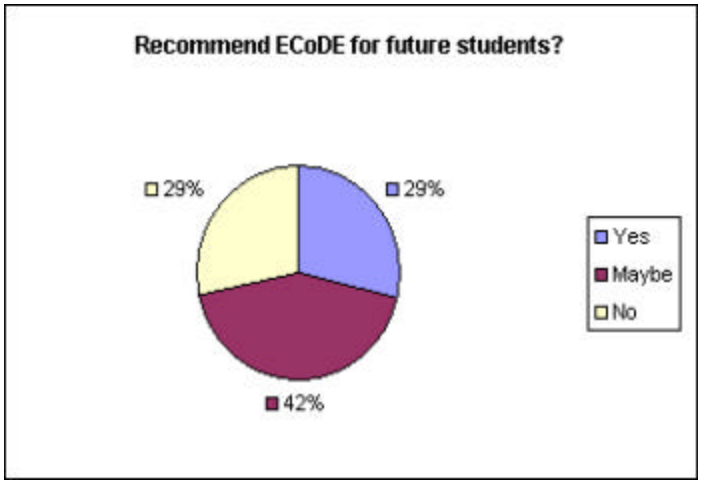
2. To what extent did ECoDE help your tem with the semester project? Please explain.

Positive impact	30
No impact	36
Negative impact	15



3. Would you recommend ECoDE to future CS 2340 students (if they had a choice to use it or not)? Please explain your answer.

Yes	23
Maybe	34
No	23
- Conditional Upon?	



4. What feature of ECoDE did you like the MOST?

See Appendix D for written responses.

5. What feature of ECoDE did you like the LEAST?

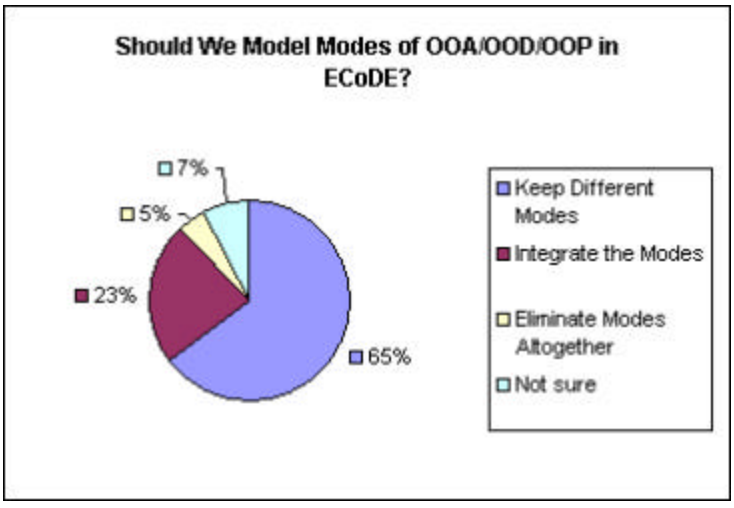
See Appendix D for written responses.

6. If you could add only one feature to ECoDE, what would it be?

See Appendix D for written responses.

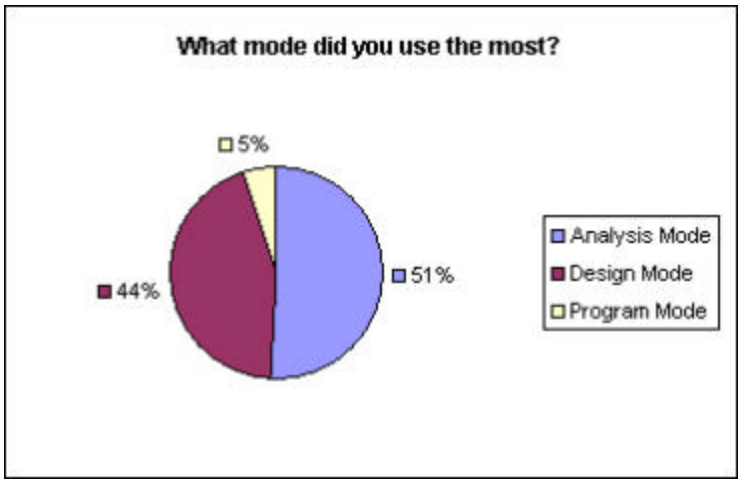
7. Do you feel that ECoDE should include modes or provide all the tools regardless of mode? Please explain.

Keep Different Modes	52
Integrate the Modes	19
Eliminate Modes Altogether	4
Not sure	6



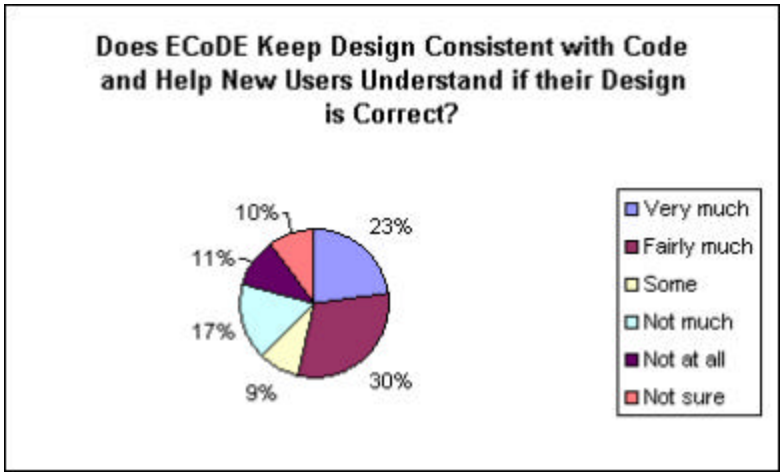
8. What mode did you use most in ECoDE?

Analysis Mode	41
Design Mode	36
Program Mode	4



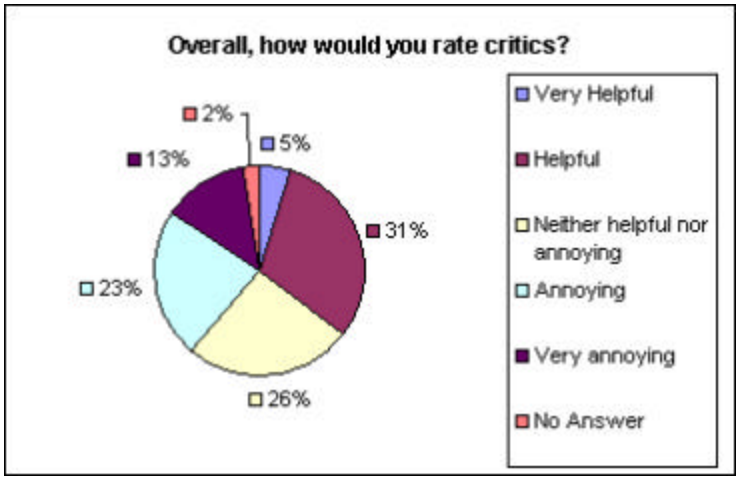
9. To what extent do you feel that critics accomplish their goals of keeping design consistent with changes over time and help new users figure out if they are doing things correctly.

Very much	19
Fairly much	24
Some	7
Not much	14
Not at all	9
Not sure	8



10. Overall, how would you rate the critic mechanisms?

Very Helpful	4
Helpful	25
Neither helpful nor annoying	21
Annoying	19
Very annoying	11
No Answer	2



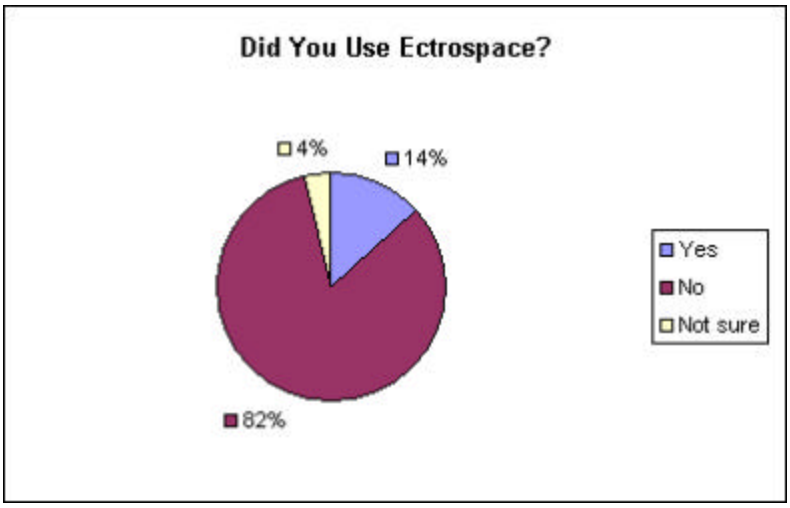
11. If you could add one critic, what would it be?

See Appendix D for written responses.

12. Did you use Ectrospace? If you used Ectrospace, how helpful was it and why?

Yes	11
No	67
Not sure	3

See Appendix D for written responses.



13. What can we do to improve Ectrospace?

Notable responses:

Seven(7) people said:
Allow multiple people to edit different parts of the same file at the same time (i.e., include both their changes when they submit instead of just writing over the previous one) OR let us share the code among coders better OR add real-time support

Two (2) people said:
Add ability to turn off critics

Three (4) people said:
General revising or improved interface

Appendix D: Written Responses from Exit Survey

Question #1: How, if at all, did ECoDE aid your learning and understanding of software design?

- Positive comments:
  - Made the design process easier
  - Forced the use of CRC cards
  - Eased the design process
  - Gave some visual aid to the design process
  - Helped illustrate how closely form follows design
  - Helped me connect the analysis, design, and programming parts better
  - Made organization easier
  - Provided a structured and organized way to help design structure
  - It helped bridge the gap between design and implementation
  - Aided in providing a design picture by helping give a view of the design
  - I could see how phases from A to D to P were related and independent
  - It didn't lead to me learning more about software design, but it made the design process easier
  - It allowed me to visualize the relationship among analysis, design, and implementation phases
  - Benefited me when creating scenarios and assigning responsibilities
  - Makes UML and design easier because don't have to deal with general programs like Visio
  - Helped me identify all the steps necessary to effectively design software
  - Aided in the acknowledgement of the association between CRC and methods in class
  - Helped us to organize and focus on the design aspect of the class
  - Increased understandings and helped show the relationships and roles of objects, CRC cards, and scenarios
  - The organization of CRC and scenarios was very helpful
  - In understanding the different aspects required and different section helps in a complete software design
  - Facilitated the relation between responsibilities and methods
  - It helped sell the CRC card idea
  - I think it aided in good procedures, but did not really aid in my understanding
  - Had a better vision about how the program will be structured
  - Maybe if we had used it more, we would have learned more from it
  - It certainly helped in setting founding ideas down on "paper"
  - It helped structure and centralize formal design
  - Helps create an easier way to design using UML and CRC cards
  - It helped me learn exactly what is a CRC card and how they figure in the design process
  - It helped me to understand how the phases of design are supposed to work
  - It made the process of CRC, scenario, and UML creation feel more unified
  - ECoDE helped have a UML setup that could be easily altered if needed
  - Made distinction between OOA/OOD/OOP clear
  - ECoDE made it very easy for us to analyze and design our project

    - Helped learn difference between OOA, OOD, and OOP
    - Its strongest aspect was how it forces one to conform to one's design and aided in my learning
    - Somewhat learned differences between OOA, OOD, and OOP
    - Helped to illustrate the difference between design and analysis
    - It allowed the connections between classes to be more easily understood
    - Seeing the entire design process in one phase helped me to see the relationships between the different components and steps along the way
    - It was helpful
    - ECoDE provided an example of a useful program that had been written in Squeak
    - ECoDE itself provided me with an example of how a good design does not necessarily make good software
    - It helped force us to conform to the standard method of writing CRC cards
- Negative comments:
    - Too many problems or bugs
    - Too much aggravation
    - Scattered in functionality
    - It didn't!
    - It didn't help, sorry. We did our design outside ECoDE and then added it.
    - Did not help me in learning
    - After using Rational Rose and Visio for representing designs in three previous classes, ECoDE didn't aid learning or understanding at all
    - Didn't help much because I took 2335 and 2340 (got a "D") already
    - Didn't help
    - It was annoying more than anything; it's easier to type CRC cards into Word
    - It got in the way more than anything
    - As I didn't really learn anything new in this class, I can't say it impacted my learning
    - I really didn't learn anything from it; it's just a tool
    - Proved to be a burden in the requirements, rather than an aid
    - Overall, it integrated design and implementation. I think it was a great idea, but should be implemented in something other than Squeak. Squeak is slow and pathetic
    - Did not play a significant role in my learning or understanding of software design
    - Didn't really help with my understanding
    - Did not help me learn

Question #2: To what extent did ECoDE help your team with the semester project?

- Positive impact:
    - Decreases ambiguity about what teachers want us to do
    - Helped us organize our design and thought process
    - Rapid, quick design and easy to see changes immediately
    - Provided a uniform cross-platform mechanism for exchanging design data
    - Helped me with the design documents
    - Helped keep our design organized
    - Forced me to spell out the correct design and how to extend it
    - It made the UML easy, among other things

  - It made visualizing class interactions easier and was a simply way to compose diagrams
  - Without ECoDE, which automatically placed classes in almost the right place, laying out the design would have been more difficult. Having it simplified things
  - Helps to get concrete class and method names out ahead of time
  - Helped to automate certain aspects of the design process; also provided a centralized location for design elements
  - Helped us focus on learning design
  - Made it easier for the documentation guy in our group to do the documentation
  - It helped make sure we covered everything in our CRC cards and scenarios
  - Help in designing; had a clear idea from the beginning about the method sequences
  - It made it much easier to create CRC and UML diagrams
  - Helped us share rules in UML and allowed us to integrate responsibilities
  - Encouraged better design; let us focus better
  - A great learning tool
  - Since I was able to see classes, it reduced the busy work of having to write out methods and new classes in UML
  - Standard format of the design phase enabled us to stay on the same page with each other (i.e., aided in group work)
  - Forced us to organize our thoughts and design decisions; helped to have a tangible record of our decisions
  - It allowed us to keep our design more consistent than it would have been otherwise
  - The auto generated UML diagram from the CRC cards and class definitions were helpful
- No impact:
  - We were required to use ECoDE for the milestones, but we didn't really refer back to it much. We did class diagrams in Visio and mainly followed them.
  - I could have designed a better project working on my own.
  - Although it is a good idea, I don't feel like we were able to fully gain from its benefits. It did help, however.
  - We had to redo everything anyway and although it was nice to have everything in one place, it was tough to use it in Squeak due to errors and various problems.
  - It helped with understanding, but not enough to offset the difficulty in using it correctly.
  - I've taken this class before and already know how to do these things.
  - It helped as described in #1 (creating scenarios and assigning responsibilities), but was a bit tedious to keep up to date to the actual implementation
  - Overall, ECoDE became useless because of our heavy reuse
  - The only thing we used was the CRC cards
  - We had a decently good design and, thus, there was not 'extra' benefit from ECoDE
  - The UML section needed some help, but the other parts were good
  - I think ECoDE seemed to be more of a chore to us, probably because we did not design our applications completely with it
  - Made UML a slight bit easier, though I still opted for Visio
  - We basically worked with the code first and then put the design into ECoDE
  - CRC cards and scenarios are interesting, but might be done best on paper

    - Although it seemed an unnecessary requirement, it did not adversely affect the team
    - It was helpful, but at times unreliable so it was not used as much as it could have been
    - Marginal impact at best; collaborative ability lacking, making team-based development and design difficult
    - I probably would have designed just as well with pen and paper
    - I think the same effect could have been achieved in another design program
    - Visio is better
    - We simply did what we needed to do in it
- Negative impact:
    - Waste of time
    - Too difficult to learn how to use
    - We didn't have to use it often enough
    - Waste of time; existing tools do a better job with design
    - Just something more I had to do
    - Took time, had bugs
    - Many bugs with version we had. Could not upgrade to a newer version because patch failed, and we were using Macs.
    - Having to do stuff in ECoDE and then use another tool (Dia) to draw good UML diagrams was a wasted effort
    - The documentation requirements for the project were so extensive that being bothered to update ECoDE every two minutes really slowed things down.
    - UML was confusing and caused problems with design; also ECoDE was unstable sometimes
    - Kludgey, unwieldy; lot of functionality/responsibilities that had no need to be detailed in ECoDE except to satisfy critics
    - Was not helpful and, thus, a burden to work with
    - We used because we had to for CRC cards, but that was it
    - Everything that could be done in ECoDE can be done quicker on paper
    - The benefits we gained in the CRC card/design phase was outweighed by the time spent fighting ECoDE during the UML phase
    - Everything that it does can be done better using already popular programs like Visio
    - We always lost points on ECoDE because we never figured it out
    - Due to the lack of collaborative interaction (i.e., inability to merge the disparate ECoDE files), it was typically an afterthought
    - It was difficult to figure out ECoDE from scratch and would often get confused by missing data I thought I had
    - Took time away from the project

Question #3: Would you recommend ECoDE to future CS-2340 students (if they had the choice to use it or not)?

Thirty-four students said that *maybe* they would recommend CS-2340 under the following conditions:

- Changes in the program
    - Bugs fixed
    - Better UML support

    - Forced associations
    - Ability to re-import classes
    - More user-friendly design
    - Easier access to CRC cards
    - UMGL diagrams work faster
    - Better interactivity and speed
- For certain student populations
    - Students who have taken CS-2335
    - Students who have some experience with software development
    - Based on students' performance in the class
    - Based on students' previous knowledge with design
    - When working in teams or collaborative environments
- If not using Squeak, but another programming language
- If explained, or taught, at greater length in the course
- If given extra credit for using ECoDE

Question #4: What feature of ECoDE do you like the most?

- Scenarios linking to CRC cards, which are linked to UML
- Each tool within ECoDE is particularly useful
- Good comprehensive design
- Easy-to-use interface
- Ease of use
- Export code stubs from design
- CRC cards and scenarios
- The save our design button
- UML design features
- The integration among modes
- The UML display
- Collaboration
- Simply that it was integrated into Squeak and provided a fully electronic design mechanism
- Diagram UML layout
- It allowed me to have the other team members do all the work
- Import function
- Ability to import classes and pull together code
- UML
- Import feature
- The CRL cards – connect and you can see how they relate
- How easy it was to switch back and forth between the different steps of design
- Switch between different modes
- Automatic UML diagrams
- CRC cards
- The ability to immediately add a class to the program and import all
- The ability to add a class to the program and import all of its corresponding code
- It is automated

- Class diagram generation
- Production of UMLish design for your input
- Ease of use
- Code production
- Turning the CRC cards into UML automatically
- Auto code generation
- Graphical UML representation of classes with links
- The combination of the three modes
- The linking of scenarios to CRC stuff
- CRC cards
- Integration of UML in Squeak
- CRC cards
- Ability to turn responsibilities into methods UML design
- CRC cards
- ECoDE's ability to have everything needed in one place
- CRC cards
- The CRC cards were helpful in assigning class responsibilities
- Generating UML
- Automated UML drawing
- Auto generate stub code
- In the design diagram, the green arrows showing the flow of methods that will be executed to perform a job
- The ease with which I could create CRC cards and then carry them over to UML and even to code
- CRC cards
- Importing existing classes instead of having to retype methods, variables
- Easy UML static diagram
- The modes
- Import
- Encouraged better design
- Class relationships
- Auto-generation of code
- Squeak code generation/integration
- CRC Cards
- Having some areas create content in others (in CRC responsibilities defining functions in UML)
- The colors
- CRC cards
- CRC cards
- Design critics, although they could be more robust
- Team working ability
- Graphical representation of the classes
- Tight integration between OOA/OOD/OOP; it's nice that there's a "forced" relationship between design and resulting code

- Class diagrams
- Scenario entry
- Ability to list requirements
- UML code generation
- CRC codes and UML generation
- Auto generation of the skeletal code proved most useful for tedious code segments
- CRC cards
- It took all of the CRC cards and responsibilities and generated the class diagrams for us
- The fact that it could easily switch between the different phases of the design process (analysis, design, and programming)
- Auto-generated UML
- Quit
- Fairly intuitive interface
- Writing CRC cards was faster

Question #5: What feature of ECoDE did you like the least?

- UML was just too weak to be helpful
- The program is in Squeak
- Screen could get somewhat cluttered
- The pop-up box with missing design stuff
- That its use was forced (required for the course) and not optional
- Small bugs here and there – for example, disappearing relationships between classes in UML
- UML's auto placement
- The interface
- Being forced to state design in a specific format; I didn't get to choose
- UML is totally broken and useless
- Error messages for goals
- Sometimes minor details of the program were difficult to figure out, and ECoDE often complained about little things before allowing the user to move on to the next stage.
- The UML design was a pain; I prefer Visio
- The user interface (UI)
- It was in Squeak
- It was implemented in Squeak
- Generating method names for concatenating a responsibility
- UML diagram – it always showed all classes in top left and if you click the button too
- When you did not include all the methods of a class or attributes, it would give you an error whenever exiting the program part of ECoDE.
- UML layout didn't work
- Class diagram layout
- The UML layout doesn't work, and UML positions are not saved
- Class diagrams were just a bunch of boxes on top of each other
- CRC
- Auto graph generation
- The input for the CRC cards was poor in my option – too clumsy

- It's in Squeak
- Working with UML
- Poor UML class diagram display and control
- UML diagrams never looked right to me
- The ever-annoying critics
- UML class diagrams
- The interactivity of the UML
- UML diagrams
- Too large and error problems
- UML
- The diagram feature, because it was difficult (even possible?) to save the arrangement of the classes
- CRC cards – hard to do the input, too many clicks
- ALT+S function in Squeak – we kept losing saved information, which slowed us down
- UML – ugly connectors
- The design diagram layout doesn't save; with critics, many have a button in the main UI and should not be center aligned
- I don't like the CRC responsibilities map to class methods; this seems out of place to me
- Strict scenario-based design
- Scenarios
- Having the different views seems discontinuous
- Some of the requirements for linking CRC to other parts of the program (how strict it is)
- The diagram was very primitive
- Updating all phases for each milestone
- Critics
- The UML diagram apparently isn't save (at least the location of the boxes)
- The UML diagram that you couldn't save the layout on
- Rigid relationships between CRC cards and classes
- UML
- The UML view
- Hate having to ALT+S to save; I would forget and lose data
- Diagram takes time to open up and freezes the computer
- UML class diagram
- UML diagramming, needs to save layout
- Auto placement of UML diagrams
- Confusing boxes everywhere; normal Squeak problems; ALT+S, and out-of-window focus
- Ectrospace is only minimally helpful
- When you go to view UML diagrams, it's not automatically laid out well
- CRC cards, the whole point was to have something tangible
- It was in Squeak
- CRC cards were hard to use, and there were too many critics
- Critics need to be modified to be less strict on design adherence
- Class design window
- The fact that it didn't arrange the classes neatly

- The design would always reset itself, and the classes on top of one another. Once the class system got large enough, it took some time to arrange the class diagram in a meaningful way
- It's in Squeak
- The unchangeable function name generated from the class "responsibilities"
- Ectropic design start
- Ugly coloring
- Layout button on UML didn't work

Question #6: If you could add only one feature to ECoDE, what would it be?

- Better UML
- Allowing multiple ways to edit design
- Get rid of pop-up boxes
- A better tool for class diagrams
- A user interface
- Ability to save UML designs
- Code generation
- It should make ice cream or code for you!
- Auto update
- Just fix bugs – no new features
- Add the ability to import UML or the ability to use case/sequence diagrams
- A complete UML creator, rather than a partial one
- Importing UML from Visio
- Allowing methods to be chosen by first selecting class and then method
- Don't write it in Squeak
- Not in Squeak
- Expand UML tools
- UML viewing would be nicer
- The ability to merge files together so they come together as part of one big design
- Better collaboration among team members
- Maybe a save button that accepts all fields and saves to file
- Auto layout of class diagrams
- Import existing code
- Code generation
- Ability to same UML and perhaps export the pdf/ps specifically for UML
- CVS-like ability
- Editing UML directly
- Making critics disable-able
- An auto sorting button to make the UML diagram neatly display
- Change version control to link classes to actual design
- The ability to keep UML properly spread and/or auto spreading to clear displays like Mac does in Expose
- Save UML
- Organize the diagram and remember where the boxes were located
- Need better CRC cards

- Working CVS, working UML, working save
- Make the critic not a menu, but some text; add the ability to specify inheritance starting from CRC cards
- Write it in/for a useful language, like C++ or Java, or pretty much anything but Squeak
- Read class diagram editing
- Better UML class diagram display
- Option to not show every new collaboration as a line
- UML sorting; more features in UML; better layout algorithm
- Forced associations
- Save the class box positions in the UML diagram
- Place to specify method name with responsibilities
- New color scheme
- The stop Squeak from sucking feature (maybe make ECoDE in Java)
- Make UML diagram part creation and separate thing like Visio
- Fix UML class diagram
- Saving the layout of the UML diagram
- Ability to generate diagrams from code
- Get rid of ALT+S
- Real-time collaboration for all three modes (OO-A/D/P)
- Runnable outside Squeak environment
- Make it in Java
- Add a UML generator, which leaves out code and makes UML code
- Add a turn-in feature
- Abstract classes in the class diagrams
- Something to arrange the class diagrams more neatly
- Improve intelligent set up of the class design
- Add click and drag option in UML editor
- Fix UML layout
- Needs UML layout
- Keep arrangements of UML
- Make UML layout button work

Question #11: If you could add one critic, what would it be?

- Whether scenarios have been accepted
- The program should yell at you if methods become big and too monolithic
- PMD checks
- Maybe optional sunit support to keep track of tests
- Responsibility doesn't have a corresponding method
- UML critic
- Missing method comments
- Some design concepts that the TAs could later take off points for (ex: a class with too many responsibilities)
- A critic that examines calls to other objects in the design
- Check for methods in the design that are not in the code (at least a stub)

- The type of critics that gives everyone a "two thumbs up"
- Add critic for high coupling

Question #12: If you used Ectrospace, how helpful was it and why?
- Very helpful
- Just played with Ectrospace a little
- Helpful because it gave you a booklet at your disposal to help with design
- Not helpful because worried that it would overwrite the existing work
- It was helpful
- Helpful for online data chains
- Yes, I used Ectrospace. I think it was a little useful for Milestone 3, but it was too insecure and just not nearly up to par with CVS.
- We briefly attempted to set it up but gave up – it appeared it's just a mechanism for transferring files?
- Make it integrated with CVS or some kind of version control
- It was cool for having a central place for our ECoDE project

Question #13: What can we do to improve Ectrospace?
- Allow multiple people to edit different parts of the same file at the same time (i.e., include both their changes when they submit instead of just writing over the previous one)
- Let us share the code among coders better
- Add real-time support
- Allow all group members to edit at the same time
- Add ability to turn off critics
- Needs revising
- Needs improved interface
- Needs ability for multiple changes to be made from different users at the same time

Question #14: Other comments?
- Using ECoDE was frustrating
- It is nice, and so is the TA who developed the tool
- Overall, an excellent tool
- Needs to be beta-tested
- Make it more student friendly
- Just keep polishing it and get rid of the minor bugs
- It seems like it could be really helpful, but my team didn't really use it much
- Needs a cooler-looking interface, but then I hate Squeak at this point
- Overall, a good system; just needs to get bugs out
- Once its many problems are fixed, it might be more of a help than a hindrance. However, I doubt that it will ever be more useful than the existing methods of design. That's why we didn't really use ECoDE.
- I thought it was a pretty thoroughly effective diagram tool, but I would like a way to save the UML layout.

- ECoDE is a great idea, but it would be more helpful with more widely used languages like C or Java – not Squeak.
- I think the design tool is a good idea, but using Squeak makes it totally not worth it.
- The way responsibilities and designated methods are paired needs more work.
- Ecode is not a bad program for its purpose; it just needs to be cleaned up a bit. The UML did not generate so well, and renaming methods and responsibilities could become tedious because of the menus.
- Fix the major bugs and usability issues before adding new features
- I think it has potential, but shouldn't have been mandatory for the class. Still needs improvements, like remembering layout of the class diagram, maybe also generating sequence diagrams based on scenarios.
- Having to accept all the fields is annoying; please consider having a button to accept them all
- We had issues with cross-platform and across different ECoDE versions compatibility of the .ect files at one point. If this has not been fixed, then it could be nice to do so.
- Honestly I think learning to use a "real-world" tool would have been better. Skills of ECoDE go nowhere.
- Should be completely optional until a solid release is made
- It's mostly good and obviously well intentioned, but at times it comes off as being clunky. This probably stems from the fact that it is in Squeak (I'm not a big fan).
- A good idea that's been crippled by being stuck in an environment that is shy of completely usable: write is in something other than Squeak, and I'd use it regularly.
- It isn't a good sign when the TA's and students agree that part of a tool (ECoDE UML) is more trouble than it is worth. This should be fixed ASAP, and students shouldn't be required to use a broken tool.
- Provide more training to students; spend more time building the significance of CRC and relation to OOD specifically in ECoDE
- My main complaint is with the UML diagram
- It's a good tool
- I think it has good potential and is moving in the right direction
- Again, I would find an algorithm to organize the diagram; it would make things much better
- CRC cards need work
- It's a good idea; it just had its quirks that made me want to bash my head in – just like everything else in Squeak
- Needs work, too much functionality is flaky
- It has potential, especially for younger people first learning OOP
- It's nice; I hope to use it in other courses; very helpful
- Overall pretty useful, but would be much more so it weren't written in Squeak
- In designing object-oriented systems, I find it easier to model the problem in terms of actions and components at first, and scenarios are part of it, but not the driving force.
- I think it should be scrapped in favor of smoother design tool
- I think it is a helpful tool, but it seems just short of an effective application; add more interactivity
- I don't see how ECoDE is different from any other design tool; I would expand on the modes and critics and work on making it scale up for larger projects

- We often did our design outside of ECoDE because the limits of responsibility descriptions for CRC cards were restrictive and the auto-generated functions did not match our naming conventions
- I think once it has been more fully developed, it'll be a very useful tool, but in it's current state, there are too many missing/broken features
- I would like ECoDE to be more fluid to better match the realities of design; perhaps only enforcing critics when moving to the final phase would allow more design possibilities
- I feel that ECoDE at its current state is just a useful place to put CRC cards electronically. The program could become more useful with an improve UML interface and less bugs.
- Very interesting program, but Squeak sucks. WE never integrated it into our workflow. The CVS functionality and UML fixes will be nice.
- I felt very limited with ECoDE; I felt like I could never document the way I would have wanted because I had to do one thing (dumbed down) for analysis, but then trying to do the design portion as suggested would end up ruining the analysis.
- ECoDE is promising, but a lot of work needs to be done to make it useful
- I would rather have setup the UML class diagram by hand, rather than having to keep using the button, which sometimes made unreadable diagrams
- A huge need to improve on diagramming; I believe this is a very easy way to convey a significant amount of information in a very small space
- Maybe create a help file
- I wish it could go from code to UML and vise versa
- ECoDE is a good tool for teaching OO design, analysis, and programming, but as a development tool, it lacks the intuitive design features to be a great help for the projects. So, if the goal is to teach an idea, then ECoDE is a success, but as a true aid in software development, it falls short.
- Nice interface. Pretty easy to use for anyone with a little knowledge about OOD
- ECoDE seems like a very promising tool; if it could be polished a bit, then it would be very helpful tool in CS-2340
- It still feels a little buggy and could use some refinement, especially to the UML layout.
- It was both helpful and annoying to have new version released during the project; perhaps the software would be better tested by those not using it for graded projects
- Make the GUI less constraining; there's too much clicking to enter info.
- I would have liked to see it outside Squeak, because Squeak put a bad taste in my mouth from the start
- The fact that everything is in one file and cannot be merged prevents multiple people from working on it at the same time

Appendix E: Background Questionnaire

Ectropic Software Design Study: PARTICIPANT #_____

Part 1: Personal Information

1. Major: _____
2. Class Standing:
   - ☐ Freshman
   - ☐ Sophmore
   - ☐ Junior
   - ☐ Senior
   - ☐ Graduate Student
3. Overall GPA:
   - ☐ < 2.0
   - ☐ 2.0 - 2.5
   - ☐ 2.5 - 3.0
   - ☐ 3.0 - 3.5
   - ☐ > 3.5
4. What letter grade do you expect to receive for CS 2340? _____

Part 2: Background

5. Is this your first educational course in software design?
   - ☐ Yes
   - ☐ No
   - ☐ Maybe - Explain _____
6. If no or maybe, please list other courses you have taken in software design:

_____

_____

_____

7. How many years have you been programming? _____
8. Please list the programming languages you know in the following table, and describe your skill level for each. For each language, estimate the number of lines in the longest program you've written. Skill levels are as follows:

1. I have reading knowledge.
2. I have written programs as part of a GT class.
3. I have been paid to write programs.

Language	Skill Level	# of lines in your largest program

9. Briefly describe any software development experience you have had outside of a classroom environment:

Appendix F: Exit Survey

Ectropic Software Design Study: PARTICIPANT #_____

Part 1: Your Impression of ECoDE

1. How, if at all, did ECoDE aid your learning and understanding of software design?

2. To what extent did ECoDE help your team with the semester project? Please explain.

- ☐ Positive impact
- ☐ No impact
- ☐ Negative impact

3. Would you recommend ECoDE to future CS 2340 students (if they had a choice to use it or not)? Please explain your answer.

- ☐ Yes
- ☐ No
- ☐ Maybe - Conditional upon? _____

4. What feature of ECoDE did you like the **most**?

5. What feature of ECoDE did you like the **least**?

6. If you could add only one feature to ECoDE, what would it be?

Part 2: Modes in ECoDE

7. ECoDE has Analysis, Design, and Program modes to represent object-oriented analysis (OOA), object-oriented design (OOD), and object-oriented programming (OOP), respectively. An alternative approach would be to provide all the tools at the same time (no modes) and let you work at whatever level of detail you prefer. Do you feel that ECoDE should include modes or provide all the tools regardless of mode? Please explain.

8. What mode did you use the most in ECoDE?

- ☐ Analysis Mode
- ☐ Design Mode
- ☐ Program Mode

Part 3: Critics in ECoDE

9. ECoDE contains design critics intended to keep the design consistent as the software changes over time (for example, the message "your code contains the method (x) that is not part of the design"). They also try to help new users figure out if they are doing things correctly (for example, the message "responsibility (x) in scenario (y) is not assigned to a CRC Card"). To what extent do you feel that critics accomplish these goals?

10. Overall, how would you rate the critic mechanisms?

- ☐ Very helpful
- ☐ Helpful
- ☐ Neither helpful nor annoying
- ☐ Annoying
- ☐ Very annoying

11. If you could add one critic, what would it be?

Part 4: Ectrospace

12. Ectrospace is our name for the collaboration tools in ECoDE. Did you use Ectrospace? If you used Ectrospace, how helpful was it and why?

13. What can we do to improve Ectrospace?

Part 5: Free Response

14. What other comments or suggestions do you have about the ECoDE tool (please be constructive)?

Appendix G: Consent Form

Georgia Institute of Technology
Human Subject Consent and Release Form

1. Project Title: Ectropic Software Design Study
2. Investigators: Spencer Rugaber (PI) and Jonathan D'Andries (Graduate Student)
3. **You are being asked to be a volunteer in a research study.**
4. Purpose of the research: To relate effective use of the ECoDE tool with quality object-oriented design processes and products. Also relevant is whether or not students would choose to use ECoDE over alternative methods of documenting their designs. This research will require an background questionnaire, data logging, and a post-experiment questionnaire.
5. Approximate number of subjects to be involved in the research and how they are selected: Approximately 110 undergraduate students enrolled in the CS-2340 (Objects and Design) course at Georgia Tech will be recruited as subjects for this research. All students in the course are currently required to use the ECoDE tool. Only those students signing this consent form, however, will be included in the study. All data will be recorded anonymously.
6. Expected duration: The background and post-experiment questionnaires will each take about fifteen minutes to complete. The data logging is a biproduct of using ECoDE for class. The research will be completed by the end of this semester (April 2004).
7. Explanation of procedures to be followed: The participants will complete a background questionnaire asking for a description of their experience with computer programming and object-oriented software design. The data-logging requires access to the participants' log files from using ECoDE. At the end of the semester, participants will complete an post-experiment questionnaire.
8. Foreseeable risks or discomforts: minimal risk.
9. Expected benefits for the participants or others: The tool provides feedback for students on their object-oriented designs. Others will benefit from the use of a structured design tool to scaffold their learning experience.
10. Compensation: Participants will receive one extra-credit point on their final average in the course for their time. If you choose not to participate, there will be alternative ways to earn extra credit.
11. Costs to you: there are no costs.
12. Alternate procedures that can be advantageous to the subject (if applicable): N/A.
13. Confidentiality of records: Log files are anonymized so that it will be impossible to trace the data back to the participant's name.
14. Reports of injury or reaction should be made to Spencer Rugaber (PI) at 404.894.8450. Neither the Georgia Institute of Technology nor the principal investigator have made provision for payment of costs associated with any injury resulting from participation in the study.
15. If you have questions about the research, contact Spencer Rugaber (spencer@cc.gatech.edu; College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332-0280; 404.894.8450) or Jonathan D'Andries

(bane@cc.gatech.edu; College of Computing; Georgia Institute of Technology, Atlanta, Georgia 30332-0170).

Subject Rights

- Your participation in this study is voluntary. You do not have to be in this study if you don't want to be.
- You have the right to change your mind and leave the study at any time without giving any reason, and without penalty.
- Any new information that may make you change your mind about being in this study will be given to you.
- You will be given a copy of this consent form to keep.
- You do not waive any of your legal rights by signing this consent form.

Questions about the Study or Your Rights as a Research Subject

- If you have any questions about the study, you may contact Dr. S. Rugaber, at telephone (404) 894-8450.
- If you have any questions about your rights as a research subject, you may contact Ms. Alice Basler, Georgia Institute of Technology at (404) 894-6942.

If you sign below, it means that you have read (or have had read to you) the information given in this consent form, and you would like to be a volunteer in this study.
Participant's Name (print): _____
Participant's Signature: _____ Date: _____
Investigator's Signature: _____ Date: _____