Teaching Garbage Collection with Open Source Virtual Machine

Delvin C. Defoe
Rose-Hulman Institute of Technology
5500 Wabash Ave
Terre Haute, IN 47803
defoe@rose-hulman.edu

ABSTRACT
Garbage collection is an integral and fundamental component of every modern memory-managed programming language platform such as Java and Microsoft .Net. Yet, few Computer Science and Information Technology programs offer students a course on garbage collection. High tech companies wish that their new employees knew something about garbage collection as this would help them with decisions concerning languages that are most appropriate for implementing software solutions on behalf of their clients. To address this limitation, we have created a junior level undergraduate garbage collection course that gives students practical experience in exploring, designing, and implementing garbage collection algorithms in the context of an open source virtual machine. We have taught this course twice and on both occasions students were pleased with their learning experience.

Categories and Subject Descriptors
D.3.4 [Programming Languages]: Processors—Memory management (garbage collection); K.3.2 [Computers and Education]: Computer and Information Science Education—Curriculum

General Terms
Design, Experimentation, Human Factors

Keywords
Garbage collection, computer science education, virtual machines, open source software

1. INTRODUCTION AND MOTIVATION
Had it not been for the advent of GC in the nineteen-sixties, we would still be chasing pointers as is done in programming languages like C and C++. A significant portion of software development time would still be invested in avoiding dangling pointers [1, 5] and chasing memory leaks [6]. The invention of GC has lead to a plethora of memory-managed programming languages that not only introduced flexibility in the Computer Science curriculum, but also in computing research and in industry. Research labs care about quick prototyping, simulation, and visualization of their results. High-tech companies care about throughput and productivity. These institutions and organizations choose programming languages that facilitate software implementation—they choose memory-managed (i.e., garbage collected) languages.

Rose-Hulman Institute of Technology offers a course on the Theory and Practice of Garbage Collection - a junior level one term course that gives students practical experience in exploring, designing, and implementing garbage collection algorithms in the context of an open source virtual machine. Students are exposed to the state-of-the-art in garbage collection technology and are given an opportunity to pursue projects (several small and one large) involving designing and implementing garbage collectors in the runtime environment of a garbage-collected language.

We were motivated to offer this course because most Computer Science departments use modern, high-level, memory-managed programming languages like Java, Python, and C# to teach their Computer Science courses. Yet, few expose their students to the field of garbage collection (GC)—the process by which the runtime system automatically reclaims the memory occupied by dynamically allocated objects that an application no longer needs. Many students hear about GC en passant; some receive shallow coverage of it; and fewer students receive enough coverage of GC to appreciate its significance to their Computer Science education. The Jikes Research Virtual Machine (RVM) Project lists current users, teaching resources, and courses that have been run using the Jikes RVM on their website [12]. Only a small minority of those courses relate to garbage collection, and most are taught as graduate level seminars in which students survey the garbage collection literature.

Recall that garbage collection plays an important role in software development for institutions and organizations that care about speed, throughput, and productivity. Simply choosing a memory-managed programming language to write software solutions for a particular problem domain is not sufficient for large software companies like Interactive Intelligence [7]. These companies often make decisions about programming languages that are most appropriate for developing software solutions for some of their products (e.g., business communications systems, and enterprise IP tele-
One of the factors that contributes toward these decisions concerns the memory management behavior of the language. Employees who receive practical training in GC during their undergraduate tenure can be empowered to contribute in the decision-making process of such companies. During a recent campus visit, Interactive Intelligence expressed disappointment that most of their new employees lacked memory management experience.

We address these limitations by offering a Theory and Practice of Garbage Collection course in which students work with a large-scale, real-world software platform and interact with the wider garbage collection community. We trust that this will serve as a model for other Computer Science departments that desire to add a GC course to their undergraduate curriculum.

2. CURRICULUM

The instructor who regularly teaches the Theory and Practice of Garbage Collection course has performed research in the area of GC. GC is the primary focus of his research and in addition to his expertise in this area, a survey was conducted of existing courses that cover GC. Table 1 gives a representative list of courses found by searching the Web, word of mouth, polling the SIGCSE and Jikes RVM communities, and following links referenced from other course web pages. This is by no means an exhaustive list of all GC courses, although these are the only ones that we found. It merely gives us an idea of the types of courses that focus primarily on GC concepts. Most of these courses are graduate level seminars where students survey the literature and present or critique technical articles. They are theoretical in nature and offer little or no practical experience with building and deploying garbage collectors in the context of a real virtual machine.

In keeping with the hands-on nature of our students’ education, it was decided that our Theory and Practice of Garbage Collection course would use a more pragmatic approach that focuses on skills acquisition. The course would expose students to a variety of GC algorithms and have them implement at least one in the context of a real-world virtual machine. The benefits to students include practical experience in working with industry-scale software systems and preparation for the job market. The course would also provide students with the opportunity to interact with the wider GC community. To these ends, the following course outcomes were developed. Students who successfully complete this course should be able to:

1. Describe the process of Dynamic Storage Reclamation or Garbage Collection.
2. Explain the two-phase abstraction of the basic function of a garbage collector.
3. Classify garbage collection techniques into a number of distinct categories (e.g., stop-the-world, incremental, generational, and concurrent).
4. Compare and contrast garbage collection techniques based on their space and time trade-offs.
5. Evaluate virtual machines for a garbage collected language, e.g., Java.
6. Design, implement, and analyze garbage collection techniques in the context of a virtual machine.
7. Read, discuss, and present technical papers in the field of Dynamic Storage Reclamation.

The assessment tools for evaluating these outcomes include homework and programming problems, exams, in-class presentations and discussions, and project development. Students complete a major term project in small groups of two or three students and do the other assignments individually. They contribute toward evaluating each other’s presentations and give feedback to fellow classmates. At the end of the term students reflect on and evaluate the course by completing an anonymous survey. A summary of their evaluation is given in Section 6.

Topics covered in the course include GC Basics, GC design choices, Reference-counting GC, Mark-sweep GC, Copying GC, Generational GC, Incremental and concurrent GC. These topics are covered during the first eight weeks of the course.

The last two weeks are reserved for presentation and discussion of technical articles and project demonstration.

3. TEXTBOOK

We searched the Web extensively and contacted a number of publishers for an appropriate text to use for the GC course. The only text we found was the Garbage Collection book by Jones and Lins [8], which is a bit outdated. We contacted the author(s) and received material that was helpful in preparing the course content. We also pointed students to a few classical papers on garbage collection (examples include articles by Paul Wilson) [3, 9, 15, 16]. Some of these articles date back to the nineteen-sixties and give a historical perspective of the field of garbage collection.

One of the activities that students engage in is the presentation of scientific papers from the field of Dynamic Storage Reclamation. To prepare students for this activity, they are presented with a list of GC articles that represent a cross-section of the evolution of the field of garbage collection. They can present articles from that list or any other article that interests them. We were amazed by the variety of articles that were presented during the first two offerings.

1 We would be delighted to share our modules with the community, upon request.
of the course; articles on GC for Haskell, incremental and concurrent collection, real-time GC, multiprocessor GC, and environments for implementing GC were presented.

4. USING AN OPEN SOURCE VM

In order to give students practical experience designing and implementing garbage collectors for a memory-managed language, a real-world open source virtual machine (VM) is needed. Several open source VMs exist including OpenJDK (also known as The Java HotSpot Virtual Machine), Jikes RVM, Kaffe, SableVM, and Joeq Virtual Machine, among others. We decided to use an open source platform because since the last decade [11] many businesses have started to use or build upon open source software. Moreover, about 90% of our students take industry positions after graduation. Since we prepare them for careers in industry, we thought it would be worthwhile to expose them to the type and scale of software they should anticipate when working in industry. Furthermore, students build confidence, experience empowerment, and feel more motivated when they pursue meaningful, real-world projects [4, 13]. The final project, described in Section 5, gives them such experience.

One of the key assignments that students must complete involves surveying available VMs and recommending a VM that would be appropriate for implementing their group’s garbage collector. They are given the following questions to guide them in the process.

1. Does the VM have a large active community of developers and users that supports it?
2. Is the VM associated with a community that uses a mailing list or other media to offer support to both users and developers?
3. Is the VM written in a language with which you are comfortable programming?
4. Is this a VM whose design is simple, modular, and easy to use for implementing memory management algorithms?
5. Does the VM contain instructions on how to add or modify garbage collectors (the availability of instructions for incorporating at least one GC would make the VM attractive)?

Both times the course was offered, Jikes RVM was selected as the VM of choice. In addressing the questions listed above, students cited its modularity, strong support community, wide usage in research and academia, documentation, and the fact that most of it is written in Java as reasons why it is a great environment for completing their project[2]. During the first offering of the course, students recommended Jikes; so, for the second offering, the instructor made it the standard VM. Jikes RVM is an open source VM that "provides a flexible open testbed to prototype virtual machine technologies and experiment with a large variety of design alternatives" [12]. Its memory management toolkit (the MMTK) is the platform on which garbage collection algorithms are implemented. Many researchers use Jikes RVM and the MMTK to pursue garbage collection research and teach courses. The MMTK provides implementation of several algorithms including:

- NoGC – no garbage collection is performed.
- SemiSpace – a copying semi-space collector.
- MarkSweep – a mark-and-sweep (non copying) collector.
- GenCopy – a classic copying generational collector with a copying higher generation.
- GenMS – a copying generational collector with a non-copying mark-and-sweep mature space.
- CopyMS – a hybrid non-generational collector with a copying space (into which all allocation goes), and a non-copying space into which survivors go.
- RefCount – a reference counting collector with synchronous (non-concurrent) cycle collection.

Students are advised to join the Jikes RVM rem-research mailing list to acquaint themselves with the larger memory management community, explore their archives, keep up with the latest developments in memory management research, and post questions/comments/concerns as they interact with the VM and the MMTK. Students are also encouraged to explore the garbage collectors listed above.

5. PROJECT AND LAB ASSIGNMENTS

The main project for the course is the implementation of a garbage collector for a memory-managed programming language like Java in the context of an open source virtual machine like Jikes RVM. To prepare for this project, students must first install and familiarize themselves with Jikes RVM and in particular, its memory management toolkit. A LINUX environment is recommended for this project. Several small assignments are given to help students gain experience with Jikes and the MMTK. During each assignment, they are advised to collaborate with each other, consult with their instructor, and interact with the larger memory management community. The latter can be done by exploiting the Jikes RVM rem-research mailing list. A few sample assignments are described below[3].

5.1 Installation of Jikes RVM

Students are pointed to installation instructions on the Jikes RVM website, which they should follow to install the RVM on a LINUX machine. While installing the VM, they should document their experiences and share them with their instructor. The following questions are used to guide their thought process and shape their report.

- Did you encounter problems?
- How did you address them?
- Were you required to install additional software?
- What were they and how did you install them?

Note: a strong Java background is required for this course. Thus, selection of a VM written in Java was important to our students.

3The instructor would be happy to make all the assignments available to instructors who are interested in teaching a course on garbage collection, upon request.
• What is the status of your installation?

• Can you give an example of how you use Jikes RVM to run a Java application on your machine.

After completing the installation, students submit their report to their instructor. They are also required to demonstrate to their instructor that they have successfully installed Jikes RVM on their LINUX machine and are able to use it to run a Java program.

5.2 Exploration of Jikes RVM

The next assignment challenges students to explore and become familiar with Jikes RVM. They are to describe its object layout, being careful to list some of the fields including meta-data fields that each instance of an object must contain and to describe the function or purpose of at least two such meta-data fields. The MMTK guide, javadoc, and VM code all describe the object layout in detail.

Students must delve deeper into the MMTK to answer the following questions.

• What is the default garbage collector implemented in the MMTK? List the other garbage collectors provided in the MMTK.

• Describe the framework in which the aforementioned garbage collectors are implemented.

• Discuss the challenges involved in selecting a different garbage collector for consecutive executions of a Java program.

• Briefly describe the process of adding a new garbage collector to the MMTK.

Jikes RVM is a large-scale software framework. Students can be petrified if this is their first experience working with a product of that scale. The above questions help guide their exploration and minimize their concerns. This is by no means an exhaustive list of all the questions that can be utilized to guide students’ exploration. It merely gives an idea of the types of questions that can be considered to serve that purpose. Another instructor could conceivably generate a different, yet appropriate list of questions that serves the same purpose.

5.3 Major term project

The major term project involves designing and implementing a garbage collector in the context of Jikes RVM. However, before students engage in adding a garbage collector to the MMTK, they must implement the MMTK Tutorial Mark-Sweep Collector. The collector begins as a GC with no collection policy and is modified such that the allocation policy is changed from bump-allocation to free-list allocation, yielding a collector with yet no collection policy. The mark-sweep collection policy is then implemented, yielding a complete mark-sweep collector. Detailed instructions for adding the tutorial collector to the MMTK are provided [2]. The resulting collector can be modified further to yield a hybrid copying/mark-sweep collector or a generational mark-sweep collector. Successful implementation of this collector gives students the hands-on experience that they need to enhance their learning and prepares them to add a collector of their choosing to the MMTK.

There are several approaches that can be employed in deciding on a collector for students to implement. Students can be asked to modify an existing collector; they can be prescribed a collector to implement; the class can be made to vote on a collector that every team of two to three students implements; or student teams can be given the option, with some guidance of course, to select a collector of their choice. We elected to embrace the latter approach because it provides flexibility and empowers students to choose to design and implement a garbage collector that is unique, one they are motivated to develop and can have fun with implementing.

Recall that one of the learning outcomes for the course is that students read, discuss, and present technical papers in the field of Dynamic Storage Reclamation. During the second half of the term students are engaged in this activity, which gives them ample opportunity to select from among collectors that were not presented in class. We noticed during the first two offerings of the course that some teams were motivated to select from among those collectors and did an exceptional job on their project.

An important question that can be posed is the following. “How do we keep students engaged when they collaborate on a project of that magnitude for a period of about four weeks?” We challenge them to set their own milestones, commit their documentation and code to a shared repository that the instructor can access, and meet with their instructor at least once a week to report on what they have done so far and their plans until the next meeting. The instructor evaluates their progress, gives feedback on their progress and guides their plans as they flesh-out the details. Students have typically enjoyed these sessions. At the end of the four week period, students present their work to the class and give a short demo of their collector. Both instructor and non-presenting students evaluate and give feedback on each presentation. This challenges students to get involved in every aspect of the course. This is an upper level course that prepares students for the real world. Engaging students in peer-evaluation is a valuable skill that will serve them well in their future careers.

6. DISCUSSION

The curricula covered during the 2008 and 2010 offerings of the course were the same as described in Section 2 with one exception: for the first offering, students recommended and were allowed to select a VM of their choice, but for the second offering, Jikes RVM was chosen. In both cases, Jikes RVM was the only VM used; also, the term project was the same. The exams administered in both offerings covered the same topics and were both void of programming assignments. Similarly to McCown’s experience with his web information retrieval course [10], we observed that the overall exam averages were quite similar for the two offerings of the course (81 % in 2008 and 79 % in 2010). The project averages were also similar (82 % in 2008 and 86 % in 2010). The class with the slightly better exam average got the slightly worse project average and the class with the slightly worse exam average got the slightly better project average. Although we cannot draw any conclusions from this observation due to small class sizes (4 students in 2008 and 8 students in 2010), this may suggest that factors other than the term project contributed toward the overall learning of garbage collection concepts. According to course evaluations.
administered at the end of each term, students in the 2008 class rated the quality of their learning very high, 4.5 on a five-point Likert scale. Students in 2010 did not rate their learning as high (3.57 on the same scale).

The end-of-term course evaluations give students the opportunity to compare the workload for this course with the workload for other courses of equal credit. In both cases, students indicated that the workload was about the same as the workload for their other courses. Each term the course was offered, the instructor solicited informal feedback from students on the delivery of the course and the assignments. In both terms students were pleased with the lectures and the exams, but felt a bit overwhelmed with Jikes RVM. In 2008, Operating Systems was not a prerequisite for this course, so the instructor understood students frustration with LINUX and Jikes RVM. In 2010, Operating Systems was made a prerequisite to this course. Consequently, students struggled less with LINUX. Working with Jikes RVM was still a challenge, but a majority of the teams was able to demonstrate a working version of the garbage collector that they implemented.

Many students expressed being overwhelmed by the complexity of Jikes RVM. Configuring, installing, and building the VM incurred a large learning curve on the part of both the students and instructor. It was challenging to follow some of the installation instructions because several versions of the VM are available and different instructions refer to different versions as the most stable version. For example, the user guide gave instructions for a particular version and the quick start guide gave instruction for the repository head. These two are not always the same. The documentation for the RVM project is substantial. However, it is scattered on several sites, e.g., [12, 14], and some guides are out-of-date. The most complete MMTK guide, for example, dates back to September, 2006 [2]. This was written for Jikes RVM version 2.3.7. The most current version when the class was last taught was 3.1.1. After installing the VM, students felt somewhat intimidated by the size and complexity of the code base of the MMTK. Although the code base is modular and is written in Java, students had limited experience modifying a code base so large. The instructor walked them through the process of exploring the code of a few example collectors so they could become familiar with the general architecture. The instructor also recommended that they consult the RVM mailing list to pose questions or concerns that they had. A few groups took advantage of that resource and were able to make meaningful progress on their project. Others surveyed the archives, as suggested, to find answers to some of their concerns.

Overall, students felt that the project was successful in giving them practical experience in exploring, designing, and implementing garbage collection algorithms in the context of an open source virtual machine. Although they struggled a bit with the complexity of Jikes RVM, they spoke highly of their learning experience. Here are a few quotes that summarize their sentiment.

"The lectures for this course were just great! I learned so much of theoretical aspects during the class and also code aspects on the project."

"It is a very interesting topic. I would definitely suggest that this course be offered again."

"Nice to have some hands-on experience with implementing GCs. Good overview course, not too demanding"

Several lessons were learned from the experience of teaching this course on two occasions.

1. Consider exploring the implementation of extant collectors in Jikes RVM as the course progresses. That way, students can see how the code pieces of each collector connect together, and how each GC plan works.

2. From a learning perspective, students would prefer that we begin working on the garbage collector early so it is not too demanding later on in the course.

3. Teaching with open source software systems is challenging to both students and instructor. A significant learning curve is associated with installing, configuring, and using a large-scale open source system. Creating appropriate assignments to facilitate student learning is also challenging. Systems that are widely used, well supported, and associated with a great support community should be considered.

4. Involve students at every level of course development and delivery. They feel empowered and appreciate the effort that instructors invest in delivering a course to them. It also prepares them to be leaders in their career endeavors.

The project experience could have been more rewarding if students could have reformatted the MMTK to suit their programming styles and plug in their collector in their custom designed toolkit. However, this was beyond the scope of this course. Moreover, a ten week term does not afford us enough time to do justice to a project of that scale. Another option is to consider other languages, e.g., Python or Haskell, and explore their virtual machines. We are not certain that our experiences would be different.

7. CONCLUSIONS AND FUTURE WORK

Garbage collection is an important topic that should be included in the Computer Science curriculum. Students typically hear about garbage collection in passing, with little detailed coverage. Only a handful of Computer Science programs include a garbage collection course as part of their curriculum. Among the few departments that offer such a course, the typical format is a graduate level seminar that surveys the garbage collection literature. Little or no practical experience in designing and deploying garbage collectors in the context of a real virtual machine is offered.

We have described our approach to introducing a practical course on garbage collection in our curriculum. The course includes projects that give students practical experience in adding garbage collection algorithms to a real-world open source virtual machine environment. We have shared some of the highlights and challenges to this approach. Our hope is that other departments can learn from our experience.

We are planning to offer our garbage collection course again in spring 2012. We will have students install the virtual machine during the first week of classes and sprinkle a number of smaller projects throughout the term to give students more experience using the virtual machine. We will also devote a few lectures to “getting to know your VM”. We anticipate that the course will continue to grow in popularity as guest lectures on garbage collection are given in other courses and opportunities in the field of memory management increase.
8. REFERENCES


