Supporting CS Education via Virtualization and Packages

Tools for Successfully Accommodating "Bring-Your-Own-Device" at Scale

Andy Sayler, Dirk Grunwald, John Black, Elizabeth White, and Matthew Monaco
University of Colorado
Boulder, CO, USA
<first>.<last>@colorado.edu

ABSTRACT

Higher education is facing a paradigm shift in the ownership and use of computer hardware. The school computer lab is no longer the primary place of student computer use. Instead, students increasingly expect to use their own hardware to complete their school assignments. This creates a challenge for computer science educators: we must now support a wide range of heterogeneous hardware without the benefits of tight control over its use. To address this "Bring-Your-Own-Device" (BYOD) challenge, we leverage virtualization and software packaging systems to gracefully deploy and support a standardized development environment for all core CS courses across a range of both school-owned and student-owned computing devices. We have deployed and evaluated our system for the previous two years at scale and continue to actively use and develop it. It has effectively helped us support multiple classes comprising hundreds of students with very limited IT staffing. We describe the design and management of our system, present our experience using our system, and discuss the lessons we've learned. We also provide data reflecting current student user experience with our system. Our system has proven very effective in addressing the student BYOD challenge in a manageable, cost-efficient, and easy-to-use manner.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms

Management, Economics, Human Factors

Keywords

BYOD, Virtual Machines, Education, Development Environment, Package Management, Tools, Best Practices

1. INTRODUCTION

BYOD (Bring Your Own Device) has quickly become the buzzword du jour in many IT circles. The term refers to the increasing prevalence of employee-owned hardware in traditionally corporate-owned IT settings. Higher education is not immune to this trend. Increasingly, students show up to college with their own laptops and expect to complete all of their required assignments on these machines. We have observed many recent students graduating without ever having entered a school-owned computer lab or worked on a school-owned computer. This creates a challenge for educators and the IT departments that support them: how do we ensure that students can successfully complete assignments on their own machines? How can we guarantee a good experience doing so? And how can we achieve both of these objectives while adhering to budgetary limits, without prohibitive increases in support staff hours and overhead?

In computer science, the issue is exacerbated by the predominance of unfamiliar (at least to incoming students) applications and operating systems. Like many universities, our institution primarily uses a UNIX-derived development environment (GNU/Linux). While a Linux development environment has many advantages (discussed below), the fact that most students arrive without Linux installed on their computers poses a significant challenge. It is a non-trivial task to ensure students can run Linux, complete with a preconfigured suite of versioned compilers, debuggers, and editors to ensure uniform results across student, instructor, and grader machines, on their own hardware.

The problem could likely be solved with sufficient human resources: if we paired each student with an experienced CS practitioner to help them setup a dual-boot, properly-configured development environment on their machine, we could likely deploy and support a uniform environment across a range of hardware; this is in fact the method traditionally used in many educational computing environments. But in the age of shrinking budgets, growing teaching loads, and a push toward MOOC courses that can scale to thousands and tens of thousands of students, such a brute force approach is simply not feasible in most situations.

Instead, we have leveraged existing off-the-shelf virtualization and software management systems to create a common virtual machine that is used across all of our core computer science courses. This system allows us to provide students with an easily deployable solution that allows them to complete all of their CS work on their own machines without forcing them to significantly modify their existing setups. Our approach has allowed us to support over eight-hundred undergraduate students, many with little to no prior CS experience, for the cost of a single part-time graduate student appointment. The primary contributions of this paper are twofold: We demonstrate how VMs can be deployed *en masse* with minimal staffing to provide a single standardized department-wide development environment across both VM and lab machines. We also show how using centralized software management tools can simplify and streamline the distribution and maintenance of such a common development environment.

2. USING VIRTUAL MACHINES

Leveraging virtualization technology to provide a uniform development environment in an educational setting is not a new idea. At our institution, for example, individual faculty have used virtual machines (VMs) in classes for ≈ 7 years. For example, VMs have long proven useful in computer security courses, both for use as servers that students can practice attacking, as well as for use as clients that come pre-installed with the necessary security tools [15]. We have also used VMs since 2006 in a computer systems course to enable new pedagogic techniques. VMs have also proven useful in topics such as robotics [7] and SaaS [8]. These ad-hoc, per-course VM deployments have a several downsides: they require students to install a separate (and often large) VM image for each course, they require a separate VM support stack for each course, and they require students to learn simple tasks on multiple unfamiliar systems.

2.1 Our Requirements

To improve upon this situation, we have pursued the development a standard department-wide development environment. Our goal was simple: provide students with an easy to install and use development environment that they could use across all their CS courses. The development environment should be available both on department lab machines, and as a VM for use on student-owned machines (e.g. as a "lab in a box"). Underlying this goal were a number of expectations and requirements from various stakeholders in the education environment.

From the student perspective, our solution had to: a) Run on a range of host systems; b) Be easy to install; c) Be easy to use and maintain; d) Minimize side-effects on the host system; e) Provide a stable experience throughout the semester.

From the instructor perspective, our solution had to: a) Keep the students happy; b) Minimize instructor IT overhead; c) Provide consistent results across student, grader, and instructor machines; d) Provide all necessary software for the course; e) Provide the ability to update software as the course progresses.

From the IT and management perspective, our solution had to: a) Keep the students happy; b) Keep the instructors happy; c) Be easy to support and maintain; d) Minimize resource (personal, cost, etc) requirements.

2.2 Selecting Hypervisors and VMs

The first component of any virtualization system is the hypervisor: the software that allows the host machine to run virtual machines. There are a variety of hypervisors available addressing a range of applications from server to desktop environments. We chose to use Oracle's VirtualBox [10] hypervisor for several reasons.

First, VirtualBox is a true multi-platform hypervisor providing a single unified platform for Windows, OSX, and Linux. Unlike other products such as HyperV, Parallels, or VMWare, VirtualBox allows us to support a single piece of software across all common student platforms, reducing the burden of supporting a range of student hardware and operating systems. Second, VirtualBox is both Free and Open Source; students do not have to pay to use it, and they are able to analyze or modify the source code if they wish. This avoids requiring students or departments to purchase extra software and does not require students to install software on their personal machines that they have no means to vet or modify. Finally, VirtualBox has solid support for Linux guests. This made it easy for us to configure our guest OS to interact with VirtualBox in a manner that ensures an optimal experience, allowing us to support functions like copying between the VM and host system, sharing files between systems, automatically scaling display resolutions, etc.

Atop the hypervisor, we deploy a virtual machine based on the Ubuntu 12.04.3 Desktop OS [6] (the latest Long Term Support Ubuntu Linux release). We opted for a GNU/Linux-based development environment for its robust support for a variety of CS tools and software, its widespread industry appeal, and its free, open, and permissive licensing. We opted to use Ubuntu due to its large support base in the Linux community. Ubuntu also has a rich library of software packages, meaning that required software (i.e. ruby, scala, python, gcc, etc) would be available with little additional work. Ubuntu also provides a five-year supported product life-cycle and is designed to be fairly user-friendly for new Linux users.

Our VM image starts as a standard Ubuntu 12.04.3 install. We then configure it to work optimally on the average student system. We use disk images that provide up to 30GB of VM HD space using a disk format that expands on demand to minimize actual disk use on student host systems. We configure the system to use up to 2GB of RAM and up to 2 CPU cores by default. We make a few minor modifications to the system configuration directly (i.e. setting the desktop background to an image that includes information for using and getting help with the VM), but for the most part, the majority of the post-install VM config is handled automatically using a series of software meta-packages we have created (described below). We distribute our VM as a single Open Virtualization Format (`.ova`) image file which allows students to import/install it with a single click (assuming VirtualBox itself has already been installed).

2.3 Managing with Packages

One of the key challenges behind using a single VM across the entire department is how to manage the specific requirements of each individual class using the VM. This challenge is not unique to VMs: we also face the same issue when managing departmental computer labs. A multitude of instructors across a variety of courses all have specific requirements for what must be installed in a lab and how it must be configured. We decided to solve the general lab management issue and then apply the same solution to both the VM and our lab machines, leading to our ability to maintain a unified environment across both VM and lab images.

Fortunately, there already exist systems designed to make the management of software across a multitude of machines easy: package management utilities. In particular, given our use of Ubuntu, we looked to the Debian package management tools for building, serving, and managing `.deb` packages [2]. Debian packages traditionally contain several components: the executable code and auxiliary files for a specific program, a set of configuration scripts that guide the install/uninstall processes, and a list of other packages that must be installed as dependencies. We leverage the latter two components of a `.deb` package to control our lab software environment. To install all of the required software and configuration for a given course, we simply need to create a package that lists the required course-software as dependencies and contains any course-specific configuration in its install script. This type of dependency/script-only package is normally called a *meta-package* to differentiate it from a full-fledged program package. Installing such a course meta package via the normal tools (e.g. `apt-get`) will trigger the installation of all required software and configuration for the given course.

We set about creating meta-packages for each course in the department that required use of our lab machines or our common VM. Creating these packages in the standard Debian format is generally quite simple, requiring only a few basic text files that provide, among other things, a list of required software for a given course [5]. In cases where a course requires more in-depth configuration beyond simply ensuring all the required software is installed, we can also include configuration scripts to execute arbitrary operations within our development environment. Thus we can create a very powerful, per-course package that is capable of automatically deploying all required configuration and software for a given course. Our current course packages (and the scripts we use to build and maintain them) are open source and available on GitHub [13].

After creating packages for each course, we had to make our packages publicly available via a web-based package repository. Again, the standard Debian tools make it easy to deploy and manage such a repository in a manner no more complicated than creating and deploying a simple website [9]. Our `apt` repository is publicly available at [14]. We simply add our repository's web address to the repository list on the base VM and lab machines, install our package-signing key, and these machines have direct access to our packages alongside the standard Ubuntu packages. Configuring the lab or VM then becomes a simple matter of installing the package for each course. To simplify matters further, we actually created an additional meta package for both the lab and VM that simply lists all of the active courses in a given semester as dependencies. Thus, we must only install this single master package on the VM or lab machines to trigger the installation and configuration of the entire common development environment.

Packages have a number of additional benefits beyond a highly streamlined and simplified install process. For instance, they provide us with the ability to update lab and VM configurations during the semester without needing to redeploy the VM or lab images. If an instructor decides to use a different software library half way through the semester, we can simply update the package and push the update to the package server. All of the student VMs and lab machines will automatically pull down the new change and apply it seamlessly without requiring messy re-installs or manual intervention[1]. In addition, packages provide a simple centralized means to document all of the required software and settings for a given course. No longer do we have to tediously remember what configuration changes were applied in the lab or what software has been installed. It's all documented within the package configuration files themselves and can be easily managed and updated from semester to semester.

Finally, controlling all per-course configuration via packages is an efficient way to use a single VM that can be easily extended to support all courses. Students can download a "lean" version of the VM that has only the core course packages installed, and then they can install extra packages for any course they are currently enrolled beyond those included on the lean VM. This makes it easy for students to manage their own VMs from semester to semester and to grab all of the software for their current courses without having to waste hard drive space on courses they aren't taking. Students who have native Linux installs or otherwise prefer to set up their own machines without the use of the VM can also use our course packages to ensure their self-configured setups contain all of the same software as the VM or lab machines. VMs may provide the body of a lab machine on each student's personal computer, but per-course packages provide the brains that ensure everything stays manageable, uniform, and up-to-date across VM, lab, and self-installed environments.

3. SUPPORT INFRASTRUCTURE

In addition to providing the VM itself and packages for each course that uses it (or any other lab machines), we also provide the necessary support infrastructure for these systems. The goal of this support infrastructure is to shift the bulk of the burden of providing per-course IT support to a single individual or group in charge of the VM, course packages, and related lab environments. Currently, we employee a single grad student in a 20 hour/week appointment in charge of VM and course package support for ≈ 800 students spread across ≈ 5 core courses. We have found that when properly coordinated, such a position can easily support the scale we are operating at and beyond. We have found it far more efficient, both in terms of time and cost, to employ a single dedicated development environment support person over requiring each instructor to dedicate portions of their time to the support of their course IT. This arrangement allows instructors to focus on instruction while a small number of highly-trained environment support staff focus on environment support.

3.1 Supporting Students

Effectively supporting students proves to be particularly challenging: we need to have ≈ 800 students successfully using the virtual machine for their courses within about one week. To accomplish this, we provide a variety of support resources for students. First and foremost, we provide a central web-page [11] that provides information about the VM, download links, installation instructions, common troubleshooting steps, and related self-help information. We also

[1]In time sensitive situations, relying on the automatic update system is often undesirable due to its opportunistic nature. But even in this case, forcefully updating a VM or lab machine is a simple matter of manually running the system update program.

provide YouTube videos describing the installation and usage of the VM. We have found that about 80% of our user base [12] is capable of setting up the VM on their own using these resources.

For the remaining students who require or desire in-person help setting up the VM, we provide a series of "install sessions" at the beginning of each semester. We advertise these sessions to all courses using the VM and open them up to anyone who wishes to attend. During the sessions, we provide copies of the VM on USB flash drives (to avoid having to wait for students to download the image) and walk students through the install process step by step. We have one or more VM experts on hand during these sessions to assist with unforeseen issues and help students work through individual problems. Many students who attend these sessions do so after attempting to install the VM on their own and encountering some issue. They then attend the session as a fallback to installing the VM themselves.

After the initial install sessions, we provide two to four staffed hours of weekly VM support "office hours" through the remainder of the semester in a central department location open to all students. These weekly hours provide opportunities to assist students who encounter issues after the initial VM install or who require extra help getting up to speed with the usage of the development environments (Linux, editors, etc). These hours allow courses using the VM to focus their own office hours and contact time on course content and forward students with VM or environment issues to the dedicated VM office hours each week.

3.2 Supporting Instructors

Our support resources for instructors are focused primarily on communication, feedback, and training. Prior to the start of each semester, we meet with each instructor and gather a list of the required software and configuration for their courses. These are then used to build the per-course packages mentioned previously. We follow this up with a beta-release of the VM image that instructors and their TAs use to test their assignments in advance of the start of classes. Feedback from this beta release is then incorporated into the course packages (and thus the VM) prior to the student-facing VM release.

We also provide basic VM training to the instructors and their TAs in several group sessions at the start of each semester. This training focuses on the basics of what the VM does and how it works. The goal is not to make the TAs and instructors experts at supporting the VM directly, but to ensure that they are knowledgeable about its use during class demos and that they are capable of recognizing abnormal behavior when it occurs so that they can forward issues to the dedicated VM support staff. We continue meeting with instructors and TAs throughout the semester to elicit feedback on the VM, coordinate any mid-semester updates to the course packages, and monitor their ongoing experience with the VM. This culminates in a end-of-semester survey that we send to all VM users in an effort to gather feedback on the VM and to improve the system in subsequent semesters.

4. EVALUATION

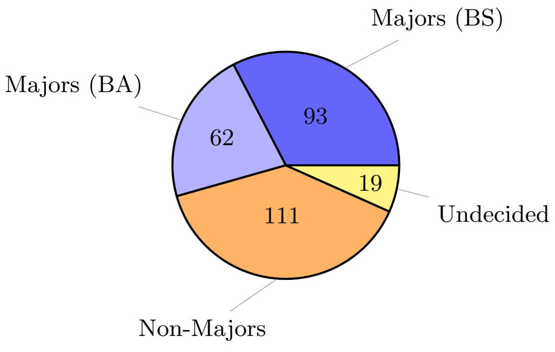
We are currently in the second full year of using the common CS VM development environment. We have deployed the system for the previous three semesters, and are currently using it this semester with the largest user base yet. Over this time, we have had a thorough opportunity to evaluate the VM in both qualitative and quantitative measure across a range of users.

Each semester we deploy the VM, we have conduced an end-of-semester user survey. The survey is sent to all students, instructors, and TAs known to be using the VM as their primary development environment, which includes all of the core CS courses. The survey is web-based, anonymous, and optional. It takes about 5 minutes to complete. Figures 1 and 2 present data from our most recent survey: Fall 2013. The Fall 2013 survey had $\approx$ 300 responses from VM users, representing about 40% of the $\approx$ 800 student VM user base across our core courses. The full results of the survey, including user comments, are available in [12].

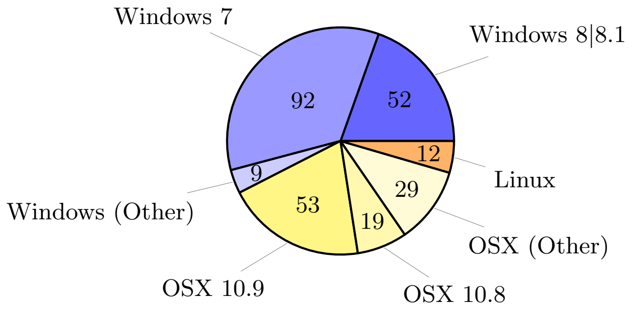
Figure 1a shows the basic demographic breakdown of the VM user base. Our current user base is primarily split between Engineering (BS) CS major, Arts and Sciences (BA) CS major, and non-major undergraduate students. Figure 1b shows the host Operating Systems that our VM users report using. This indicates the range of hardware that can be supported with a single common VM platform. At this time, we do not officially recommend a specific type of machine to incoming students, leaving them free to select any machine they like. The majority of our users run the VM on their personal laptops. A small portion use desktops, often at home or in their dorm rooms. About 57% of our VM user base uses Windows, primarily Windows 7 or Windows 8. About 38% of our users use OSX (Macs). The remaining 5% use Linux. Over three quarters of our users have machines with modern Intel Core i-series CPUs. With the benefit of multiple cores and hardware virtualization extensions, these processors are more than adequate to run the VM. About a quarter of our users tend to use slightly older CPUs, which may or may not run the VM optimally depending on their design.

Figure 2a shows our users' impressions related to using the VM. Two thirds of our users rated their experience using the VM as "Good" (4 out of 5) or "Excellent" (5 out of 5). Only 6% rates their experience as "Poor" (2 out of 5) or "Terrible" (1 out of 5). Thus, the vast majority of our users seem satisfied with their VM experience. Likewise, almost 80% of our users report using the VM for "All" or "Most" of their coursework, indicating that the system is in fact usable and preferable to the alternatives (working in the lab, installing Linux directly on their devises, etc). VM performance tends to be the most common complaint related to VM usage. Virtualization does impose an overhead on performance, especially in the absence of enabled processor virtualization extensions.

While Linux is a new experience for over half of our users, very few of them seem to struggle with the new OS. We provide both live and videotaped supplemental lectures covering the basics of the Linux OS and standard development tools for those that are interested in extra assistance or a more in-depth introduction, but only a fraction of the user base pursues this supplemental material. Modern Linux Desktop OSes like Ubuntu provide a very familiar experience for OSX and Windows users. Furthermore, in our introductory courses our users need only a small fraction of what Linux has to offer, reducing the scope of new OS functionality they must learn. We have observed more than one of our users

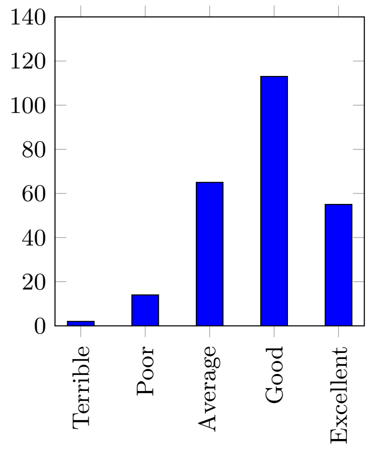


(a) User Major

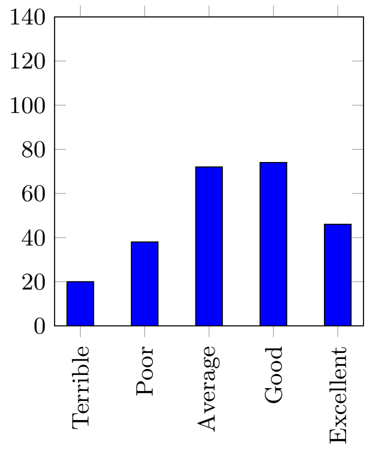


(b) Host OS

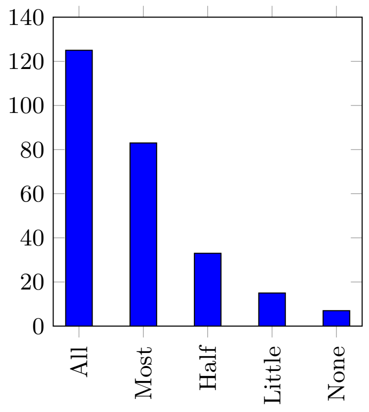
Figure 1: VM Users



(a) Overall Experience



(b) Performance/Speed



(c) Coursework on VM

Figure 2: VM User Experience

struggle more with the switch from Windows 7 to Windows 8 than from Windows to Linux.

5. SOCIAL IMPACT

Before students owned their own machines, they were forced to sit in university-equipped labs, sometimes even queuing up to wait for an available machine. This often had a useful social benefit: students, in physical proximity, would form social bonds that enhanced their educational experience and transformed the lab into a social nexus. Our VM approach allows a student to "take the lab anywhere" and thereby removes the potentially-positive effect of required lab attendance.

While the lack of forced lab attendance is a potential concern, we have found the use of virtual-machines has actually promoted more effective pair and group-work by students. Students still work in the labs, but they use their laptops instead of the lab machines. In addition, they work together at many other locations on and off campus. The VM provides the flexibility to meet and work as a group anyware, without imposing the arbitrary limit of requiring access to a univerity compuer lab.

On a related note, the dedication and time-commitment required by a modern CS curriculum often has meant lab-bound students enter and leave the facility at all hours: it is not unusual to see students in our lab throughout the night. When informally surveyed, nine students (eight of them female) indicated discomfort at the idea of entering or leaving the lab after-hours due to safety concerns. Our approach alleviates this concern, and related concerns for others who are unable or uncomfortable gaining access to a university computer lab.

6. LESSONS LEARNED

Our experience with the VM has led us to develop a set of best practices, as well as to identify the continued sticking points that we hope to resolve in the future. In addition to the practices already discussed (per-course software packages, one-click installs, dedicated support staff, etc), we highlight a few additional best practices below. We recommend these guidelines, as well as the guidelines discussed in [12], to anyone looking to build a similar system.

Virtualization Extensions: Modern CPUs provide extensions that enable a fast, smooth and enjoyable VM experience (i.e. VT-x). Unfortunately, many non-Apple PC manufacturers ship their machines with these extension *disabled* in the BIOS. Getting students to enable these extensions can be a challenge, but makes a big difference in their overall impression of VM usability. One way to force students to enable these extensions is to use a 64-bit and/or multi-core VM, which VirtualBox will not start without virtualization extensions enabled. Such a VM will force students to either enable the extension themselves, or seek help at an install session, drastically improving their subsequent VM usage experience. Our experience has been that this approach requires an increase in start-of-semester help sessions, but the resulting boost in performance leads to higher overall satisfaction.

Multiple Distribution Methods: Each term starts with a rush of students trying to obtain the multi-GB VM image. Having a single direct download link is a sure way to overload your servers, overwhelm your network, and otherwise disrupt the initial VM roll out. We avoid these issues by distributing our VM images primarily via the BitTorrent peer-to-peer network [1]. We provide direct downloads to students who cannot use BitTorrent and also provide USB thumb drives with copies of the VM at all install sessions. This multi-pronged, distributed approach avoids a single bottleneck during the critical start-of-semester roll out.

Off-VM Backup System: Students are often unaware of the fact that virtual machines share the foibles of physical computers, including potential file system corruption if the VM is improperly "powered off". Thus, it's important to provide students with an easy-to-use off-VM backup mechanism. We settled on distributing the VM with the Dropbox [3] client pre-installed and encouraging students to use the free service to store their files on the VM, ensuring they are automatically synced to the cloud. We have also used GitHub [4] in a similar (although less transparent) manner. Using Dropbox or GitHub has the added benefit of providing an easy means for transferring files between the VM, the host machine, and/or other machines. Using such a system simplifies resolving VM issues by allowing our support staff to simply re-install a fresh copy of the VM when non-trivial issues arise. The student can then restore their work to their new VM install from Dropbox or GitHub.

Despite its continued success, there remain parts of our system we hope to improve. The main open issues are discussed below.

BIOS Access and Configuration: As mentioned above, getting virtualization to work smoothly often requires accessing the system BIOS and enabling virtualization extensions. This can be a challenge since the method for doing this differs by machine, and since many students have never had to access their system EFI/BIOS menus before.

Download Corruption: The majority of the VM install issues we see stem from corrupt VM image downloads. Even today, downloading multi-GB files over wifi networks tends to cause corruption in a notable number of cases. We publish md5 checksums for each VM image and encourage students to verify them before installing the VM.

No|Slow|ARM Computer: Some users can not use the VM do to their lack of machine or ownership of a low-spec machine (e.g. Intel Atom). In addition, the rise of ARM-based tablets and laptops and other non-traditional hybrid machines (ChromeBooks, iPads, etc) mean that some students are showing up with hardware incapable of supporting x86 virtualization at all. Such users must often fall back to using the actual computer lab. While our package system ensures that the lab is configured in the same manner as the VM, this still tends to put them at a disadvantage by limiting the spaces in which they may work. We are beginning to experiment with laptop loaner programs and virtual desktops to help mitigate these issues.

7. CONCLUSION

We believe our VM system has been very successful at allowing students to utilize their own hardware while also providing a common development environment that greatly simplifies instruction. We plan to continue expanding our system and improving upon the outstanding issues. We currently use the system across all core CS courses. We plan to expand the program to all CS courses, including electives and graduate classes, in the near future. We also plan to continue gathering feedback and data from users to drive the continued improvement of the system. Our system allows us to support a heterogeneous BYOD environment while still delivering an enjoyable learning experience to a wide range of students at minimal cost.

8. REFERENCES

[1] Bittorrent. `http://www.bittorrent.org/`.

[2] Debian policy manual. `http://www.debian.org/doc/debian-policy/`.

[3] Dropbox. `https://www.dropbox.com/`.

[4] Github. `https://www.github.com/`.

[5] M. Bialasinski. equivs. `http://packages.debian.org/wheezy/equivs`.

[6] Canonical. Ubuntu. `https://www.ubuntu.com/`.

[7] N. Correll, R. Wing, and D. Coleman. A one year introductory robotics curriculum for computer science upperclassmen. *IEEE Transactions on Education*, 56(1):54 – 60, 2012.

[8] A. Fox and D. Patterson. Engineering software as a service - bookware: Vm instructions. `http://beta.saasbook.info/bookware-vm-instructions`, 2013.

[9] B. R. Link. reprepro. `http://mirrorer.alioth.debian.org/`.

[10] Oracle. Virtualbox. `https://www.virtualbox.org/`.

[11] A. Sayler. Cu cs standard development environment. `http://foundation.cs.colorado.edu/sde/`.

[12] A. Sayler and D. Grunwald. Cu cs computign survey results - fall 2013. Technical report, Univerity of Colorado, Boulder, Decemeber 2013.

[13] A. Sayler and M. Monaco. Cu cs apt packages. `https://github.com/asayler/cu-cs-apt-packages`.

[14] A. Sayler and M. Monaco. Cu cs apt repository. `http://apt.cs.colorado.edu/`.

[15] G. Vigna. Teaching network security through live exercises. *Proceedings of the Third Annual World Conference on Information Security Education (WISE)*, June 2003.