Sustainable Automated Software Deployment Practices

Dan R. Herrick
Colorado State University
Engineering Network Services
Fort Collins, CO 80523-1301
970-491-3131
Dan.Herrick@Colostate.edu

John B. Tyndall
The Pennsylvania State University
Information Technology Services
University Park, PA 16802
814-865-2886
jbt8@psu.edu

ABSTRACT
Many organizations follow the same error-prone, time-consuming, and redundant procedures to install software manually, whether as part of a master image or on individual computers. Usually this involves visiting a system, executing some sort of interface, selecting a subset of modules or configuring certain options, and waiting for the installer to complete. There is another way: automated software deployment, which affords greater efficiency, consistency, and ultimately, service.

This paper discusses the organization and detailed implementation of automating software installations and updates using silent and unattended methods, with various levels of administrative intervention, from help desk to systems administrator. We also describe different approaches to creating such an environment for both "mass" devices (e.g., public computer lab systems) and individual devices (e.g., faculty/staff desktop systems).

Key concepts include leveraging management software and resources you may already have (i.e., the "zero budget" approach) versus efficiency gains from third-party resources, high-level administrative toolkits along with low-level control methods, and developing a workflow for automated and semi-automated software installations.

Categories and Subject Descriptors
C.5.3 [**Computer System Implementation**]: Microcomputers—*personal computers, portable devices, workstations*; D.0 [**Software**]: General

General Terms
Design, Documentation, Performance, Standardization, Theory.

Keywords
Baseline, best practice, EASI, EASI Make, installation, process, software, software deployment, software distribution, software installation, software packing, thin imaging, UPDATER

1. INTRODUCTION
This paper discusses automated software deployment scenarios that have been tested and used in production environments at both the College of Engineering at Colorado State University (CSU) and Penn State University (PSU).

Collectively, both universities provide regular, managed installation support for 300-400 automated software packages.

1.1 Colorado State University
Within the College of Engineering at CSU, an IT group of 8 FTEs and 25 part-time student employees supports about 4,000 users (545 of which are faculty/staff with at least one university-supported computer) and about 1,650 computers (350 of which are centrally managed computers, e.g., computer lab PCs, servers, virtual desktops connected via thin clients).

1.2 Penn State University
University Services, a sub-department in Information Technology Services, manages approximately 7,500 lab and classroom computers state-wide and offers a systems management service for an additional cooperative administration of 21,000 faculty/staff computers throughout the University. Instead of using traditional *thick* images with pre-installed software, computers are provisioned with a *thin* image with software deployed in the background. Originally, Group Policy software installations [3] were used; however, since 2008 PSU has been employing IBM Endpoint Manager [2].

2. PROCESS OVERVIEW
The process of automated software deployment breaks down into three phases: software packaging, software distribution and integration, and systems management. This paper will primarily focus on software distribution and integration, although an understanding of key elements from the other two areas is important to this process.

Software must be distributed, either using automated or manual methods (or a hybrid of both), after it has been packaged and before it can be managed. Software distribution is the actual process of installing the prepared software package onto client computers. Once the software is installed, it may need some integration to accommodate the specific computer, the needs of the user(s), and/or the network environment.

3. SOFTWARE PACKAGING
When automating installation of software, the installers that we receive from the Original Equipment Manufacturer (OEM) generally do not work for our needs; that is, they require user intervention: to select a destination folder, to select install-time options, and to select the appropriate components of a program. We find situations where we need to modify the OEM installation packages to allow for silent and unattended installation.

SIGUCCS'13, November 3–8, 2013, Chicago, Illinois, USA.

ACM 978-1-4503-2318-5/13/11…$15.00.
http://dx.doi.org/10.1145/2504776.2504802

3.1 Silent and Unattended Installations

A *silent* installation is one that does not display to the user any windows or messages while it runs. An *unattended* installation is one that requires no user interaction and does not always imply a silent installation (e.g., a progress bar may show the status of an installation as it automatically completes each step); however, a silent installation typically is also unattended.

This non-interactive nature lends itself easily to deploying software to large numbers of computers, in particular without interrupting users' sessions. A software installer may have a command line parameter that enables it to run silently; several common applications are built with these capabilities [5]; however, not all are, nor are they always easy to find.

The best way to find a command line parameter for a silent installation is usually an Internet search. Websites dedicated to software deployment [1] may be helpful, and vendors may list silent capabilities in their documentation for the product. If a silent mechanism does not exist, repackaging the application into a deployable format (e.g., **.msi**) may be necessary; however, this should be avoided, if possible, for reasons beyond the scope of this paper. Additionally, command line parameters often allow the ability to specify options (in the form of switches/properties or "response"/configuration files) that typically are entered at install-time.

Proper testing is also an important component and ensures that software installations and deployments work correctly [5].

3.2 Managing Software Installation Packages

Key to managing automated software installers, even in a relatively unmanaged environment, are common software management factors like file organization, version control, and administrative access. In fact, it is a good idea to treat a system of automated installers like a software development project. Some type of revision control system is helpful (and necessary if several administrators contribute to the resource). The assumption is that these automated installers will be re-used frequently and used by more than one technician.

As a baseline, a central repository for software installation files should be used, and the automated installer support files either stored either those installation files or in an organizational system that parallels the software installation files.

Note that in Figure 1, the file structure for the automated installer support files parallels the file structure for the software repository that contains the source (i.e., original) install files. In Figure 2's structure, the automated installer support files can be integrated with the source install files. Either method can be used, although for EASI Make (see Section 3.2.1) and EASI, the file structure presented in Figure 1 must be used.

Having a neat, tightly organized file structure for installation files is good practice in any case, particularly when sharing with multiple administrators or groups as discussed in Section 3.4.

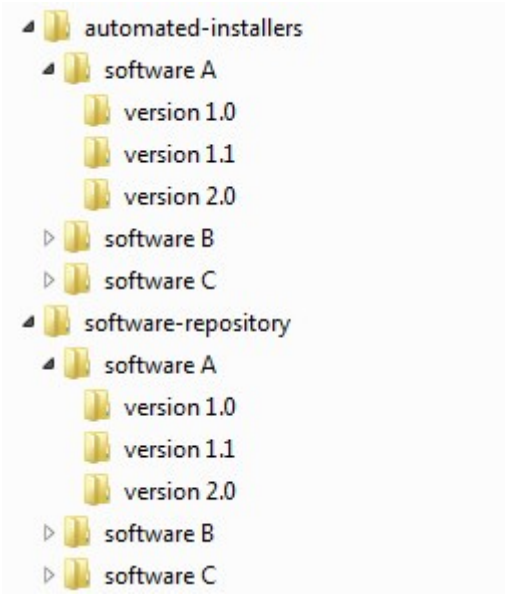

Figure 1. Parallel File Structure.

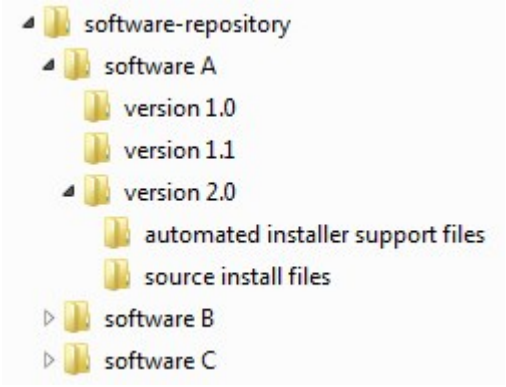

Figure 2. Integrated File Structure.

3.2.1 EASI Make

While there exist several commercial solutions for automated software deployment, an organization may not have the budget for one of these solutions. Accordingly, CSU created a tool to more easily manage the process of creating unattended installers (also called automated installers) called EASI Make. This tool takes as inputs the name of the program, the version, and optionally, components of the program, and outputs a batch script to a central store of automated installers. This script contains template code for running the automated installer, plus the specific information asked for in the EASI Make tool. The code to run the automated installers is simple; however, the administrative management of an automated installer infrastructure can be complex, particularly when multiple administrators are involved, and this helps reduce that complexity. One of the advantages of this tool is that it keeps the automated installer support files organized and consistent.

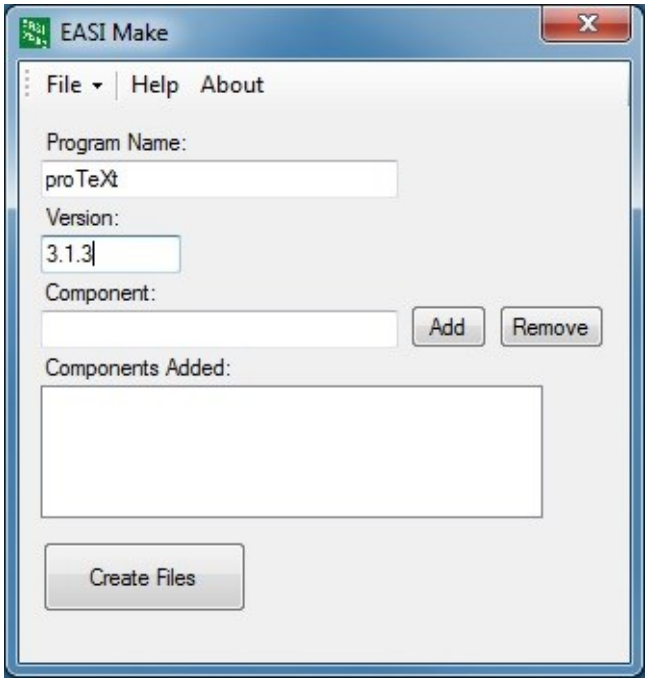

Figure 3. The EASI Make utility.

3.3 Bundling Software Packages

When packaging software installation files for custom distribution, it is convenient to re-package multiple installers as one single installer. There are multiple methods to accomplish this. The simplest is to use batch scripting (via Windows PowerShell, for example) and create a "wrapper" script that installs the desired software packages consecutively. Alternately, individual software installers can be re-packaged into new installer packages after changing installation options, and these new installer packages can be combined into one single installer package that installs multiple software packages from different original sources.

A commercial example of this type of service is Ninite [4], which allows users to select multiple common (and free) software installers into one installer package. The user then runs the combined Ninite installer, which downloads the specified software packages and installs them silently and automatically with Ninite-specified custom options. These options include using the default installation location, skipping up-to-date applications, skipping reboot requests from installers, and de-selecting toolbars and "add-on" programs. Such interfaces perform the functionality mentioned in Section 3.1 with the advantage of a convenient GUI.

CSU has developed a program called Engineering Automated Silent Installer (EASI) to perform a similar service. The technician selects from a list of available automated installers, and EASI builds a software package to install each selected software package. EASI creates a custom, on-the-fly PowerShell script from a template to run each selected installer consecutively. It is important to an error-free execution of this process that the installers are not run *concurrently*, since they may modify the same files on the computer. This "master" script simply calls each individual installer script, which can be a PowerShell or Windows Batch script.

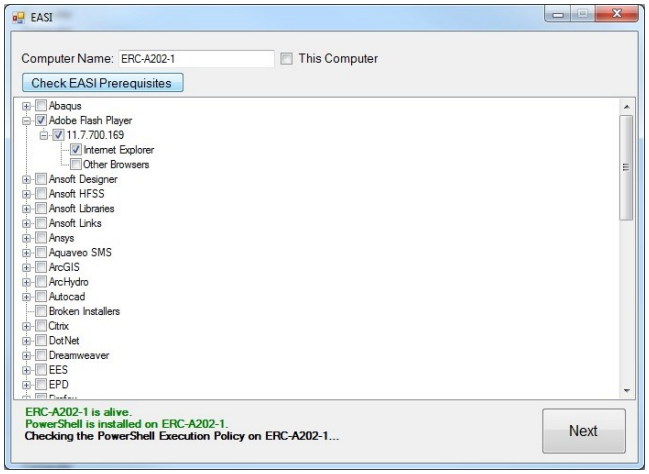

Figure 4. The EASI utility.

Rather than download the software provider's own installer packages, EASI uses automated installer packages that have been prepared by in-house technicians, so these installer packages are already customized to our environment. It should be noted that communication between technical team members is key for consistent operation of these scripts. In many cases, multiple automated installers exist for the same software packages, but with different options which suit different user groups or use cases. One example is a 32-bit version of the program versus a 64-bit version.

3.4 Sharing Packages

While most universities have a central IT organization, IT departments are generally decentralized for practical (e.g., regulatory concerns, specializations) or political reasons. With that said, almost all of these different organizations eventually install the same packages on their users' systems, potentially even with the same settings and options.

One way to prevent this duplication is to share installer packages. For simple applications, this is for convenience; however, for more complicated installs, this helps save resources as well as time. An example is explained below.

A centralized "install" share (e.g., install.university.edu) is accessible to any staff member who is considered an IT employee. The root of this share has folders denoting units or departments (referred to as division), e.g., UNI, COE, HBG. In this case, the UNI division is the abbreviation of the university that contains freeware and/or site-licensed "university" packages; the COE division is "College of Engineering" and may have additional subfolder division (e.g., EE for Electrical Engineering, CE for Computer Engineering); similarly, the HBG division is for the Harrisburg campus (typically only one division unless it has multiple IT departments). Creating separate divisions/folders for each IT unit gives the ability to granularly allow access to certain packages; there are several options (1 is always assumed).

1. Allow read-access to the UNI division for all IT staff.
    a. Since these are considered packages installable to all university-owned machines (e.g., Microsoft Office, Firefox), all IT staff should be able to use them.
    b. An alternative is to allow both read and write access so that other IT staff can contribute to this share; however, it may be more beneficial

to have them submit the package to the department hosting the share. This way, that department can quality-check the package to ensure it works correctly (and is properly licensed) before releasing it to the rest of the community.

2. Allow read-access to all other divisions for all IT staff.
    a. This option allows IT staff to see what packages other units have already created. For example, the Harrisburg campus may need to package SolidWorks (typically an engineering application); instead of packaging a duplicate, they could see and use COE's SolidWorks package.
    b. One problem with this method, obviously, is that the HBG might not be licensed to use COE's package (it also could be the case that COE's package has license server information embedded in the package and would not work on HBG's machines anyway).
3. Allow read-access to top-level division folders and write-access to sub-level division folders for members of a division.
    a. For example, only IT staff members who are part of COE (regardless of whether or not they are EE, CE, etc.) have read access to the COE division but not the HBG division. Packages in the COE division (e.g., MiKTeX, MathType) would be common and installable for anyone in CE or EE. The only IT staff having write-permissions to COE would be central COE IT staff.
    b. CE staff would have read-access to COE (i.e., common engineering) applications as well as write-access to their CE division (but not EE) for their own CE-specific packages (e.g., ActiveHDL).
    c. It is assumed that anyone having write-access to COE would also have write-access to CE, EE, etc.; however, this does not necessarily have to be the case if it is not desired.

Such granularity is relatively straightforward to set up if a centralized directory service (e.g., Microsoft Active Directory) is used in the university. Permissions to the divisions are best handled by assigning permissions to groups rather than individual users.

Finally, it may be helpful to have two primary shares (e.g., Deploy and Source) on the package file server. That is, \\install.university.edu\source and \\install.university.edu\deploy. Both have the same divisional structure (e.g., UNI, COE, HBG); however, the unmodified, original installer is placed in Source, while the modified, deployable version is in Deploy.

4. SOFTWARE DISTRIBUTION

The ultimate goal of creating silent and unattended installations is to deploy them via some mechanism, usually to automate the software installation process. In this section, we discuss two types of methods for doing so: unmanaged and managed.

The terms "managed" and "unmanaged", when applied to deployment methods, is different from a "managed" versus "unmanaged" computer. With regard to the operating system and software, there are various levels of managed computers: a public lab computer, for example, is at the high end of the management spectrum, while a laptop belonging to a faculty member might be at the low end, depending on each organization's policies. Every level of computer can benefit from the unmanaged deployment methods described here, but typically only highly managed computers can benefit from the managed deployment methods. That is because most of the managed deployment factors are dependent upon a number of conditions being met (other system software at the correct level and version, remote and administrative access enabled for the computer, etc.)

4.1 Unmanaged Deployment

In order to accommodate a wide variety of use cases, the automated installers are made as modular as possible. Thus, if a technician needs to simply install a program on a unique computer and not have to guide the installation manually, he or she may start the automated installer manually, then come back to the computer when it is finished. This alone improves the technician's efficiency.

A typical automated installer is a Windows PowerShell or batch script. The script can set environment variables and perform error checks (such as a check to see if the program is already installed) before triggering the silent unattended installer. The unattended installer is normally a **.msi** file containing the vendor's software installation package, along with command-line arguments [5]. One example is a silent, unattended installer for Adobe Flash Player:

```
msiexec /i "install_flash_player_11_plugin.msi" /qn
```

In a desktop support scenario, unmanaged deployment methods work well with both technicians in the field and at the bench. Unmanaged deployment methods may be used with any level of computer, from a tightly controlled public lab computer to a relatively uncontrolled laptop. For this type of method to achieve maximum potential, it is helpful to bundle software application installers (see Section 3.3).

4.2 Managed Deployment Components

Managed deployment, in this case, refers to some centralized mechanism of which end user computers are a part. This could simply be by joining a computer to the organization's domain or by installing a systems management agent on all University-owned computers. The goal is to automatically install (in this case) software without physically or remotely visiting an end user's workstation.

Since software installations occur automatically either when the machine starts or as a background process, this is why determining how to silently install applications via the command line is important. This command can then be added to a script or systems management application, which will then remotely run on the computer without interrupting their sessions (and without IT intervention).

Determining which software is applicable to a machine, i.e. "can be installed" is an important concept for implementing a truly automatic deployment process.

4.2.1 Systems Management Utilities

Systems management is the enterprise-wide administration of (often) distributed systems, typically from a single, centralized software interface. In most platforms a small software agent is installed on managed endpoints and communicates with the central server. Administrators of the central system can execute remote commands on client endpoints without having to physically or remotely visit the endpoint.

Several utilities exist, with varying levels of complexity, scalability, and cost. Common features may include built-in inventory, software and/or image deployment capabilities, patch management, and even power management.

IBM Endpoint Manager (IEM) is a commercial systems management tool compatible with multiple platforms (e.g., Windows, Mac, Linux). All managed endpoints are displayed in a single console regardless of whether or not they are members of the same (or any) domain, which is helpful.

IEM can be used to automatically and silently deploy software either immediately or at a scheduled time to managed endpoints. Besides having the ability to install software without interrupting users' sessions, an added benefit is not having to restart the computer to initiate an installation. If a software installation is optional, it can be deployed as an offer, which displays a window on the screen with a list of offered applications (a user can choose to accept or reject an offer; a user can also postpone an offer but then eventually have to accept it after a pre-determined time period). A tiered server structure conserves network resources, and installations occur locally on endpoints. IEM can typically perform any task that can be executed from the command line and also has built-in capabilities to determine whether or not an application has installed and then to retry the installation if it has not.

4.2.2 Group Policy

While purchasing a systems management tool certainly makes administration of large deployments easier and more organized, there is typically a large cost involved as well as significant learning curves and back end infrastructure to maintain. Such tools can be overkill or just simply not realistic for smaller organizations.

Organizations in a Microsoft Active Directory environment typically also use Group Policy Objects to centralize management of user settings (e.g., password policies, folder redirection, firewall management). Group Policy is a familiar interface that can also be used to deploy software in a managed fashion.

4.2.2.1 Software Installations

Software installations are a quick and easy way to deploy Windows Installers (.**msi** files). An application deployed via group policy automatically determines whether or not it needs to install on a system, and Windows Installers also provide a mechanism to determine if they are performing a new installation or simply updating an older version.

Typically, installing applications on lab computers (where installations are mandatory) is a computer-based policy with an assigned deployment type. For faculty/staff systems, optional packages can be user-based policies that "publish" an application; this allows the end user to decide whether or not he or she wants to install the application by using Add/Remove Programs.

To uninstall an application, simply remove the application from the software installation group policy.

One caveat to using group policy software installations is that the package must be in Windows Installer format. While many applications are in this format, many are not. Repackaging software installers as **.msi** packages can be time-consuming and error-prone. For installers that are not Windows Installers, a startup script can be used.

4.2.2.2 Startup Scripts

A startup script is a Windows script (e.g., .**bat** file) that runs when the computer starts (similarly, a logon script runs in the user context when a user first logs in to a computer). Startup scripts can be used to execute command-line installations for non-Windows Installer applications, though startup scripts can be used to install those, too.

Something to keep in mind when using startup scripts is that there are not any built-in mechanisms to determine whether or not an application needs installed. Thus, one will have to be written. One benefit of using a startup script, however, is that post-installation configuration can be built in to a single location.

For example, creating a startup script that installs Notepad++ might consist of the following. First, the silent installer command is needed; in this case, it is simply **/S**. The best way to determine whether or not Notepad++ is installed on a system is to see if its Uninstall Registry key exists; when Notepad++ is installed on a system, Windows creates the following registry key:

```
HKLM\Software\[WOW6432NODE]\Microsoft\Windows\CurrentVers
                  ion\Uninstall\Notepad++
```

Note that the WOW6432NODE is only if Notepad++ is installed on a 64-bit system (since the application is 32-bit).

Using this information and considering the deployment share described in Section 3.4, the complete deployment package and script can be created as follows:

1. Since this is a university-wide application, create the following folder:

   \\install.university.edu\deploy\UNI\NotepadPPxx,
   where xx is the version.

2. Put the installer in the NotepadPPxx folder, and then create a text file called installNotepadPPxx.bat. This will be the full deployment script that the software installation startup script points to.

3. To determine whether or not Notepad++ is installed, use the DOS **reg** command:

   ```
   reg query HKLM\SOFTWARE\Microsoft\Windows\
   CurrentVersion\Uninstall\Notepad++
   ```

4. If the above command returns 0, then Notepad++ is already on the system; if it returns something else, Notepad++ is not installed. Use the following command to check if Notepad++ is *not* installed:

   ```
   if %errorlevel%==1
   ```

5. The following command starts the silent installation of Notepad++ from the deployment location and waits for the installer to complete.

   ```
   start "" /wait "\\install.university.edu\
   deploy\UNI\NotepadPPxx\npp.Installer.exe
   /S
   ```

6. An additional check using the **reg** command to see if the application installed is optional, but might be helpful.

7. Post-installation processing of moving the Start Menu shortcuts can be used with the **mkdir** command (to create the new Start Menu category folder) and the **move** command (to move the shortcuts).

Make sure that the script works correctly. Additional scripting may be desirable to ensure that it eventually stops processing if there is a problem. Otherwise, the computer could hang at startup.

4.2.2.3 Permissions

One thing to keep in mind is that, natively, installations and scripts execute from a server location. One benefit to this is that adding or editing files and/or scripts is organized since they are in a centralized location. This location must be shared so that managed endpoints can access it, but it may not be a good idea to give open (or even authenticated) read-access to all objects in a domain.

Computer-based software installations and startup scripts execute using the built-in SYSTEM account, which is an operating system account used for background processes.

Using the folder structure discussed in Section 3.4, assume an organization places deployable software installers in a deployment share located at \\install.university.edu\deploy. The deploy folder should be shared read-only with *Domain Admins* and *Domain Computers*. Security permissions for the folder should let *SYSTEM* and the *Administrators* group of the local machine have full control, and only let *Domain Computers* have general read access (e.g., Read & execute, List folder contents, Read). Users will see that a *Deploy* folder is shared on \\install.university.edu; however, they will not be able to view its contents. Yet, because *Domain Computers* has read access, group policy software installations and startup scripts will execute correctly.

4.2.3 Client Scheduling

Another way to manage automated software deployment is for the client computer to initiate the process through a scheduled task. The advantage is that you can fine-tune each client and select when it should update based on a number of factors. For example, if you wish to have a classroom computer automatically install software when it is not in use, you may find it practical to only perform an update when it is guaranteed that no classes are scheduled in the room.

Like with group policy-based installations, you may also schedule a client to attempt automated updates on each reboot. The key is to set the computer to always perform this check, but leave the list of programs to install blank until you manually add it. This is especially useful for a scenario when a client user needs a software update urgently, but cannot leave the computer at the time the technician is available. (See Section 6.3.) In this case, the technician can remotely edit a control file on the client computer containing the list of programs to automatically install, then ask the client user to reboot the computer at his or her convenience. The computer will reboot, triggering the task which checks for a list of software to install automatically. Finding this list non-empty, it will perform the required software updates automatically, silently, and without any user or administrative intervention.

4.2.4 Control Files

A control file is often useful to manage and guide installations as well as to manage exclusions. The control file is normally a static text file placed in a designated location on the client hard drive. This file can include useful information such as the build version, build date, and a list of software that should be excluded from automatic updates. It may also include environment variables, such as the location to check for an automated update script.

For example, instead of relying on checking the registry for whether or an application is installed, writing a control file if the installation was successful is another way to know if the application was installed. Also, additional pertinent information pertaining to the installation, including a log of events, could be stored in the control file.

A control file can also be used to help determine flow. Group policies typically execute in an ordered fashion; however, if an earlier component failed, it might not be desirable to install another component later on. If the installer does not check for this dependency, a control file might be a good way to ensure that second component installs only if the first component is installed (i.e., the second component looks for the first components control file).

4.2.5 Master Script

An alternative to the control file is a master script which processes and redirects clients to the appropriate "master script" by a variable passed to the master script, such as IP address or computer name. The master script resides on a server, rather than the client, so that changes to the list of software to update, or even the installation process itself, can be made globally.

CSU has developed a series of modular scripts called "Unified Process for Distributing Automated Tasks and Executables Remotely," or UPDATER. UPDATER has evolved from a simple wrapper script to an optimized series of modular steps that include several error-checking components (learned from trial and error). UPDATER detects when an installation fails and, preventing the computer from ending up in an infinite installation loop, integrates with other timed events (such as Scheduled Tasks), differentiates between different OS builds, and provides detailed logging for troubleshooting purposes.

UPDATER's logic works like this:

1. Match the computer name (passed to the UPDATER script) to the correct group, so it can run the appropriate automated installers associated with that predefined group (e.g., "the GIS computer classroom").

For each software package:

2. Check to see if the program is installed; if it is, skip the installation of that program.
3. Switch to the alternate Windows logon screen (see Section 4.2.6).
4. Check the log for a previously "broken" installation of the same package (see Section 5.2).
5. Begin logging the installation attempt.
6. Call the individual software package's automated installer script.
7. Check for successful installation attempt and log appropriately.
8. Repeat steps 2-7 until it reaches the end of the list of software packages to install.
9. Re-enable the normal Windows logon screen, and re-enable logons.

4.2.6 Visible Status to the User

Typically, silent and unattended installations should not display information to the user. In certain cases, however, this may be desirable. One instance is during a series of several sequential installations (e.g., during initial provisioning or major updates), particularly when initiated remotely or via a managed deployment. During this time, computer performance may be impacted, and a user may not want to use the computer because of it.

Changing the Windows logon wallpaper [5] is one helpful way to indicate to users that a machine either is not ready to be logged into yet, or that updates are currently in progress and performance may be slower than normal. In some cases, we may not want users to log on to the computer at all until the installation finishes. In such cases, we temporarily disable logons to the computer. At the end of the installation process, logons are re-enabled, and the logon wallpaper is restored to the normal one.

5. SOFTWARE INTEGRATION

5.1 Post-Installation Scripting

Most of us will still use a "thin" or "thick" master image, deploy it to multiple computers, and use automated software deployment as a secondary method of distributing software (of course, automated software deployment can be used as the primary software distribution method, and that requires much more planning and testing). Typically, a script is run after the master image is deployed and a new hard drive has been cloned, to set variables such as the computer name.

Such a post-installation script is an ideal place to deploy an automated software installation script. The automated software installation script can install software based on certain variables (e.g., computer name, which may indicate which lab it belongs in), and can also be configured to install updates to software that may have occurred after the master image was made. This can result in a substantial decrease in the manual intervention of a technician to each computer.

5.2 Synchronization and Error-Checking

It will quickly become apparent that a complex script or set of scripts requires comprehensive logging features, to trace when there are problems. One problem we resolved via logging was when in this subset of the UPDATER workflow:

1. Check to see if the program is installed. It is not, so call the installer.
2. Switch to the alternate Windows logon screen and disable logons (see Section 4.2.6).
3. Call the individual software package's automated installer script.
4. The automated installation fails.
5. Go back to step 1. The UPDATER script is now in an infinite loop and the computer is unavailable to users because logons are disabled.

To solve this problem, we enabled detailed logging and error checks (see Section 4.2.5). In addition, when a software installation fails on a computer, that failure is logged centrally, that computer is flagged, and an administrator is notified so he or she can follow up.

5.3 Coexistence with Other Automated Processes

All computers have regularly-scheduled events and automated processes, some built into the operating system (such as operating system updates), and some created by third-party software (such as an update checker). In addition, administrators may have created other automated processes or externally-triggered events. For example, CSU public lab computers have a scheduled task which powers the computer down after it has been idle for a certain time, after a certain time of day. This feature saves power use and increases the longevity of the computer, so it has merit and cannot be discarded in favor of an automated software installation process. Yet, we found that our automated software update service was not being triggered when the computers were powered down, and we did not want the updates to run when the computer was powered on, because it would interfere with class time. Thus, we had to carefully orchestrate the order and timing of various automated processes on the same computer. This should be taken into account when planning an automated software deployment infrastructure.

6. MANAGING SYSTEMS

Managing systems (i.e., computers and the networks of which they are a part) is not a homogenous or even mutually exclusive experience, even if the systems are the "same" at a lower level. Faculty/staff systems versus computer lab systems is also not even a standardized battle.

Besides determining what software should install on what systems, there are several other factors to consider at a higher level than the system itself.

6.1 Consideration of Administrators' Needs

Waiting to make the occasional mouse click or answer a rhetorical question from an installation program is wasteful. Clearly, the efficiency gains from automated software deployment systems have the most direct effect on the productivity of the administrators and technicians for the computers involved. Automating any process results in greater efficiency, and software installation is no exception.

Consider, however, that efficiency gains to the technician who must install the software may have indirect effects on other IT administrators, such as the network administrator. Consider staggering computer updates so that the network is not overwhelmed with multiple, concurrent software installations from a single network share (See Section 6.2).

6.2 Consideration of Users' Needs

Computers should be separated into groups carefully; a one-size-fits-all automatic software deployment clearly does not make sense. Even for very common programs, such as a Java client, we have found that there is always at least one exception to the "everyone should update" rule. A granular approach to organizing groups of managed computers can require more time up front, but pay off over time.

One clear example is classroom computers. Of course, we do not wish to schedule software installations to conflict with a class. We also discovered that we did not wish to schedule software installations at the time the computer starts up, because it may have been turned off for the night (and often are, to save power), and they may be turned on just in time for the 8 a.m. class. A

software installation that takes 20 minutes and makes the computer unavailable during that time is inappropriate.

In public computer labs, it is disheartening for a user to walk in and find all of the computers updating their software at the same time, and no seat available. Thus, good practice is to stagger the automated software installations. This can be easily accomplished with the random delay feature of the Windows Task Scheduler.

In many cases, users are not able to surrender their computer when a technician is available. Normally, the needs of the software installation for the user will eventually trump the inconvenience to the user of not having their computer available for the time it takes for the installation, but this can be difficult to schedule for the user and the technician. With good automated software practices, this conflict is avoidable.

6.3 Service Desk Workflow

Many software installations or updates are initiated at the request of a user, often to solve an immediate problem. When a user contacts the service desk to request a software installation, the technician can determine if it is a managed or unmanaged computer. If manual intervention is required, the technician may log in to the client computer (remotely or in-person) and initiate the automated software installer. Alternately, the technician may configure the client computer remotely with the requested software installation packages, and ask the user to reboot the computer at his or her convenience, which will automatically initiate the software installation. (See Section 4.2.3.) If the computer in question belongs to a group of managed computers, the service desk technician passes the request to the managed computer's administrator, who schedules the automated update through the normal managed deployment. (See Section 4.2.) In all cases, the modularity of the automated installer process works to the service desk's advantage, because even in the least case, they have gained efficiency.

7. CONCLUSION

The goal of automated software deployment is to reduce the effort (and inconsistencies) involved when installing the same application on multiple computers. This paper demonstrated and detailed workflows and recommendations for converting traditional GUI-based installations to scripted, semi-scripted, or managed automated installations.

As discussed, automating software installations is an involved process; however, it greatly benefits computer provisioning and maintenance whether it is used as a supplement to traditional methods or as the only method. This benefit not only extends to IT administrators (who spend less time servicing installs) but end users (who receive faster service) as well as the organization in general (which receives more consistent builds and updates).

8. REFERENCES

[1] Dell Inc., "ITNinja (AppDeploy)," [Online]. Available: http://www.itninja.com.

[2] IBM, "IBM Endpoint Manager," [Online]. Available: http://www.ibm.com/tivoli/endpoint.

[3] Microsoft Corporation, "Group Policy Software Installation Overview," [Online]. Available: http://technet.microsoft.com/en-us/library/cc738858(v=WS.10).aspx

[4] Secure By Design Inc., "Ninite: Install and Update All Your Programs at Once," [Online]. Available: http://www.ninite.com.

[5] J. B. Tyndall, "Building an Effective Software Deployment Process," in *Proceedings of the 40th Annual ACM SIGUCCS Conference (SIGUCCS '12)*, New York, NY, 2012.