

# Building a Custom Installer for Cisco Unity – An Experience Report

---

**Lita Chakma, Kavyashree Keelara, Sanjay Lakkad, Trupti Mande**

Seattle University,

Seattle, USA

{chakmal, keelaras, lakkads, mandet}@seattleu.edu

**Project Guide:** Dr William Poole,  
Chair and Professor,  
Department of Computer Science,  
College of Science and Engineering,  
Seattle University,  
Seattle, USA

**Project Liaisons:** Darren Massey, Harvinder  
Chowdhary, Glen Frison, Jason Swager,  
Cisco Unity Project Team,  
Cisco Systems,  
Seattle, USA

## **Abstract**

Cisco Unity patch distribution required a custom designed installer that can seamlessly deploy or update over an existing unity installation. In past, this process has been largely manual and required fair amount of plumbing. To address this issue we devised a software package that automates and streamlines entire process of patch generation and packaging, greatly reducing burden from human counterparts. In this paper we have described the requirements from Cisco Unity Product team, design that we have come up with, implementation, and analysis of what we have done. We have also included a comparative advantage of our approach, user testimonial about the product and possible feature extension scope.

## **Keywords**

Patch Installer, Wrapper, Bootstrapper

## **1. Introduction**

Software Installation and distribution is an integral part of software life cycle. In most cases software is usually developed by a small group of people, tested on a small sample subset of the target machines and then released to be used by a large audience. In addition, the software life cycles spans beyond the development of first release of the product.

Software maintenance has assumed as much if not more importance than the initial development of the product. Even after the release of the product, updates in terms of new features and bug fixes need to be made frequently over the life time of the product. Hence, we need a software distribution mechanism that handles not just the initial installation of the product but also subsequent updates.

In the past, installation was a post production activity which was viewed as a procedure to copy file(s) to a particular location(s) and installation program was an afterthought. This coupled with the absence of stand-alone installers that were agnostic of the software they were installing, led to product teams developing their own custom installers that launched and maintained their products. These custom installers were tightly coupled with the products and could not be easily adapted to different software products.

Over the life time of the product the installers were required to distribute patches of different kinds, which meant that they needed to be updated in order to accommodate these new kinds of patches. With the advancement of computer technologies, software distribution methods have also changed. The custom installers had to be flexible enough to support these changes. For example: Software packages are now distributed both on CD's and through the internet. These continuous improvements to the custom installers have made it difficult and expensive to maintain.

In today's world, companies aspire for an installer product that can be economical, ready to use, operating system compatible, platform compatible and maintainable thereby allowing companies to concentrate on their products rather than the installer products.

In this paper, we are going to present our experience about the Patch installer for Unity, a Cisco System's product. Cisco Unity is a powerful Unified Communications solution that provides advanced, convergence-based communication services such as voice and unified messaging on a platform that offers the utmost in reliability, scalability, and performance.

The existing patch Installer that Cisco's Unity team was using was plagued with all the above mentioned problems. The custom installer that the developers have initially come up with was ill equipped to handle the needs of the product team. The patch making mechanism was becoming a hassle. In order to work through these problems we decided to design a new patch installer for the Cisco project team.

Our goal with the experience report is to provide a starting point for other project teams who are evaluating their options for a patch installer. This report offers

- Detailed information about the installer problem we encountered
- A generic solution to the above problem
- Pros and Cons and tradeoffs that influenced the solution design
- Software Engineering methodology used
- Analysis of the new design

## 2. Problem Statement

Cisco Unity releases close to a hundred ES's<sup>1</sup> over its supported lifetime, i.e. between Unity version upgrades.

The objective of the project was to develop a script based patch installation system, which used a standardized installation technology like Windows installer, for the Unity Product. The decision to replace the existing patch installer was made due to the following reasons:

- ❖ Cisco uses their own home grown patch installer that uses CRC mechanism to install patches. The CRC mechanism also means that Cisco cannot install any new files with new patches. Current installer can only patch files but it can not add or remove files.
- ❖ Currently, for Cisco, combining ES's is a completely manual process - for both Unity developers and the end-users. As the numbers of ES's released has grown, the numbers of dependencies between files have also increased. Due to this the manual management has become complicated, tedious and cumbersome.
- ❖ Creating an ES was difficult and was becoming labor-intensive, and ensuring that the affected files are identified correctly has also become extremely difficult. Some patches for Unity have had incomplete or incorrect files.
- ❖ Unity installation is done by remote client on the client machine. In case of errors, due to insufficient logging facilities, it is highly difficult for user to understand the reason behind the failure
- ❖ If an ES patch is reinstalled and then uninstalled, the system will not restored to its original state.
- ❖ The installation wizard is tightly coupled with the Unity installation environment, so it cannot be used to upgrade files that are not on the Unity server but are involved in Unity feature operations.

Here is an example of a common scenario the Cisco Unity product team and customers faced.

- ❖ Suppose Cisco Unity needs to release a new patch to apply to their existing installed product on the client machine and as part of this patch they had to update a file foo.dll and place two new files - firstscript.cde and bar.dll on the Unity file system. Using the existing patch installer, only the updated file foo.dll could be delivered to the client machine. It was not possible to install the new files firstscript.cde and bar.dll to the customer via the patch installer. These files had to be packaged separately and copied over manually to the client machine.

---

<sup>1</sup> \* A patch for Cisco Unity system is known as Engineering Special (ES).

### **3. Literature Review**

#### **3.1 Software installation methods**

Software installation is the process of making an application / software system part of a computer. Over the years the techniques used to install software has evolved. Listed below are the most common software installation methods.

##### ***3.1.1 Simple Copy***

One of the methods of installation is basically copying files to a target location on the system. Copying files can provide the necessary environment settings for an application to run. This type of installation is typically performed by a batch file or other scripting languages like VBScript or JavaScript. Disadvantage of this method is that applications installed using this method do not show up in the Add/Remove Programs control panel applet and usually require manual uninstallation either by a batch file or manually deleting files. This method of installation also does not handle custom user configuration without additional programming.

##### ***3.1.2 Script Based (Setup.EXE) Installation***

Copying files for an application on the target system can be time taking and inefficient. To save application developers time and effort for creating software packages, frameworks like InstallShield could be used to build an installation package using a GUI rather than building the installation script from a text editor. The end result of these kinds of tools is a compiled executable, usually setup.exe that contains all the files, settings, and logic necessary to install an application. This method allows easier customization during installation by the end user. One of the benefit of this method is that the application will then appear on in the Add/Remove Programs control panel, making it easier for user to remove the application.

##### ***3.1.3 Windows Installer Installation (.MSI)***

Window Installer was introduced by Microsoft with the release of Office 2000, for application installation on the Windows operating systems. In this technique the software product to be installed is packaged into a Setup file or MSI file. This MSI file is like a database in that it contains everything required to install an application. Like script based installation, Windows Installer also makes the software installed available on the Add/Remove programs control panel. Windows Installer also allows the application developer to customize the installation with user defined custom actions. Other benefits of the Windows Installer over script based installation include – ability to customize / choose the features that are installed, ability to repair a software installation using the reinstall option, ability to install components that can be shared between applications, and installations that can identify and upgrade previous versions of an application.

##### ***3.1.4 Java Based Installation***

This method is typically used for cross-platform deployment and employs a method similar to the script based installation using the java runtime environment to install files and configure settings. This method can include entries in Add/Remove Programs too. The drawback of this installation method is that this method requires the Java Runtime environment to be installed on the workstation on which the software product is to be installed. This method allows application developers to develop a single installation package that can apply to multiple platforms.

### 3.2 Software Installation method used in the new Patch Installer for Cisco Unity

Our goal was to develop a patch installer for the Cisco Unity product. Considering that the Unity product is tied to the Windows Operating system, and that not all client machines may have the java runtime environment installed, we narrowed down the research to studying the pros and cons of InstallShield and Windows Installer. After considering various feature trade-offs and long term implications of going with one over the other, Windows Installer was selected as the base for the new patch installer. Some of the differences between InstallShield and Windows Installer that weighted in on our decision are:

<b>InstallShield</b>	<b>Windows Installer</b>
Usable on all platforms including Windows 9.x	Usable only on Windows NT SP6, Windows 2000, and Windows XP for Release 3.6
Detects and uninstalls an older VPN Client.	Detects but does not automatically uninstall an older VPN Client. Remove previous versions via Add/Remove programs.
Provides a proprietary installation package and customizing process.	Provides a standard installation package and customizing process.
Silent installation suppresses all dialogs and messages, including errors.	Silent installation can be customized to include error reporting.
Provides no automatic rollback when installation fails.	Provides automatic rollback in case of installation failure; undoes changes to the system made during attempted installation.
No automatic replacement of deleted or corrupted files upon first use	Automatic replacement of deleted or corrupted files upon first use. Replaces registry keys associated with shortcuts under Start   Program Files.

Courtesy: [http://cisco.com/univercd/cc/td/doc/product/vpn/client/rel4\\_0/admin\\_gd/vcach7.pdf](http://cisco.com/univercd/cc/td/doc/product/vpn/client/rel4_0/admin_gd/vcach7.pdf)

## **4. Implementation**

### **4.1 Design Overview:**

We have come up with a general design which can be helpful in solving problems as stated in the requirement/problem statement section. It involves creating an MSI and a wrap around it called bootstrapper. MSI is generated using open source tool, WIX, which is responsible for installation and un-installation of the patch. We used open source Windows Installer toolset which is most commonly known as WIX (Windows Installer XML). The reason behind choosing this tool set is that it is not only an open source installer but it hides all the complexities of Windows Installer from the user by letting to use simple XML schemas. Compared to other open source installers, WIX greatly simplifies Windows Installer package generation by keeping a similarity with any typical application generation process like writing a program, then compiling and linking it to create the ultimate output in some executable format. Because of all these reasons we chose to use WIX over legacy tools.

Event though an installation package is created by WIX and Window Installer, whole process is controlled and synchronized by the wrapper unit Bootstrapper. Bootstrapper also is responsible for performing pre-requisites check, backing up and restoring back files, creating log files, installing scripts files if there is any etc. This provides a lot of benefits like creating a separate custom log file over windows event log make it easier to investigate failure. Installing scripts by bootstrapper makes it more secure as the risk of scripts getting changed by MSI is reduced etc. To lessen the customers' burden of dependency issue, we chose to implement it in C++. On the other hand C# is used to increase the easiness of development of package generation.

### **4.2 Design Details:**

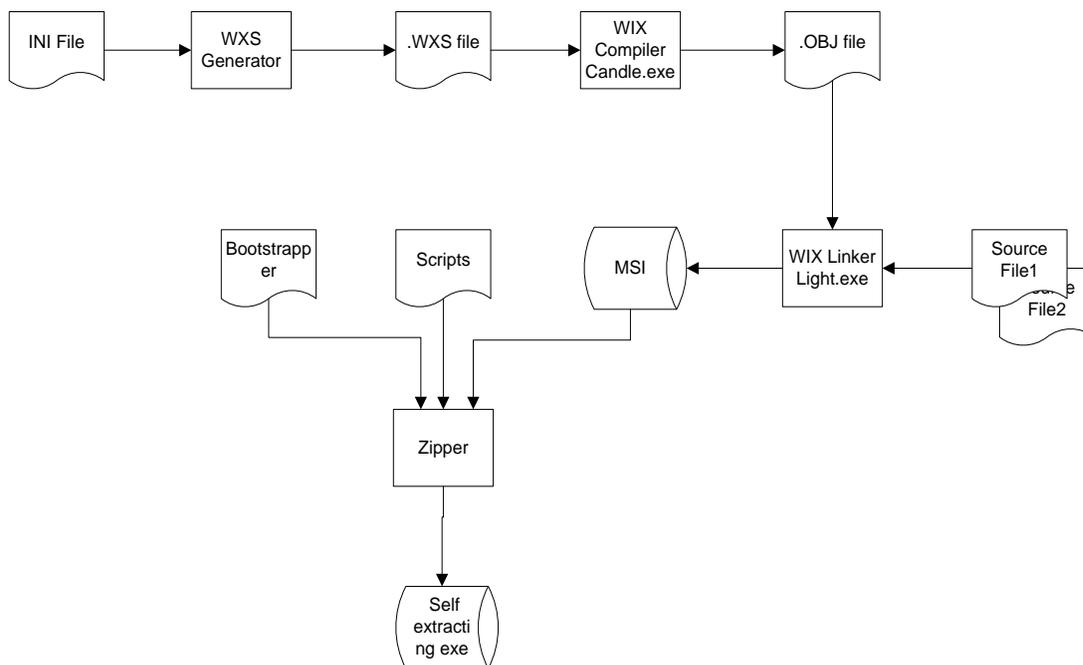
Streamlining patch generation process involves working with several disparate components, unifying them under a general process design and producing a self contained service pack module. We identified that there are two distinct life lines involved here, one is the build process of the service pack which operates in the build system and other in the runtime environment of the patch which operates in customer machines. While both of these lifelines are separated in time and space, they have to be synchronous in their behavior; hence they are not totally independent. Build process and runtime execution process are explained later in the section.

#### **4.2.1 Build Process:**

The entire build process is essentially a complex state machine. The INI file and patch binaries work as the input. INI file contains some feature description of the product like version number, file name, installation location of the files in target machine, GUIDs, indication of whether the file is to be updated, reboots required, scripts inclusion etc. Based on INI file, WXS generator emits a WXS file which is basically an xml version of MSI/Windows Installer database schema. WXS files are feed into Candle.exe (WIX compiler) which outputs the corresponding WIX object file. With object files

generated successfully, we move on to the link process which is performed by another WIX tool Light.exe (which basically is a linker), Light.exe operates on both source files to be patched and object file and generates a self contained MSI file. MSI file is able to install and uninstall the files into the target machines. But that process is started by a separate component called Bootstrapper. Bootstrapper basically works as the controller for the installation and installation process. This requires a second layer of packaging which done by the zipper. Zipper takes Bootstrapper, MSI file, and scripts if there is any to create a self extracting exe and which is the final output of the build process.

A data flow diagram of the build process is included here for clarity of understanding.

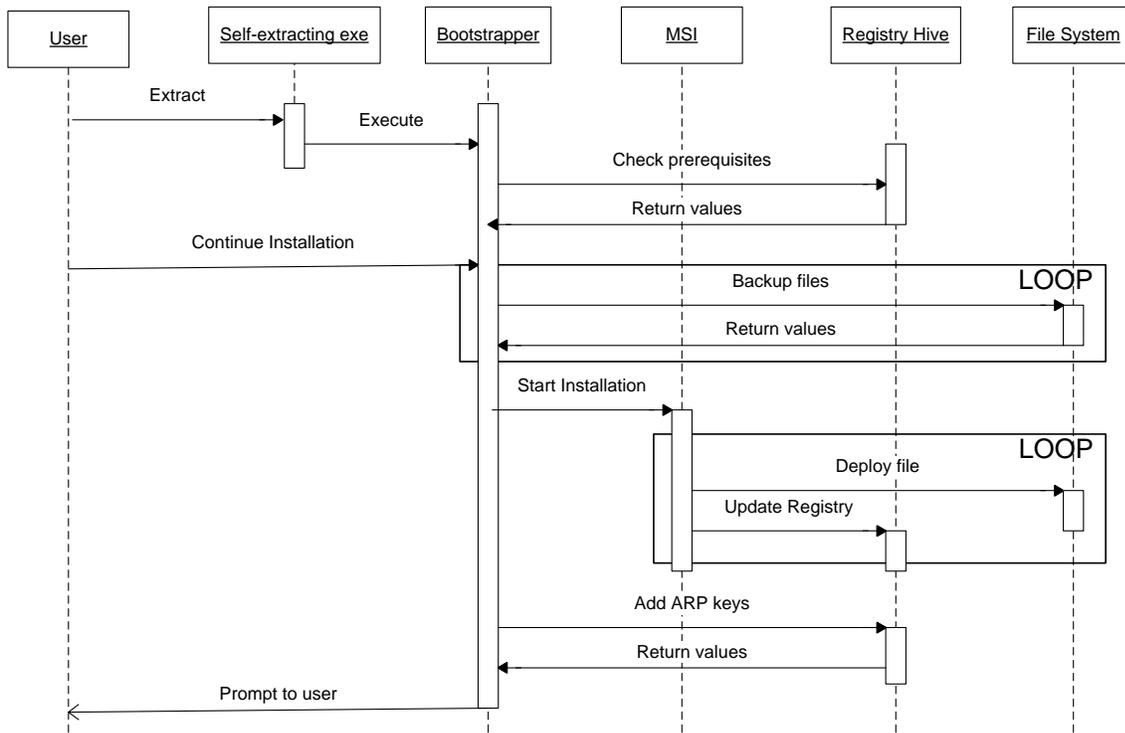


**Figure 1 : Patch Generation Data Flow Diagram**

#### **4.2.2 Execution process:**

There are two main phases in execution process: install and uninstall. Both can be best understood by sequence diagram which are given below.

##### **4.2.2.1 Install Process:**



**Figure 2 : Sequence Diagram of Install**

Installation is initiated by the user activating on the self-extracting executable (e.g. by double clicking or pressing enter). Once the self extracting executable is run, it deflates the contents into a local folder and runs the Bootstrapper. Bootstrapper cross checks the environments, validates pre-requisites. If pre-requisites are met and the user chooses to continue installation process, it backs up existing file if there is any. MSI module places the files in proper location and updates the registry as necessary. In the end of MSI installation operation, an entry is made into the Add/Remove Program for uninstallation option.



update the product. The solution that we came up with was to always consider each patch release as a new product. This, however, brought up another issue. During the uninstallation of any product, Windows Installer removes all files that it had previously installed on the local machine. In the same vein, if Windows Installer was used as is to install a patch as a new product during uninstall of the patch, it would delete all files that it had installed or updated as part of the patch install from the local machine. This did not fit well in situations where Windows Installer was used to update a file. If Windows Installer were to remove these updated files from the file system during uninstall, the product would be in an unstable state. This is where the bootstrapper played an important role. During the creation of the patch, the packager recognized which files were being newly installed on the customer machine and which ones were being updated and created packagers in such a way that the Windows Installer only uninstalled the newly installed files during the uninstall. The responsibility of restoring the updated files to their original state was given to the bootstrapper.

The old custom installer used the CRC mechanism to install files. This posed a limitation that the product team could not release new files via patches. If a patch required a new file to be installed on the customer machine, this had to be done manually i.e. the new files had to be copied over to the customer's machine manually. The new patch installer uses Windows Installer to install all files, and Windows Installer does not distinguish between old files and new ones. This makes it possible for the product team to release new files via patches.

The troubleshooting experience with the old patch installer did not meet the requirements of the product team. They needed more robust logging capability when patches are created, edited, installed, more so at the customer end where bringing up a debugger to see what went wrong is not always an option. The Customer support representatives of the product team are expected to be able to debug a failure just by looking at the logs. Hence in the implementation of the new patch installer, special attention has been given to the product's logging capabilities.

The design of the packager and the integration of the wrapper with the Windows Installer has been made keeping extensibility in mind. If the product team decides to use InstallShield instead of Windows Installer in the future, the switch can be made easily with little changes to the wrapper and the packager.

One of the long term goals of the product team is to use Windows Installer to install the product in subsequent version releases of the product. Since the patch installer uses Windows Installer behind the scenes to install patches, when the product teams makes the move to install the product itself using Windows Installer, the changes needed to the patching mechanism will be minimal and simple. Only minor modifications will be needed is to not package the wrapper along with the MSI files.

A big concern in this project was that the new patch installer was going to be developed and maintained by different people. So the code and the design had to be easy to understand, maintainable and well documented. Hence when we evaluated the designs that we had come up with for this new patch installer, we went with the design that was the most intuitive.

Currently the patch installer is being used by Cisco internally. This gives us an opportunity to find and fix any bugs in the product before the installer is released to Cisco Unity's customers for external consumption in August.

### **5.1 Summary of new features added to the patch Installer**

The new patch installer possesses all the functionality seen in the existing installer. In addition, the new patch installer brings the following new features:

1. Discard the old CRC mechanism of installing files.
2. Reduce the cost of maintaining the patch installer by using an out of the box installer to install the files.
3. Make it possible to install new files in addition to updating existing files.
4. Improve logging during the creation of patches.
5. Improve logging at the customer end while installing a patch
6. Add logging at the customer end while uninstalling a patch.
7. Make it possible to silently reinstall a patch (previously if a patch was reinstalled and then uninstalled, the system was not restored to its original state. This is fixed now).
8. Make it possible to roll up patches.
9. Design the patch installer to be extensible so that moving towards an out of the box installer to install the whole product becomes easy.
10. An Object Oriented implementation of the product

### **5.2 User Testimonials**

Since the patch installer is being used internally in the product team we have been able to collect some information about how the users of the patch installer rate the new patch installer against the old custom installer. To gather this data, we prepared a questionnaire for the users which comprised of questions that can be broadly categorized into the following buckets:

1. Usability of the product in terms of ease with which patches can be created, ramp up time for existing users etc.
2. Ease with which the new patch installer can be plugged into the existing product environment
3. Extent to which specific requirements of the product team are met in terms of additional functionality, extensibility, troubleshooting experience.
4. Quality of the product – maintainability and readability of the new patch installer code.

The complete questionnaire can be found in Appendix B.

The target audience for this exercise was limited since only a few developers/ testers who were aware of the internals of the old patch installer system and were in a position to evaluate the new patch installer system. This is because of the way in which the patch installer system fits into the rest of the build environment of the product. The patch installer's packager system is tied to a front end Web System that hides all the complexities of the packaging mechanism. All developers and testers who want to create patches have to do is input the files that are to be a part of the package. The new patch installer system will continue to work in the same way with the front end Web System. So most users will not be aware of the how the packages are actually built. The only users who will be aware of the patching mechanism are the developers/ testers who maintain the patch installer.

From the survey we conducted via the questionnaire, the audience's opinions are:

When it comes to creation and edition of the ES, audience's the answers varied from same as before to better than before. Since the creation and edition of an ES Package is hidden behind the front end Web System, the audience felt that the users would not see any changes at the package creation end. However, at the Customer end, the audience felt that things had improved over the existing patch installer. The fact that a customer can go to add remove programs and uninstall a patch, instead of running uninstall batch files as was previously done, was considered a big plus.

The audience felt that the new patch installer integrates with few visible changes to the existing product build environment and hence it should be moderately easy for the existing users (developers/ customers/ customers) to ramp up to use the new patch installer.

The target audience voted that the new patch installer has more of the features they need than the existing installer. In particular the ability to install new files, better error recovery in case of abnormal configuration, use of standardized technology (MSI) rather than relying on home-grown patching and scripting mechanisms were considered the big pluses. The audience also felt that though the usage of Windows Installer and bootstrapper was not an optimal way to use Windows Installer, the new implementation of the patch installer is still better than the existing patching mechanism. They also felt that the use of bootstrapper with the Windows Installer makes the patch installer more stable, maintainable and gives us more control over the patch installation process. In end, it will be easier for them to add new functionality to the new patch installer than to the existing patch installer.

With regards to the quality of the new patch installer's implementation, the audience felt that the readability and maintainability of the new patch installer is better than the existing patch installer.

In addition to questionnaire, we considered the option of conducting a focus group and letting users compare the existing patch installer with the new one. The reason this was not pursued was that we needed educated users familiar with the Cisco Unity environment and patch processes to participate in the focus group. Because of logistics and the changes that were required to make the

front end Web System to work with the new patch installer it was not possible within the timelines of the project.

### 5.3 Code Complexity Analysis

Since easily maintainable code was one of the requirements of the project we have run some code complexity analysis over the parts of the old patch installer and the new patch installer that ran on the client systems. A code complexity analysis was not made on the part of the system that creates the patches as the new patch making system was written in C# previous version used assembly level language and C++.

The open source tool “Source Monitor” was used to obtain these code complexity numbers.

	Lines of Code	% Branches	% Comments	Number of functions (not part of an object)	Average Complexity	Maximum Complexity	Average Depth
Old Patch Installer	2954	11.0	15.8	17	2.67	27	1.29
New Patch Installer	5090	8.9	18.8	0	2.9	24	1.36

1. Though the new patch installer adds more lines of code at the wrapper end, it significantly reduces the number of lines of code required to create a patch. Most of the additional code enables new functionality in the patch installer.
2. The new patch installer does better than existing version when it comes to the number of branches in the code and the number of functions in the code because the new patch installer adopts an object oriented approach.
3. The new patch installer has more comments than the previous installer to address the concern that the patch installer is developed and maintained by different groups of people. There has to be enough information for other group to maintain the product.
4. The code complexity of the product is considered to relate to the number of bugs that are expected to be found in the new product. Though there are no industry standards that prescribe what the code complexity of a product should be, staying towards a smaller number is recommended. According to source monitor metrics, it is recommended that the code complexity should stay within 8. The closer it is to 2, higher the quality of code. The average complexity of the new patch installer is about 2.9.
5. Two areas that the new patch installer needs to improve on are maximum code complexity and average statements per method. To address these problems, we have walked through the code of the new patch installer and identified method that need to be re-factored.

The new patch installer performs reasonably well on the maintainability front. It should also be kept in mind that this performance comes along with several new features that the new patch installer brings to the product's patch system.

## **6. Conclusion**

The Cisco Unity product used a home grown patch installer as its patch delivery system. This custom installer was developed before installation products like InstallShield and Windows Installer became popular. It used a CRC mechanism to install files. This meant that the patch installer could not be used to install new files. The implementation of the patch installer was also tied to the Unity product. As a result the installer could not be used to patch other unity related products. The regular maintenance of the patch installer, fixing bugs in it and adding additional functionality to it was becoming prohibitively expensive. Over the lifetime of any Unity version several hundreds of patches are released. Maintenance of these patches had also become a hassle.

To overcome all these issues with a custom installer, we analyzed the different patch installation software available in the market. After analyzing the pros and cons of popular installation solutions, in particular InstallShield and Windows Installer, and keeping in mind the requirements of the Cisco product team, we picked Windows Installer as the base for the new patch installer. We analyzed the patching and product installation mechanism employed by Windows Installer, the host environment for the Cisco Unity product, and the existing Cisco Unity build environment. After the analysis, we came up with a design for the new patch installer that used i) packager to create patches and ii) Bootstapper in addition to Windows Installer to install the patches on the client machine.

While arriving upon this design solution for the new patch installer for Cisco Unity, we realized that this problem is not specific to Cisco Unity. Several software products that were released to market before independent software installation solutions like InstallShield and Windows installer become popular will be facing a similar situation. A product team might have decided to use a home grown software product and patch installer to distribute their software product and to maintain it, for any number of reasons. But over the time, these product teams might be realizing that these custom installers do not meet all their needs from an installation mechanism. Adding these additional functionality to their existing solutions and maintain them might be turning out to be expensive and cumbersome. Our attempt with this experience report is to highlight that a simple solution exists for such a problem. We believe that our design for Cisco Unity's patch installer can be adapted to a wide number of other similar products and that this paper is a good starting point for any product team that is contemplating changing to their custom patch installer.

### **Future work:**

- As with as any software product, the design of the new Cisco unity patch installer has evolved with time. Several design time assumptions were proved wrong during implementation; hence we had to find workarounds to make sure that the patch installation

proceeded in the way we needed it to. As a result of this the code structure of the new patch installer has become less robust. Hence there is a need to revisit the detailed design and implementation of the new patch installer and to refactor some classes and methods.

- As mentioned above, the new patch installer that we have come up with for Cisco Unity can be adapted to apply to several other similar situations. Generalizing this new patch installer to apply to additional products will help more product teams.
- Today's popular installers like InstallShield and Widows Installer cannot be used directly to solve problems similar to the problem the Cisco Unity product team was facing. Analyzing the design of these popular installers in more detail and proposing modifications to these popular installers to make them more adaptable is also an interesting work item.

## **Acknowledgements**

We thank the Cisco Unity Project Team and Seattle University Faculty for giving us the opportunity to work on this project. We really appreciate the time and effort with which the Cisco Team and Dr. Poole guided us through a year long project. We also appreciate the Cisco Team for giving us the leeway to come up with our own design for the new patch installer and for helping us work through many obstacles that we encountered during the implementation phase of this project. Their technical guidance was extremely helpful. We thank Dr Poole for his guidance and insight throughout this project.

## References

- <http://www.cisco.com/en/US/products/sw/voicesw/ps2237/>
- Internal Process Details document for Cisco Unity's existing patch installer, build environment
- <http://consumer.installshield.com/utilities.asp>
- [http://cisco.com/univercd/cc/td/doc/product/vpn/client/rel4\\_0/admin\\_gd/vcach7.pdf](http://cisco.com/univercd/cc/td/doc/product/vpn/client/rel4_0/admin_gd/vcach7.pdf)
- <http://www.installsite.org>
- [http://msdn.microsoft.com/en-us/library/aa369425\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa369425(VS.85).aspx)
- <http://wix.sourceforge.net/>
- <http://www.winzip.com/prodpagese.htm>
- <http://www.campwoodsw.com/sourcemonitor.html>
- <http://www.visualsvn.com/> [Tool used for version control of software product]
- <http://tortoisesvn.net/> [Client view source control]

Numerous MSDN library pages were also used as reference during the entire software implementation stage.

## Appendix A

### Glossary

---

ES Installer     Engineering Special Installer

ES Package     Engineering Special Package

## Appendix B

### Questionnaire

---

Please highlight the answer that best fits the following questions. If more than one answer apply please highlight them all.

Please rate the new ES Installer (that uses Windows Installer) against the existing ES Installer (that uses CRC mechanism) with regards to:

#### Usability:

1. Ease of creation\edition of ES's (from developers and testers perspective)
  - a. Better than before
  - b. Same as before
  - c. Worse than the existing one

Additional Comments:

2. Ease of installing/ uninstalling the ES Package on client machine (customer perspective)
  - a. Better than before
  - b. Same as before
  - c. Worse than the existing one

Additional Comments:

3. Ramp Up for existing users (customers, developer, testers)
  - a. It will be very easy for existing users to learn to use the new ES Installer
  - b. It will be moderately easy for existing users to learn to use the new ES Installer
  - c. It will be difficult for existing users to learn to use the new ES Installer
  - d. It will be very difficult for existing users to learn to use the new ES Installer

Additional Comments:

#### Plugging into the existing Unity build environment:

4. Ease with which the new system will integrate with the rest of the existing ES Build environment (ES Website, ES build automation system).
  - a. The new ES Installer will work seamlessly with the existing ES Build environment without requiring any visible (visible to the end user – Cisco Client and Cisco developers and testers) changes to the environment
  - b. The new ES Installer will require a few visible changes to be made to the build environment in order for it to work with the rest of the system
  - c. The new ES Installer will require a large revamping of the existing build environment in order for it to work with the rest of the system

Additional Comments:

### Specific requirements:

5. Functionality of the installer (installing new files, updating existing files, running scripts)
  - a. The new ES Installer provides more functionality than existing ES Installer
  - b. Same as before
  - c. The new ES Installer provides lesser functionality than the existing ES Installer

For this question please also list down what the new features are if there are any.

Additional Comments:

6. Using an out of the box installer like [Windows Installer + Wrapper] to drive the installation **Vs** using a [home grown algorithm] (like CRC's) to install files
  - a. [Windows Installer + wrapper] makes the product more stable and maintainable.
  - b. Does not add any value to the installation mechanism.
  - c. [Windows Installer + wrapper] makes the custom installer more difficult to use and maintain.

Additional Comments:

7. Using [Windows Installer] out of the box **Vs** Using [Windows Installer + Wrapper ](please choose one or more options as applicable)
  - a. Using [Windows Installer + Wrapper] makes the design of the product **more** complicated than if we had used only Windows Installer.
  - b. There is no advantage is using one over the other
  - c. Using [Windows Installer + Wrapper] makes the design of the product **less** complicated than if we had used only Windows Installer.
  - d. Using [Windows Installer + Wrapper] gives **more** control over the installation process.
  - e. Using [Windows Installer + Wrapper] gives **less** control over the installation process.

Additional Comments:

8. Extensibility of the product.
  - a. New functionality can be added to the new ES Installer more easily than the existing one.
  - b. No difference
  - c. It is more difficult to add new functionality to the existing ES Installer than it is to add functionality to the exiting installer

Additional Comments:

9. Troubleshooting experience (finding out what went wrong without having to bring up the debugger using only log files)
  - a. Better than before
  - b. Same as before
  - c. Worse than the existing one

Additional Comments:

### **Quality of the product:**

10. Readability of the product (how easy/ difficult it is follow the flow of control in the program).
  - a. Better than before
  - b. Same as before
  - c. Worse than the existing one

Additional Comments:

11. Maintainability (comments, breakup of functions etc)
  - a. Better than before
  - b. Same as before
  - c. Worse than the existing one

Additional Comments:

### **General Comments**

If you have any other comments for this project please feel free to pen them down in this section.