Week 8 Report

Group 1: ITL Enabler Installer

Jacob Copeland (#16430978)

Siqi Zhao (#17140639)

Giovanni Saberon (#00100188)

Jaime Curnow (#03336247)

Table of Contents

Process

Updated Project Plan

As can be seen on our GitHub project Kanban board, every single unfinished issue is now mapped to a resource (person). These are prioritized with issues relating to either BOTH Client and Server, or just Client, as high priority (there are only two of these left which need to be scripted, and three others which would be integrated if not for bugs). Server-only modules are assigned to Giovanni and are also being scripted - while server was low-priority to begin with, it is still being worked on to ensure everything gets done.

Completed tasks are as follows:

- Create logging capability
- Create fully functional User Interface/Wizard
- Install .NET 4 and .NET3.5 runtimes
- Finish an install after a reboot
- Get Registry/File system values
- Install User documentation/manuals
- Setup Environment Variables and Registry keys
- Instruct the installer to reboot under certain conditions
- Install Enabler Drivers
- Remove registry variables
- Add uninstall information to the Log
- Initialize variables, get any command line parameters, then perform startup processing with this information
- Update the system config (power/hibernation settings)
- Check disk space before an install

- Create the Beta v4 license
- Install security components
- Install DLLs
- Port over any and all comment lines from the Wise script

Bugged modules (which are labeled as such on the issue tracker) are as follows:

- API Assembly Install - cannot correctly perform a command line uninstall if the API has been installed before
- C++ Redists and Drive Compression check - both use external utility executables which return the wrong result codes to InnoSetup, so checks fail and installation is aborted.

Completed, unintegrated modules are as follows:

- Setup access permissions using utility scutil.exe
- Install Server Files - later parts of this module (the ‘else’ in an ‘if/else’) are required for client
- Install Vista service packs if required
- Install MSI installer 4.5 if not present

Project timelines, task allocation, sprints and milestones can be viewed in our project plan here (or from a link in the Appendix). Please note that task timelines are based from the date the Github issue is assigned to a team member to the date the issue is closed. Therefore, project man-hours cannot be determined from this plan because on any day a team member could have spent 16hrs on a task or 2hrs, or left it to work on another project task or University coursework entirely and simply committed and closed the task on a subsequent day.

Some aspects of the project plan require an explanation. From the perspective of the plan, it looks like Jamie was the only contributor to Sprint 1. However, the goal of Sprint 1 was to prove that the project was indeed achievable with the client’s requested

technology. We took various capabilities of the old Wise script and tried to implement them in the new technology to show proof of concept. All team members were actively communicating, researching, sharing ideas, and offering possible solutions to their challenges and to those of the other members in the team. It just happened that by the end of the exercise, a few of Jamie's tasks were able to be completed and signed off.

Again, in Sprint 3, it looks like Jamie was the only contributor. However, this is due to the project plan being based at the task level rather than the commit level. A vast amount of work was committed in Sprint 3 by Jacob and Ricky for the tasks they were working on. Unfortunately, some last minute bugs prevented them from signing off their tasks before the Sprint 3 release. Hence the reason why the completion of those tasks slid to Sprint 4. To produce a project plan at the commit level would require more work than is practical. For finer grained detail, please view the project's Github commit history.

One issue we have encountered with our project plan is the milestone of an operational 'Client' install. Deep into development it became clear that 80% of the installer's functionality runs regardless of client/server selection, and the remaining 20% is server-only. There are *perhaps* a hundred lines of logic exclusive to the Client Install *at most*.

We believe the nature of this project makes it very difficult to establish milestones. Our other option would have been to simply port the Wise script from start to finish, but this would have introduced its own problems; for example, the 'are we returning to the installer after a reboot?' registry key is set to true hundreds of lines before the point where it is set to false if there is no need for a restart. As an extreme example, being too strict with a sequential approach would have had the installer starting in reduced-functionality 'reboot' mode during testing for weeks until the relevant section of the Wise script was reached.

The project plan shows the project completing within the course timeframe. However, it's an extremely ambitious plan and doesn't allow any room for the underestimation of a task's difficulty, for existing bugs to be difficult to fix or for anything else to go wrong. At this stage, that's a tall ask. For example, Giovanni's script work for the entire project is yet to be integrated into our development branch and tested. We require a lot of luck on our side if that's going to go in without producing bugs.

Architecture

As a port job, the architecture has not changed. We had some issues in weeks 4&5 around how much of the project was going to be a direct-port job from Wise to Pascal, but these concerns were cleared up when Jacob coded up the user interface - the interface is essentially brand-new and, while it imitates the content of each wizard page from Wise, none of the code structure is similar. After this, however, it became clear that the rest of the installer logic was largely portable from Wise to Inno, with some hurdles; for example, the original .wse file is barely-human-readable, but the human-readable pdf version abridges some commands (e.g. commands like "Edit 5 registry keys" will omit *which* registry keys are being edited). Most of these problems we have solved by exploring the .wse, though a few remain, like "If System has Windows 95 Shell Interface" (which we believe to mean 'if system is 32-bit and not 16-bit' but are still unsure, we will communicate with the client about these).

Project Infrastructure

As team members have become less confident that we will complete this project 100%, Giovanni has been assigned to 5 server modules, to be worked on while the rest of us get the Both/Client modules fully integrated and tested. After being assigned these, Giovanni indicated he would like to be assigned tasks again in future rather than use

our established task distribution method of autonomously assigning ourselves work based on our current workload.

The project kanban board/issue tracker is in the appendix. We think our number of issues closed shows good progress - 26 issues are closed, with 13 open and all assigned (some are bugs, not entire sections still to be implemented).

Jacob's personal comments on GitHub usage and commits:

The SPRINT_2 stable release (made available on 31/08) was the result of a 40+ hour week dedicated to the project - I needed time to dedicate to Intelligent Machines and Professionalism assignments, so I did what I consider to be a huge amount of work between 24/8 and 31/8 to free up time for these other projects. As such, I did not expect to have much personal progress to report for the following two weeks, which I was correct about - over those two weeks I only committed the 'Install C++ Redists' module, which is currently bugged.

A mistake I made was discovered in that, because of the nature of the project (integrating code into one large main file), when I made a SPRINT_2 version in a personal branch and then *copied* it into master instead of merging with master (so that I did not have to resolve merge conflicts in the *heavily* altered main .iss file), I then deleted the dev branch (our buffer branch between master and personal branches) and made a new branch off master (called 'dev' again). Of course, this lost all the prior commit history to dev (although there was never any risk of losing code). Ricky was able to get this commit history back by merging his local version of dev, but needless to say I have learned a lesson and this won't happen again. I also deleted personal branches (for example I had one called 'mergeModulesJacob'), with the intent to keep the repository tidy, but of course this lost the extensive commit history to that branch, and now our main file leaps from about 500 lines to about 1890 in a single commit to

master (https://github.com/16430978/installer-capstone/commit/50441ada73d4e9f1073c52fc8d11ce1c848c5e6c). I put this down to inexperience with using git (I had thought commit history for deleted branches would be visible somewhere in analytics), and I will not delete branches from now on.

Jamie

When pushing a completed issue to the repo (the installation of .Net 3.5), I found that one of the files involved in the issue exceeded GitHub's 100Mb file limit. Upon deleting the file and reattempting to push to the repo, I discovered it needed to be cherry picked from every commit it existed in from the commit history. Great! Only now do I remember Aaron warning us about this pitfall in one of his lectures. Many, many hours and experiments later, I was once again able to push to the repo. I did it using only the command line (no GUI), and without having to set new commit ids for any other team member's work. Result!

Ricky

Although this is not the first time I've used git, it's the first time I've used a Kanban board. Jacob's introduction of the Kanban board has been very successful for the project. There is still a problem with my issues in the process status. Although I have finished the client part of the script, it produces the error "Could not call procedure". I'm still confused as to why it is throwing the error, but I will soon have it fixed. In addition, you may see some uncommented commits from me because I will run the installer after I finish writing the script to check for any bugs. Each run will result in some log files and enabler.exe being changed and submitted to Github.

Giovanni

Quality Assurance

Testing

Because of the nature of this project, we do not have traditional regression tests to present. We have stuck to our original plan of testing in VMs, which can then be reverted to a state before the install took place.

Due to the nature of an installer, each new/added process the installer performs is not transparent to an end user. A tester who has Innosetup installed and compiles the Enabler4Setup.iss file can see that, if they immediately run the install, the grey/green boxes on the left-hand side of the IDE indicate which statements are being hit and which evaluate to false or are otherwise ignored.

Other QA

We will be testing our current stable version of the installer (SPRINT_3_WEEK_8) at ITL's offices on 25/09. This includes platform compatibility testing (with several operating systems, plus the actual Enabler hardware device installed on some systems). There is no testing objective other than simply ensuring every statement and command implemented thus far works (alters the registry, installs a file, executes an executable etc.) as intended.

Since we have broken the Wise script down into sections or 'modules', there is to some extent an in-built unit testing approach. In particular, a module is only integrated if all syntax/compiler errors are solved - if there is a bug at this level, it is left out. In terms of functional testing, it is possible to test the functionality of the module being integrated as it is being integrated, but this only tests this immediate module - any global variables that are altered by this module may affect later modules that have already been tested

and integrated (we could not simply work from top to bottom of the old script as, for example, a functioning UX was necessary early on). Because the script is currently in the vicinity of 2500 lines and cannot be fully tested every single time something is added, it will require final testing once all modules (or all that get finished) are integrated, to ensure there have not been any regressions.

Our testing schedule can be viewed here (or from a link in the Appendix).

Product

Our prototype for week 8 can be found in the folder SPRINT_3/WEEK_8 in the master branch of our Git repository (link in appendix, Shawn has access).

This folder includes the .iss script file, plus the result of this script file - the compiled installer - and an external .exe file that the install runs if the ‘Server’ option is selected.

In Week 4, our prototype demonstrated:

- Knowledge of how to present choices to the user in Wizard format
- How to carry these choices forward as variables
- How to construct a custom Wizard in Pascal
- How to execute an external .exe file during the installation
- How to compress files into the installer, and then install them to C:\ drive when the Client install is run.
- How to do the correct one of these two tasks given the user’s choice

Now in week 8, our installer also demonstrates the functionalities from our completed tasks section earlier:

- Create logging capability
- Create fully functional User Interface/Wizard
- Install .NET 4 and .NET3.5 runtimes
- Finish an install after a reboot
- Get Registry/File system values
- Install User documentation/manuals
- Setup Environment Variables and Registry keys
- Instruct the installer to reboot under certain conditions
- Install Enabler Drivers

- Remove registry variables
- Add uninstall information to the Log
- Initialize variables, get any command line parameters, then perform startup processing with this information
- Update the system config (power/hibernation settings)
- Check disk space before an install
- Create the Beta v4 license
- Install security components
- Install DLLs
- Port over any and all comment lines from the Wise script

To repeat; due to the nature of an installer, the new/added processes the installer performs are not transparent, but running the Enabler4Setup.exe in the SPRINT_3\Output folder and then navigating to Program Files(x86)\Enabler4 should demonstrate that files are now being installed throughout different directory trees. The tester will note the cmd dialog boxes that briefly pop up before and after the install, indicating background processes.

If the tester has Innosetup installed and compiles the Enabler4Setup.iss file and immediately runs the install, the grey/green boxes on the left-hand side of the IDE indicate which statements are being hit and which evaluate to false or are otherwise ignored.

Appendix

GitHub issue tracking and Kanban board (only Shawn and team members can view):
https://github.com/16430978/installer-capstone/projects/1

Client requirements:
https://github.com/16430978/installer-capstone/tree/master/Documentation

Project Plan:

https://docs.google.com/spreadsheets/d/1KWdaBHmW7GEV2k9HiJS_x67ogKfYX0GXtzuZKCozXaU/edit?usp=sharing

Project Test Schedule:

https://docs.google.com/document/d/1r39H88Zlrl06IVrnEkrgGiOCFEF4MypJvpJ3ESDIj1s/edit?usp=sharing