Week 4 Report

Group 1: ITL Enabler Installer

Jacob Copeland (#16430978)

Siqi Zhao (#17140639)

Giovanni Saberon (#00100188)

Jaime Curnow (#03336247)

Table of Contents

*Note: The client has provided us with the installer they developed themselves in Wise Installer script. We will refer to this as the ‘old installer’.

Process

Documenting Requirements

On a basic level, our project entails compressing many files into an installer executable, performing background operations and changes in Windows while the install is performed, and launching external installers like SQLEXPR.exe during the install.

Prioritisation and Scoping

We have had an in-depth discussion with the client about scoping, requirements and prioritisation.
The final installer is expected to contain options for a ‘Client’ install and a ‘Server’ install; an installation on a point-of-sale device, and an installation on a central PC or server respectively. The client has suggested we begin work on the project by getting the ‘Client’ install option working, as this option only installs runtime files, while the ‘Server’ install installs SQL Server Express, proprietary runtimes, an SQL database (which is also auto-configured), a windows device driver and two Windows Services, and start menu shortcuts to the installed programs such as the virtual petrol pump simulator. The **major project midpoint milestone** is to have a full ‘Client’ install working, as at this stage a user or a tester can complete one of the two install types from start to finish.

As the old installer is simply one long sequential Wise script, as seen in the ‘Tasks’ spreadsheet in the Appendix we have endeavoured to separate the project into approx. 30 different ‘modules’, based on major functionalities clearly separated by comment lines in the old script. As seen in the ‘Dependencies’ spreadsheet in the Appendix, we

have made a spreadsheet of variable dependencies among the modules in the old Wise script. This allows us to order the development of the individual modules based on their dependencies. The specific order to complete the Client Install milestone is:

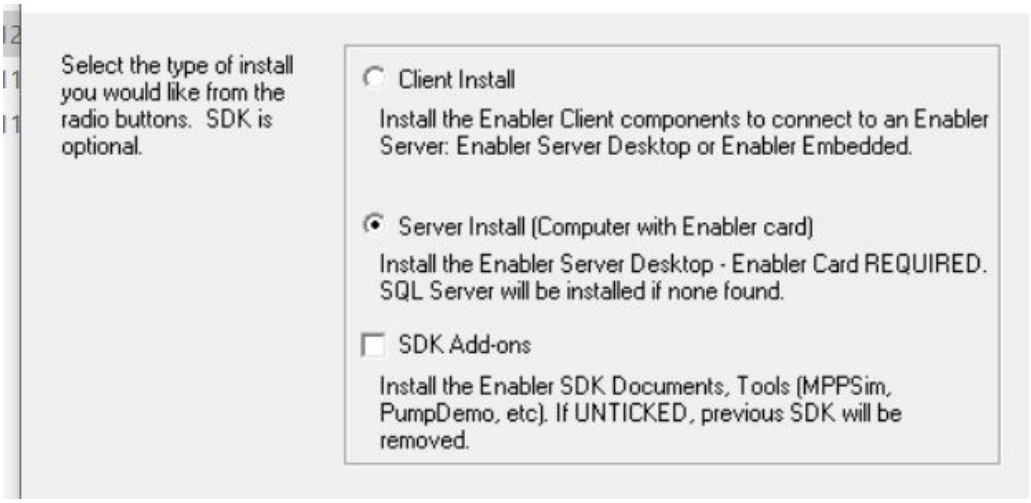
- Almost all modules create entries in Wise's INSTALL.LOG file. We need to ensure Inno is creating an equivalent log first.
- Following this, the modules which set variables, 'Set Installer Variables', 'Set Command Line Options', and 'Startup Processing', are highest priority as almost all modules depend on variables initialized in these modules.
- 'Return from a Restart' is also high priority as some, but fewer, modules depend on this one.
- 'Install third-party DLLs', 'Add notes on installer logs' (which is 100% comment lines), and 'Install security components' are short and have no dependencies, so these are also high priority to get done quickly.
- Almost all other modules are then of equal importance after this, as they only have dependencies on the three 'set variables' modules.
- Finally, the 'Install .NET 4' and 'Setup Access Permissions' come last, as they have several dependencies outside of these first three modules.
- After these modules are completed, this will leave modules which are only used during the Server install process, which are outside the scope of the Client Install milestone. These modules are easily identified in the Wise script by a "If COMPONENTS == 'B'" statement at the beginning of the block.

Our discussion with the client also pointed out elements and background functions of the installer which do not significantly affect functionality and may be considered lower priority if we run into issues with development or time management (though these are not to be considered outside of scope):

- The installer package includes several versions of SQL Server Express; it installs the most recent one in the directory, or none at all if it has been previously

installed. Using only one version for early development of the Server install option will simplify testing.

- As seen in the below screenshot, it is possible to choose not to install the SDK Add-on files with a Client or Server install. To avoid our scope becoming too wide at any one time, we may choose to implement this later.

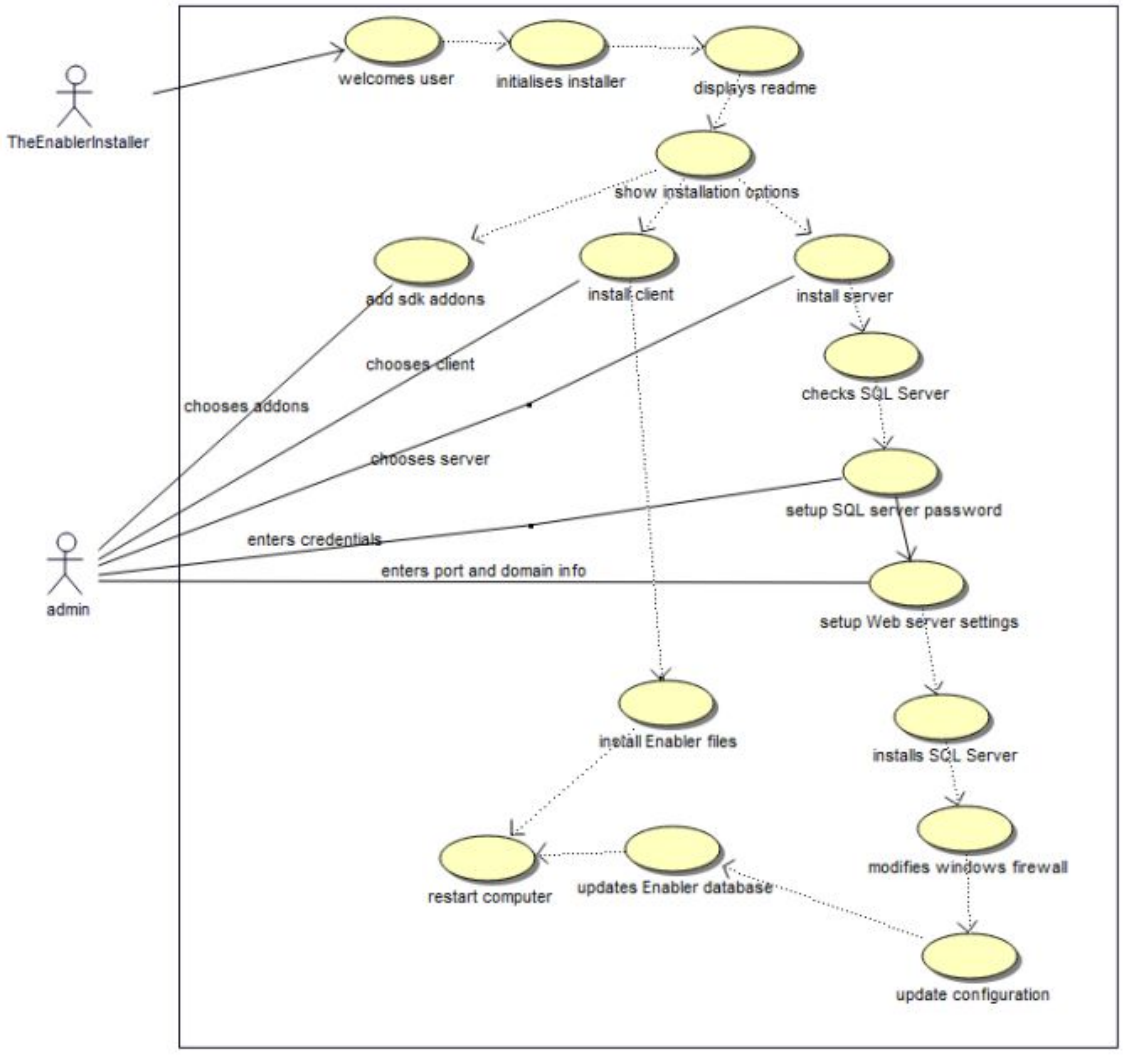


- A prioritisation which will happen naturally is testing only on one version of Windows initially. We have been invited to the client's offices to test on several versions of Windows when we feel it is appropriate, but initial testing will take place on our own PCs/Laptops, which are running Windows 10.
- An operational fresh install from start to finish will be prioritised over the updates to SDK versions and device drivers the installer does if it is run on a computer that already has an installation.
- The installer changes firewall and power settings in the background. These changes are low priority - they will complicate testing on a native OS (though this problem can be solved by testing with restore points on a VM).
- The client has requested this installer can also be run entirely in command line. This is low priority at this stage while we get to know the InnoSetup GUI and scripting.

As this is a porting job from Wise to InnoSetup, there are very few elements which are ‘outside’ the scope of the project - we need to deliver a new installer with all the functionality present in the old installer. The project definition doc says that ITL may change some graphics in the installer at a later date - this we don’t have to do ourselves.

Use Case

The below is a use-case diagram for the old installer - the ultimate goal of the project is for our installer to fulfill the exact same use cases.



(Giovanni Saberon)

Project Infrastructure

We have discussed with the client their optional request that we use Mercurial as our issue tracking and repository. Giovanni felt strongly about using Git and the client did not feel strongly about the issue so we have approval to use Git and GitHub.
Reasons for using git include:

- Familiarity; Git has been used in our university papers before, so there will be less time spent learning the functions and functionality of a new repo.
- Functionality; GitHub allows the user to create Projects, which can be assigned a Kanban issue tracking board and Milestones which a subset of issues can be assigned to. We will use this to track who is working on what, and bugs can be logged here.

Jamie has put together a spreadsheet, compartmentalizing the different tasks the installer performs. These are separated by line number in the Wise script code. Jacob has used this spreadsheet to create another sheet to track variable dependencies between modules, and prioritize them based on the flow of these dependencies. These spreadsheets have helped us create 32 ‘issues’ on our GitHub repository, which represent the modules and which have priorities attached to them.

As this project is written in proprietary InnoSetup script, ANT, Maven, Gradle etc. integration is not expected to be used. If any add-ons for Inno are discovered and used, they will be explained in future reports.

Life Cycle Model & Project Plan

Scrum is a fitting agile methodology to structure our sprints around - it fosters constant communication between group members, and lets us plan to have a deliverable every 3 weeks (scrum 1 starting beginning of week 2) with 4 scrums total. Integrating a kanban methodology, with an associated GitHub Project and issue tracking, will help us to delegate tasks.

The other university assignment obligations throughout the team are as follows (approx dates):

Professionalism in IT (taken by all four of us):

- 1st Sept
- 4th Oct

User Experience Design (taken by Jacob, Jamie):

- 25th Aug
- 4th Oct

Intelligent Machines (taken by Giovanni, Jacob, Ricky):

- 12th Sept
- 4th Oct

Computer Graphics (taken by Jamie,Ricky):

- 16th August
- 30th August
- 20th Sept
- 4th Oct

Having these dates allows us to plan a project buffer - in particular, as we all have assignments due on the 4th Oct (last day of semester), we should aim to have the project in very final stages by the end of week 11.

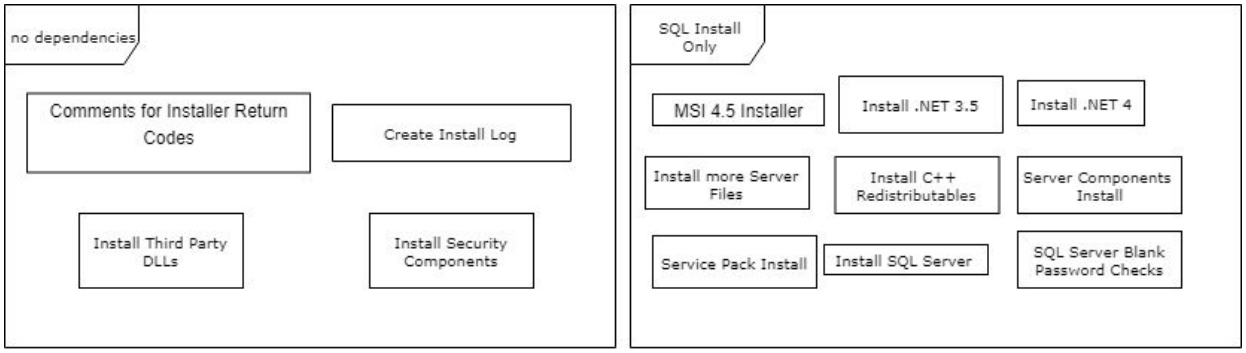
Weeks in which group members have other assignments due will be accounted for by balancing the workload across other group members, and will be noted in weekly reports.

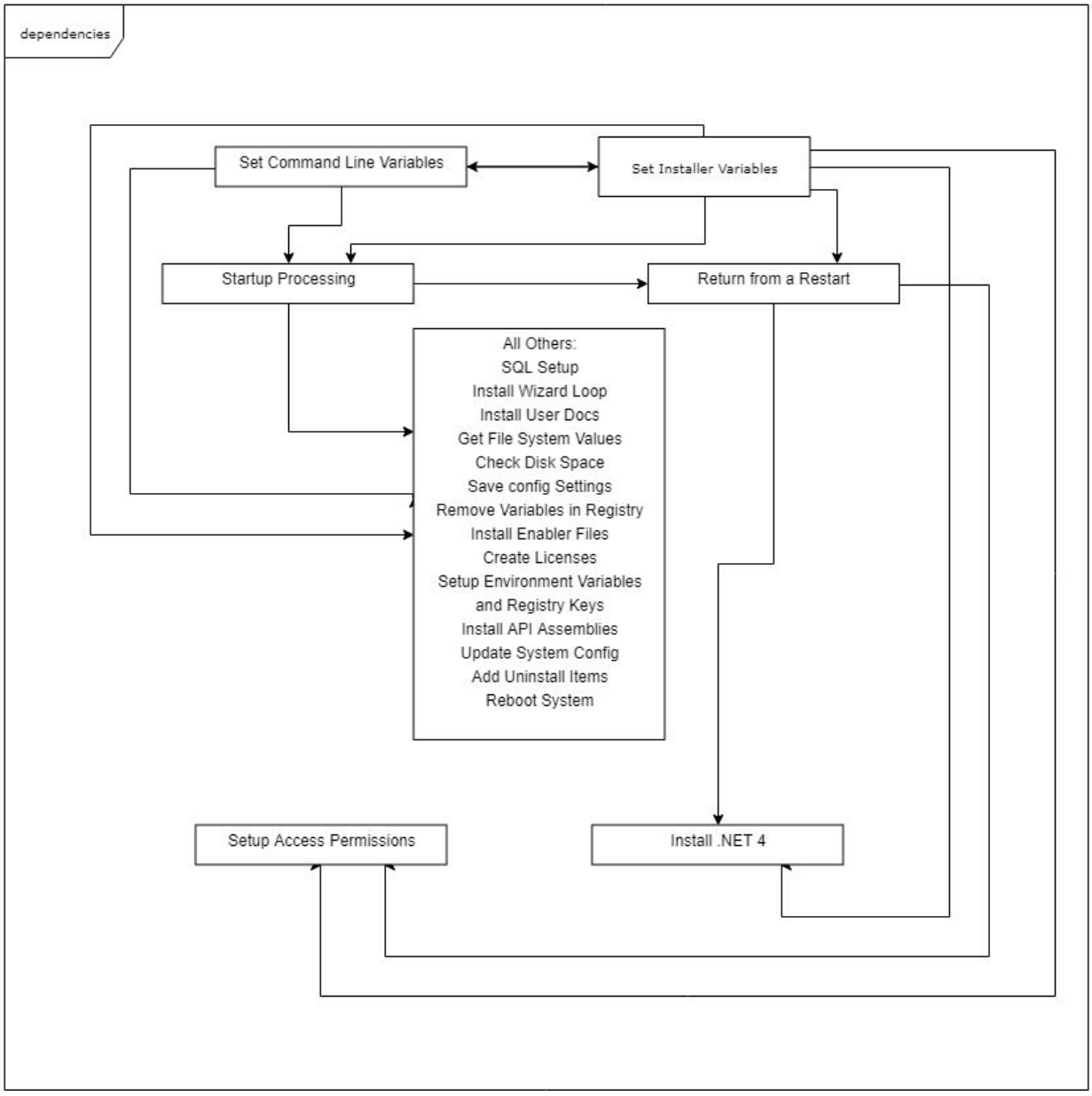
Technology Selection

Our client requires the use of InnoSetup scripting to construct the Enabler Installer, so no selection process is involved. Any extensions or scripting IDEs for Inno Script that we use will be noted if/when they are found. We have done our best to deconstruct the program into a familiar module-based component model, though we admit this is not perfect due to the sequential nature of an installer script.

Architecture & High Level Design Layout

As a sequential installer script, most modules of the installer have dependencies on other modules - for example, almost all modules use the ‘COMPONENTS’ variable, which is set when the user chooses between ‘Client’ and ‘Server’ install options. Dependencies between modules are everywhere, so our prioritisation of requirements has taken this into account and allowed us to order the development of modules. The ‘Client’ version of the install, our major midpoint milestone, should be able to operate from start to finish with implementation of approximately 20/30 of the modules completed, so calling this a ‘midpoint’ may be optimistic.





(Jacob Copeland)

Above is a UML-***style*** diagram to demonstrate dependencies between modules. Each box represents a module, and arrows represent dependencies (boxes at the top have less dependencies, bottom have more). Including the variables themselves would have made this diagram unreadably large, but these can be found in the ‘Dependencies’ spreadsheet when we need to access them.

Note that modules only required for operation of the SQL install have not been put in order of priority yet. We believe leaving these out for now is a sensible decision - noting all variables present in a module is a tedious and time-consuming task, and we think this time is better spent getting started with learning Inno scripting. We expect that, by the time we reach the SQL modules, we will have learned enough about Inno Scripting that we may either not need to do this part of the planning, or will be able to do it more efficiently. This will more than likely be elaborated on in the week 8 report.

Risk Management

Our major risk is running out of time to deliver the project in a completed form! We need to use our time efficiently, and assign tasks as realistically but efficiently as possible.

Our finished program has to make changes to the Firewall, Power settings, and in particular the Windows Registry to work. This poses a major risk to the integrity of our entire operating systems if done incorrectly, so this will be mitigated by testing on a VM.

Quality Assurance

Testing

The client has indicated we can test on multiple operating systems at their offices. There has also been the suggestion that testing could be sped up by running the install on a VirtualBox Windows VM with a restore point created before the install, so settings such as firewall and power don't have to be manually reverted, and the uninstall functionality does not necessarily need to work yet. We are not currently aware of a trustworthy way to automate our testing.

Issue Tracking Policy

Different modules can be worked on separately until they reach the ultimate dependency of needing to all be combined into one install script. Therefore, the clearest workflow is:

Take a task from issue tracker to develop one module -> Develop necessary script for this module -> test that it works -> create new issue for this module to be integrated into the final install script file with other completed modules -> Complete the integration and test that this works.

Modules may be left at the second-to-last stage for a while depending on the flow of dependencies between the module in question and other modules in progress.

Product

Our prototype for week 4 can be found in the folder SPRINT_1/WEEK_4 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.
We believe this prototype demonstrates:

- 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 page 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

To compile the .iss file, the marker will need to install InnoSetup. The installer .exe however should be self-explanatory.

Appendix

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

Project plan spreadsheets:

- Dependencies: https://docs.google.com/spreadsheets/d/1SdETWs8JnK-qYgpxqGSBiMN7VYvaaBcpQ_94nxT9jxw/edit?usp=sharing

- Tasks/issues: https://docs.google.com/spreadsheets/d/19XezZA3l-hzyD3eZUbFKjSlOmy8_6nIBnk8FFIgPWNo/edit?usp=sharing