

Final Report

INNO SETUP ENABLER INSTALLER

Massey University 2019 Group 1

Jacob Copeland (#16430978)

Siqi Zhao (#17140639)

Jaime Curnow (#03336247)

Giovanni Saberon (#00100188)

Introduction

The Inno Setup Enabler Installer is Group 1's effort to port an existing installer, written by Integration Technologies (ITL) in Wise, to the open-source installer scripting platform Inno Setup, in fulfilment of ITL's Inno Setup Windows Install requirements document.

The Inno Setup Enabler Installer encompasses both script ported directly from Wise to Pascal (the language Inno Setup is primarily written in), and specialized script created to perform the same tasks as the Wise installer that could not be directly ported (in particular in the area of user interface).

This report documents the final version of the product as it exists at 15/10/2019.

Contents

INNOSETUP ENABLER INSTALLER	1
Introduction	2
Contents	3
Project Requirements	4
Functionality Requirements	4
Documentation Requirements	5
Testing Requirements	5
Architecture and Design	6
Modular Design	6
Milestones	8
Scoping	8
Workflow	9
Technical Challenges	10
Installer Functionalities	10
Client AND Server operations	10
Server-Only Operations	13
Quality Assurance	16
Off-Site Testing	16
Unit Testing	16
Executable File Testing	17
On-site Testing	19
Future/Possible Testing Methods Given Greater Resources	20
Project Statistics	21
LOC ported from Wise (3201 total):	21

Trends in issue tracker:	21
Requirements and Limitations	23
Known Areas Incomplete	23
Porting Changes	25
Installation Manual	29
Customisation	29
Guide to Building the Product	30
Retrieving the Source Code	30
Downloading an Inno Setup Compiler	30
Building the Product	30
Commercialisation plan	32
Video Tutorial	34
Appendix	35
Acknowledgement	51

Project Requirements

Requirements as defined in the InnoSetup Windows Install document provided by ITL.

The following is a summary of the requirements document, but the essential high-level requirement of this project was to create an installer which functions identically to the existing Wise version.

Functionality Requirements

- Use a Wizard-Style UI
- Allow for Server or Client configurations
- Install Enabler Hardware runtimes
- Can also be run silently from the command line

- For the server installation:
 - Install SQL Express
 - Create Enabler Database and populate
 - Install Windows Device Driver
 - Use ODBCnfg.exe to configure database
 - Install binaries (including those which must be registered to run ITL's proprietary petrol pump maintenance executables)
 - Create start menu shortcuts

Documentation Requirements

- A redraft of the existing 'Installation Instructions' document provided with the installer to end-users, updated to fit the new installer, or at least a set of new Wizard screenshots.
- Indication of any areas not completed.

Testing Requirements

- Install works from Wizard
- Install works from cmd line
- Install works on several versions of Windows
- After installation, the petrol pump maintenance executables work
- Product can be successfully uninstalled from Control Panel

Any requirements not met are explained below in the 'Requirements and Limitations' section.

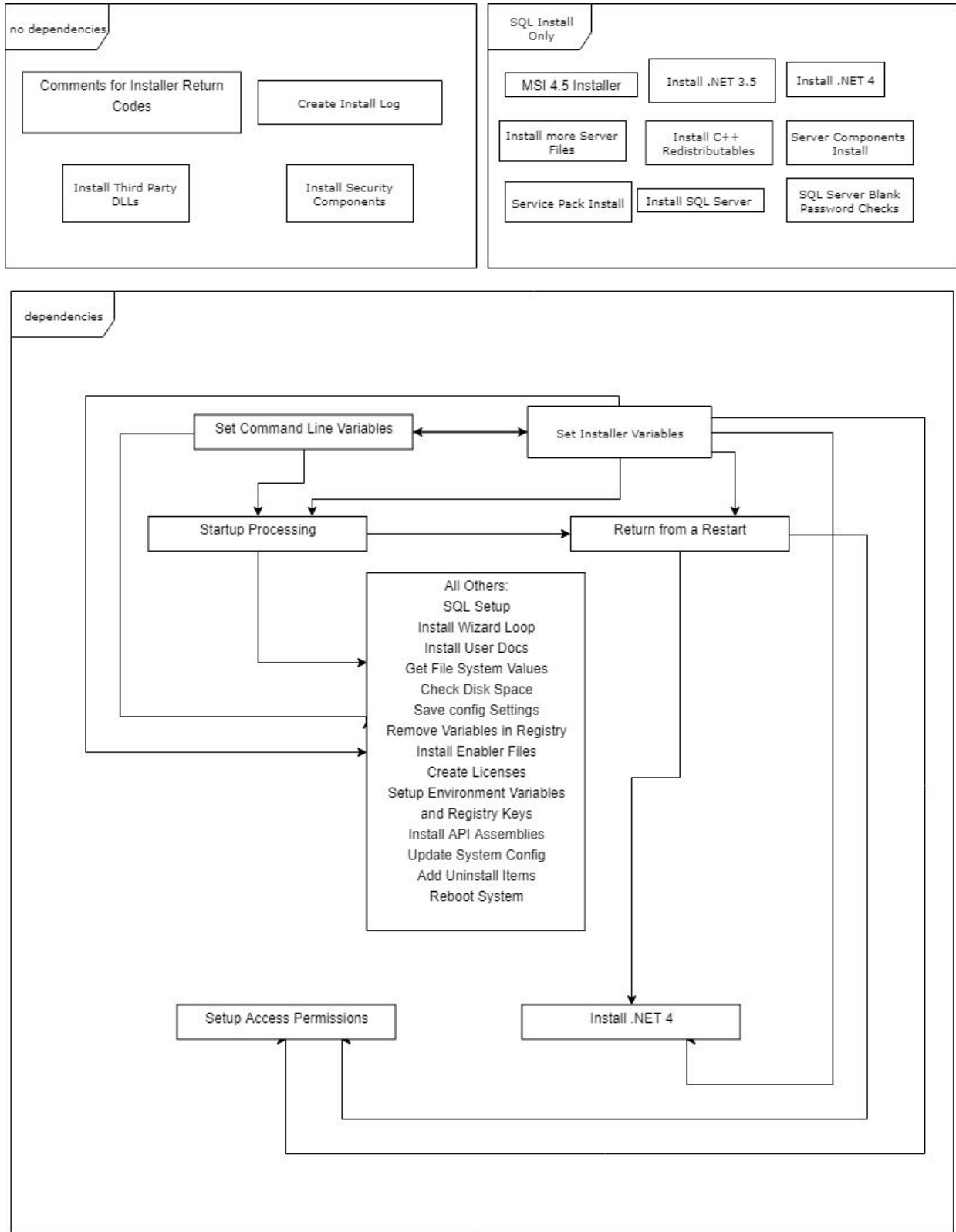
Architecture and Design

Discussion of the planning, workflow, design and product processes

Modular Design

As the old installer is simply one long sequential Wise script, as seen in the 'Tasks' spreadsheet in the Appendix we separated the project into approx. 30 different 'modules', based on major functionalities which were separated by comment lines in the Wise script. As seen in the 'Dependencies' spreadsheet in the Appendix, we made a spreadsheet of variable dependencies among the modules in the old Wise script. This allowed us to order the development of the individual modules based on their dependencies. Most modules of the installer have dependencies on other, previous 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 were everywhere, so our prioritisation of requirements took this into account and allowed us to order the development of modules.

Below is a UML-**style** diagram to demonstrate dependencies between modules in the Wise script. Each box represents a module, and arrows represent dependencies (boxes at the top have less dependencies, bottom have more). Individual variable names themselves can be found in the 'Dependencies' spreadsheet in the appendix.



Milestones

Our initial understanding of the script put the 'Client' install as our half-way milestone, as it does not perform operations to install and populate an SQL Express Server instance, among other tasks. This turned out to be incorrect - it would be more accurate to say that ~80% of the installer's processes run on *both* a Client and Server install, with the remaining ~20% running only during a Server install. There is perhaps 2% of the total code (3201 lines in Wise) which only runs during a Client install.

As such, this was a difficult project to create milestones for - essentially, of our original ~35 modules (more issues and enhancements were added later), it turned out that only 7 of them were server-only. Because of the sequential nature of the installer, to have a stable Client install working, the relevant 80% of the code needed to be complete.

'Stable' builds of this project (sprints 1, 2, 3 and 4) are best seen as versions of the installer that performed incrementally more tasks without crashing or otherwise failing. Outside of the user interface, what these versions performed was not a 'skeleton' of the install that had to be given more detail, like would normally be expected from a prototype; there were instead entire processes missing right up until Week 10 when all the modules were complete. As a port job, **this was to be expected**. There was, essentially, little need for prototyping and stable builds outside of demonstration purposes, because the requirements stayed exactly the same throughout the whole project and there was no need for discussion of changes to our aims or requirements. There were, however, discussions of scoping.

Scoping

As this was a porting job from Wise to InnoSetup, there are very few elements which were 'outside' the scope of the project - our goal from the outset was to deliver an installer with all the functionalities of the Wise installer. Even still, as the varying time commitments and efforts of team members became apparent, there were ongoing

discussions with our project supervisor about scoping, so as to avoid over-promising on this project. A final milestone of only completing the ~80% of the code for the Client install was discussed several times, at length. We put significant time and effort in to avoid having to reduce scope - this ultimately led to an operational clean Server install, which we think demonstrates a dedication to meeting the client's requirements. Work on implementing the Server install continued up until final testing and presentation dates, so we were unable to formally discuss the unforeseen 'Server Upgrade' non-functional aspect of the final product (as discussed in 'Requirements and Limitations') with the client and have it officially ruled out-of-scope.

Workflow

Sections of the script were modularised by turning them all into Pascal procedures (functions). This allowed our workflow to follow a pattern:

1. Script a module
2. Test it by calling it in the CurStepChanged(ssPostInstall, ssInstall) function or InitializeSetup function at the bottom of the script
3. Fix syntax errors
4. If possible, test that all the logic works (this is not always possible until other dependent modules are completed by team members)

This architecture allowed us to simply remove any modules with logic errors that could not be solved (saving the bugged code in a different file). While all modules had to be integrated by the end, this workflow allowed us to move on and continue scripting another section if one caused problems, instead of possibly remaining stuck on a single bug for several days. Jacob makes the comment that one of his bugged sections (Installing C++ Redistributables) was ultimately solved by further knowledge of command line execution statements gained while scripting a later module (SQL Server Install), which demonstrates the benefits of this workflow.

Technical Challenges

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 not directly ported - 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.

As for our biggest technical challenge; 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 and communicating with the client.

Installer Functionalities

Due to the processes of an installer program being opaque to the end user, we will here explain all the major, operational functionalities of the installer.

Client AND Server operations

Files installed (in brief - as seen in lines 37-471):

- Third-Party/First-party DLLs/OCXs (these are also registered)
- Enabler files
- Security Components
- Beta License
- API/Java/Enabler Assemblies
- Utility executables and batch scripts used by the installer
- User Documentation

Operations performed:

- Create Install Wizard pages (lines 3977-4284), including the following:
 - ReadMe page (which can open version notes in IE)
 - Page to select install type (Client/Server) + check/uncheck SDK files
 - CLIENT page to enter existing Enabler servername and whether it is embedded
 - CLIENT page to enter SQL server instancename
 - SERVER page explaining that SQL Express will be installed, OR
 - SERVER page explaining an SQL instance has been found (if exists), OR
 - SERVER page explaining no server exists and SQLEXPRESS files are missing (if true), THEN
 - SERVER password input page (if installing server)
 - SERVER port and domain name entry page
 - BOTH customised ready to install page, with backup checkbox if it's a server install AND a server instance exists
 - SERVER install finished page with options to start MPPSIM.exe, PumpDemo.exe and enbweb.exe.
 - CLIENT install finished page with info.
- Initialize global variables (lines 502-739)
- Declare installer-wide functions (739-992)
 - MoveLogFile() to the same directory as Enabler4Setup.exe if/when setup exits
 - ExitProcess() and Abort() for failstates
 - Checks for install type, OS type, Windows Version, SDK options as used in the Files section
 - Native Innosetup functions which decide if the install requires a restart, and if a wizard page should show or not.
- Initialize variables - read system information to populate variables such as OS and WINDOWS_VERSION (992-1099).

- Parse Command Line Options - if the install has been run from the command line, parse the parameters given by the user to decide install flow. (1100-1252)
- Startup Processing - further system checks to decide the initial state of the install (1253-1403)
- SQL Setup - decide which version of SQLEXPRESS.exe will be installed in case the user selects Server install (1405-1525)
- Return From a Restart - check the registry to see if the installer is returning from a restart and populate saved variables accordingly (1528-1581)
- Check file system - if 8dot3 name creation is disabled, abort install (1601-1619)
- Check disk space - if not enough space, abort install (1622-1655)

After these processes, the wizard first appears. After the user clicks Install, but before any files are installed:

- Save config settings - save user's configuration and choices in the Wizard to the registry in case a reboot is necessary, plus check that the install drive is not compressed (1658-1712)
- If .NET 3.5 is not turned on (>Windows 7) or installed (<=Windows 7), turn on/install it. (1850-1999)

At this point, files declared in the [Files] section (first 500 lines) are installed. After this:

- Check that VS C++ 2008 SP1 Redistributables are installed - if not, install them (2814-2849)
- Remove variables that may have been saved to the registry during the install or previous installs (2852-2876)
- Install Enabler files - Most of this section is in the [Files] section now, but this section also sets some variables to registry values (2880-2944)
- Create beta license - run a utility to convert the beta license (2948-2960)
- Setup Environment Variables - figure out if the version of set.exe to be used is setx32 or setx64, and write more registry values (2964-2992)

- Install User Documentation - Based on the type of install (Client/Server, whether SDK files was ticked), install several hundred user doc and SDK files (3249-3424)
- Set up access permissions - use the subinacl.exe utility to grant access permissions to files and folders required by the installed software, and use rgsrvr.exe to register Enabler objects. (3456-3836)
- Add additional information relevant to the uninstall process to the log (3842-3876)
- Decide reboot - write some final registry values, and if fast startup is not already off, restart the system to disable it. (3881-3973)

The calls to custom install functions are then complete (seen in lines 4297-4409), and, assuming the installer has run without hitting any Abort() cases, the installer will Deinitialize (line 4411).

Server-Only Operations

Additional files installed include more DLLs, Assemblies, batch files, utilities, as well as:

- Database scripts and batch files
- Enabler server executables
- Enabler E firmware
- Web server files/assemblies
- SSL files
- Assemblies for non-English languages
- Web files and images/animations/gifs/icons/pngs
- Javascript files and APSXs
- Pump Simulator (MPPSIM.exe) & PumpDemo.exe end-user executables

Operations performed include all mentioned above, plus:

After the user clicks install, but before files are installed:

- The version of MSI is checked and if it is not ≥ 4.5 , this is installed (1715-1846)
- If the version of SQL Express to be installed is 2014, check if .NET 4 is installed. If not, install it. (2005-2065)
- If running on Windows Vista, check that Service Pack 2 is installed, otherwise abort (2069-2103)
- Double-check that password fields were not blank - this should be impossible to fail, as InnoSetup checks this first (2129-2145)

At this stage files are installed. After this, all previously mentioned processes, plus:

- If an existing SQL instance was found, get a list of instance names (2149-2279) (more on this in Limitations section)
- Install Server Components - if an existing SQL instance was found by the installer and selected by the user, then attempt to establish a connection with the existing instance, detect the version of the instance, and detect if there is an existing install of the Enabler. If there is no existing instance and the install is about to install SQL Express 2005 version specifically, ensure that the MDAC version is ≥ 2.8 and the IE version is ≥ 6 (2285-2475)
- Install SQL Server - Install the latest found version of SQLEXPRESS.exe, then set the OSQL_PATH variable as the path to OSQL.exe. Check that the server can be started/is running. Then install the Windows Drivers for the Enabler hardware. (2479-2810)
- Install Enabler Files - this section was mentioned previously, but for a server install it also attempts to stop proprietary processes (psrvr4 & enabler web) if they have been previously set running.
- Install Server Files - the files now appear in the [Files] section, but this section also performs several other processes, including changing firewall settings and deleting any utilities that may have been installed to be used by the installer but are not needed by the end-user. (3032-3245)

- Update system config - write HostName and Password to registry (3428-3452)
- Set Up Access Permissions - if server install selected, additionally check that psrwr4 and enbweb services are running.
- Decide reboot - additionally in server install, change power settings to turn off standby and hibernate.

Worth noting is that the product can also be successfully uninstalled from the Control Panel.

Quality Assurance

Off-Site Testing

Unit Testing

None of the team had any experience with Wise scripts, Innosetup Scripts, or Pascal. To make matters worse, Innosetup's documentation was often inadequate, and the lack of results in internet searches told us that it didn't have the following of the usual popular languages we were used to coding in.

The IDE was extremely basic. No Junit equivalent plug-in to perform unit testing. No plug-in capability of any sort. No way of testing the script with test values without altering the script itself. No watch window to monitor variable values, and no intellisense to help us with the syntax.

For the reasons mentioned above, almost all work was initially coded in experimental script files separate from the installer script. This way, we could compile it, run it and verify that it performed the tasks we were expecting. The advantages of doing this were:

- It was faster to compile and run because we weren't compiling the entire installer script and running the entire installation every time we wanted to test the smallest change to the code.
- The code was isolated from the rest of the installer code, so it was easier to debug.
- We could litter the code with message boxes or write variables to a log file to tell us what values they contained at any point during execution.
- We could temporarily set the values in a variable to force the code down conditional branches when it would have been difficult to produce that condition.

Using experimental scripts gave us the freedom to play and figure out what worked without making a mess of the installer.

Once it was working in its experimental script, it was copied to the installer script where it was tested again before being committed.

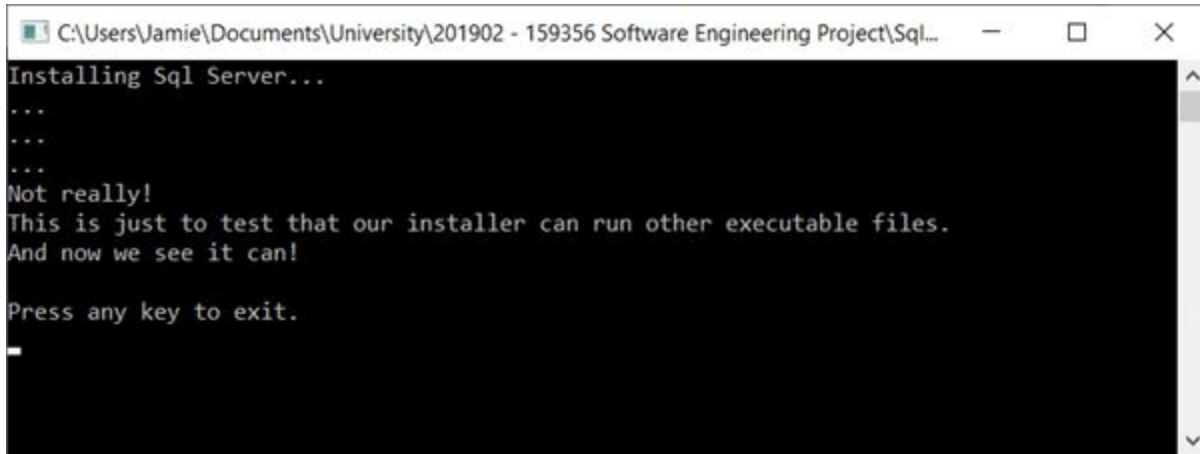
In the end, the experimental script files became manual test harnesses of sorts, except they were testing a copy of a section of script, not the actual installer script. They were comprised of a basic wizard and a 'main' method. The script being tested was put into a function or procedure, and that was simply called from the main method. One drawback was that it could only be run once per variable value test. It usually couldn't be called multiple times with multiple values because whatever it did as a condition of the variable value, had to be put back to keep the tests from influencing each other.

Executable File Testing

The Innosetup script needed to run executable files. However, it was difficult to know exactly what the executable files actually did. We could view the contents of .bat files, but not .exe executables. How do we know they really worked on our particular operating system or setup, apart from just trusting the result code it returned? What can we check if we don't know exactly what it does?

Our personal computers and laptops were our only devices for completing our university studies. If the executables broke something which stopped our computers from working, we'd be in trouble. The same applied when the Innosetup script changed our computer's registry settings, etc. One change of the wrong setting could spell disaster.

Initially, we created mock executable files written in c-sharp that the scripts ran. The files harmlessly opened a command prompt and printed what it pretended to be doing so we could verify it called it correctly. However, we needed to test the real executables at some point.



```
C:\Users\Jamie\Documents\University\201902 - 159356 Software Engineering Project\Sql...
Installing Sql Server...
...
...
...
Not really!
This is just to test that our installer can run other executable files.
And now we see it can!
Press any key to exit.
_
```

We looked into VMWare, but soon discovered we needed to install onto the virtual machine the legitimate versions of all the operating systems we needed the installer to work on.

Then we tried Sandboxie, which allowed us to test with our computer's current operating system at least. This worked well initially, compiling and running the script safely in a sandbox. Unfortunately, we came across a bug, and when we exhausted all the causes it could be, we risked testing it outside the sandbox and it ran perfectly. After a small amount of time spent unsuccessfully finding out why, we explored another virtual machine option.

Time Freeze took a snapshot of our system when we turned it on, then we could safely test whatever we liked. When we rebooted our computer, it automatically used the snapshot to put everything back exactly how it was. The problem with this was that we could never quickly test anything. If we had other work we were doing, open files, 20 browser tabs of whatever we were researching at the time and still needed, it all had to be closed to turn on Time Freeze. Then we could test. Then we could reboot. Then we could continue with our work. It was very time consuming, laborious and inconvenient. But it had to do.

On-site Testing

The client invited us to test on multiple operating systems at their offices. This included platform compatibility testing (with Windows 10 and Windows 7, plus the actual Enabler hardware device installed on some systems). Despite our best efforts, we did not find a trustworthy way to automate our testing in Inno Setup, so checks were done manually, as can be seen in the testing schedule in the appendix (this document was for week 9 and does not reflect all final tests).

Because of the nature of this project, we do not have traditional regression tests to present. We stuck to our original plan of testing in VMs, which can then be reverted to a state before the install took place. There is no testing objective other than simply ensuring every statement and command works (alters the registry, installs a file, executes an executable etc.) as intended.

It is worth noting that, as the Pascal code section of the install script uses many functions native to Inno Setup, the Pascal script could not be scripted and tested in a different, possibly more intelligent IDE. For example, the `Exec()` and `RegQueryStringValue()` functions are used in the Pascal [Code] section of the install script, but are native to Inno Setup and would not be recognised by a standard Pascal IDE.

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.

In terms of functional testing, it was possible to test the functionality of a module being integrated as it was being integrated, but this only tested that immediate module - any global variables that were altered by this module could affect later modules that had 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 final script is over 4400 lines long and could not be fully tested every single time something was added, it required final testing after all modules were integrated, to ensure there had not been any regressions.

Our testing at ITL was extensive, and is explored further in the Limitations section and in the test schedule in the appendix.

Future/Possible Testing Methods Given Greater Resources

The suggestion was made that we could have had the installer write statements to the log upon **every** success/fail of **every** operation, and automate testing by checking the final contents of the log file with a python script. With more manpower, this is how we would have implemented automated testing.

Project Statistics

Project stats from GitHub, Issue Tracker, Kanban board and other sources

LOC ported from Wise (3201 total):

Jamie	Ricky	Jacob	Giovanni
694	1480	2179	(0 integrated)

*The total here is >3201 as any sections assigned to more than one person were counted in full twice. The final version of the .iss script is 4414 lines long.

Trends in issue tracker:

Total issues: 61

Team member	Issues Opened	Assigned issues (total)	Issues closed
Jacob	53	33	32
Jamie	8	15	15
Ricky	0	6	6
Giovanni	0	6	0

Issue types

Team member	Enhancement/Core	Bug	Unlabeled
Jacob	23	6	2

Jamie	7	4	4
Ricky	6	0	0
Giovanni	6	0	0

Total commits: 160

Commits to development/master branches:

Team member	Commits
Jacob	80
Jamie	31
Ricky	25
Giovanni	0 (only committed to personal branch)

Requirements and Limitations

*These sections fulfill both a requirement of the Final Report **and** a requirement from ITL for a document explaining any incomplete areas of the script.*

Client Install or Upgrade - ~80% of total code

Operational on Windows 7 and Windows 10 (platforms requested to test on by ITL).

Server Install ~100% of total code

Requires Windows 10, not operational on Windows 7

Server Upgrade (run Server install with existing SQL Express install)

Incomplete, crashes installer

Known Areas Incomplete

Known bugs/incomplete areas in Client/Server clean install (no existing SQL server):

UninstallAPIMSI() - this function executes a command line statement to uninstall the Enabler API if the EnablerAPI.msi program has been previously run on the system. This command **does appear to work**, though it returns a failure result code. (line 2108)

installAPIMSI() - this function does the opposite later on in the install - executes a command to install the Enabler API. Again, this program appears in the Control Panel and the install appears to work, but the result code returned is wrong - therefore the Abort() statement is commented out so the entire install does not fail. EnablerAPI.msi can of course be run separately from inside c:\Enabler. (line 2996)

Install does not abort if install cannot ping OSQL.

Port and domain are not added to EnbWeb.config files.

No shortcuts are created during the install - all commands involving creating .lnk shortcuts have been replaced with TODOs.

The two 'If System has Windows 95 Shell Interface' statements (which mean 'if system is 32-bit and not 16-bit') have not been ported, as our installer checks the bit-size of the OS while initializing.

Script does check if MSSQLSERVER/MSSQL\$<servername> is running, but does not abort if it is not.

Non-functional parts of Server Upgrade process (existing SQL server):

The SQL setup module normally runs before the files are installed. However, the section that runs if OSQL.exe exists (if an SQL instance exists) has been split off into a separate function (OSQLPathFound()) which now runs **after** the files are installed, as it contains Exec functions which run files from the C:\Enabler folder (Instances.bat). This cascades into the major failstate as it exists in our final version - firstly, this section would need to run **before** the bulk of the install (as was possible in Wise, but this requires significant reworking in Innosetup to be able to install a file before the proper File Install (called ssInstall) stage). Secondly, when this section fails to get a list of named instances, an Abort() statement will be hit in InstallServerComponents() (the bug does not originate in this section, but becomes obvious here, when an Exec statement cannot make a Trusted Connection to an instance).

We believe we made intelligent attempts to get the list of named instances using the Instances.bat file, but were ultimately unsuccessful by the deadline. We believe our code ports the 'Get temporary filename into INSTANCES' Wise command correctly, and it is the statement which executes Instances.bat which is bugged.

Skeleton code has been commented out at the bottom of the script for wizard pages which display a list of instance names if instances already exist, or ask the user for the instance name if instance names cannot be retrieved from MSDE.

Porting Changes

Changes made during the porting process

Notable differences in the internal processes of the install between Inno Setup and Wise

As icons displayed in the install wizard are inserted at compile time in Innosetup (rather than at runtime in Wise) and must be in .bmp format, the installer is not currently branded with the client's .png file in the branding folder, and changing this image file will not do anything. For how to change images at compile time, see lines 26-27.

Obviously, all files compressed into the installer (i.e. accessed from the Input folder and not the Output) now appear in the [Files] section rather than in their respective positions in the Wise script. Those which should only install given certain conditions have Check functions attached to them.

Also obviously, the Wizard is constructed entirely differently.

OCXs/DLLs are now registered in the [Files] section with a Flag instead of during the Update System Config function.

Programs to be run at the end of install now appear in [Run].

All global variables must now be declared at the start of the [Code] section (since sections of code are now split into functions).

Some variables, like MAINDIR or INST, now use the form ExpandConstant('{app}'), ExpandConstant('{src}'), which point to the same places (the target install directory and directory of the setup exe respectively).

At line 1324, the installer does not check if the Windows\System folder can be written to as elegantly as Wise does.

The branding\config.ini file is accessed significantly differently at line 1369.

DriveCompressed.exe is now passed to 'cmd.exe' on all machines, rather than being passed to 'C:\Windows\Sysnative\cmd.exe' on newer, 64-bit OSes. It aborts the install otherwise.

When checking whether to install .NET 3.5, if windows version is 8 or greater, we can actually just turn this on with Dism. Otherwise, run the executable.

The function inside the EnablerInstall.dll function is no longer used to obtain the correct registry key depending on bit-size of the processor, due to unfamiliarity with DLLs. This is now done with a simple string manipulation.

OSQL_PATH is now inserted into PumpUpdate.ini significantly differently with function GiveOSQLPath().

Parts left out of logUninstallItems() as the uninstall is now done automatically by InnoSetup rather than relying on log entries like Wise.

Differences between the end-user operation of the installer between Inno Setup and Wise

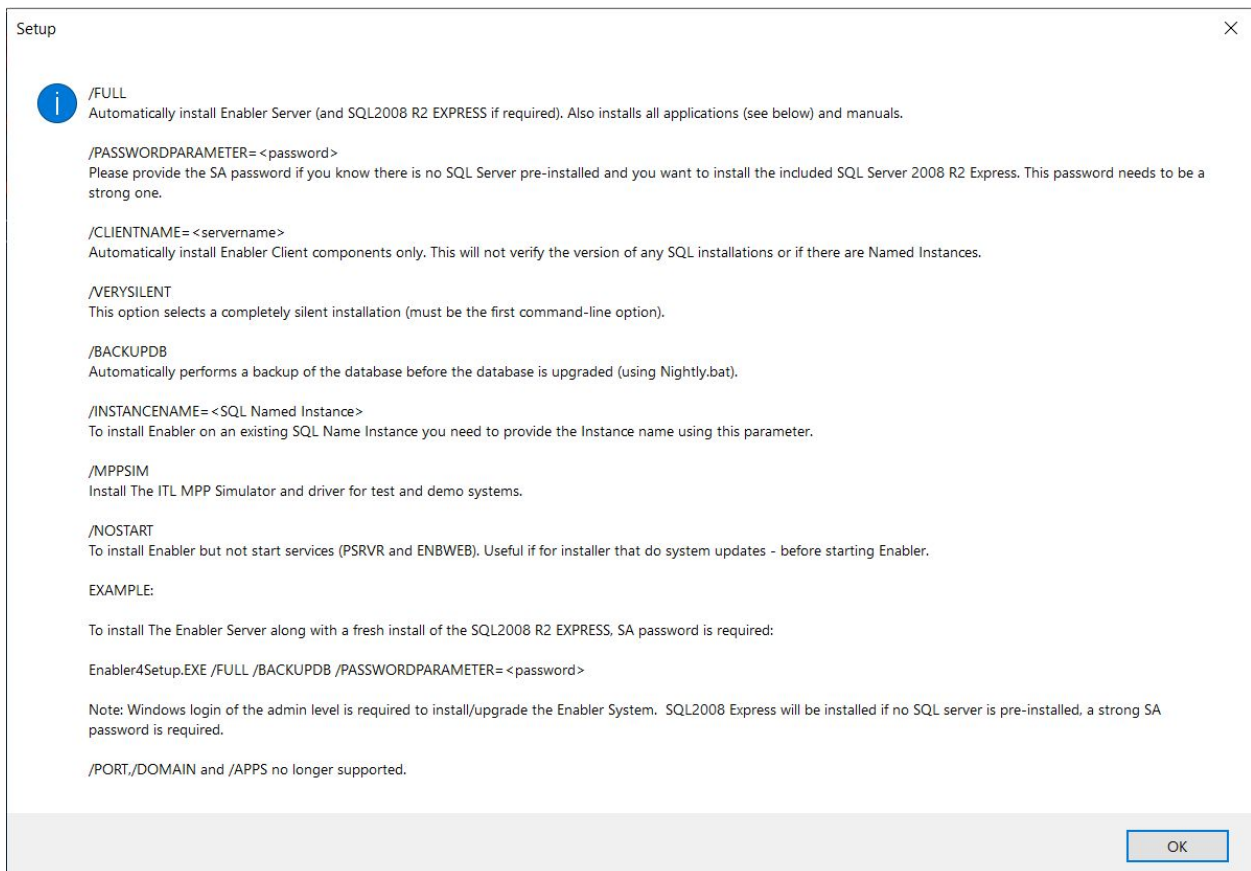
User is now asked for target install directory every time they run the install.

Changes to command line operation:

- Must use -?, /? Or -H to show help dialog (case-sensitive). /H is no longer recognised to show help dialog, to prevent an edge case where /H appears in the command line 'tail' (parameters).
- /CLIENT:<clientname>, /INSTANCE:<instancename>, /PASSWORD:<password> can no longer be used - these have been replaced

with /CLIENTNAME=<clientname>, /INSTANCENAME=<instancename>, /PASSWORDPARAMETER=<password> (case-sensitive). However, /CLIENT can still be used standalone (when not specifying a clientname).

- /PORT and /DOMAIN help text has been removed from the help dialog as these were never actually recognised by the Wise version of the script.
- /S can no longer be used for /SILENT. /SILENT will hide everything except the progress bar, and /VERYSILENT will hide all.



As the installer is not fully operational and presentable to an ITL customer, we have opted to use the time that would have been spent redrafting the client's 'Installation Instructions' pdf file on bugfixing, final testing and documentation, rather than writing an operations document for an incomplete product. We believe that the operational parts of the installer function the same as the Wise version (aside from the above changes to command line operation), and the existing Installation Instructions doc is largely

applicable to the new version. We have however provided a folder of new screenshots of all Wizard pages (in .png format as per the client's request) and rewritten the command line help prompt to reflect changes to command line operation (and included a screenshot of this). The below Installation Manual section serves as a very brief summary of the 37-page Instructions document from ITL.

Installation Manual

Installation for the user

To run the pre-compiled version of the installer, the user needs the Output folder as seen on the GitHub repository. **HOWEVER**, as some necessary files are too big to push to the repository, **the tester will actually need to use the first link on our repository readme** and download the zip folder this link leads to.

The tester can then unzip this folder and run Enabler4Setup.exe.

Alternatively, if the tester also wants the script and input files to compile it themselves after installing Inno Setup, download the second link. We have included the executable to install Inno Setup.

Due to the nature of an installer, the processes the installer performs are not transparent, but running the Enabler4Setup.exe and then navigating to C:\Enabler should demonstrate that files are installed throughout different directory trees. The tester would note dozens of cmd dialog boxes that briefly pop up before and after the install, indicating background processes, but these have been suppressed\hidden in the final version.

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.

Customisation

There is no element of user customisation to this project, outside of selections made during the install process which are explained by the Wizard dialog boxes.

Guide to Building the Product

Obtaining files to compile the Installer Script

The Enabler Installer was written in Pascal and an open source scripting language called Inno Setup. It was then compiled into an executable file using the Inno Setup Compiler. Building the installer from the source code requires installing an Inno Setup compiler, downloading the source code and associated files, and then compiling it into an executable.

Retrieving the Source Code

The source code for this project was hosted on a GitHub repository.

1. Navigate to the repository.
2. Download the second link in the repository's readme.
3. Unzip the zip file to a directory on your computer.

Downloading an Inno Setup Compiler

1. Navigate to the [Inno Setup website](#) in a browser, or use the Inno Setup install executable included with the zip file.
2. Download and install Inno Setup according to the site's instructions. An Inno Setup compiler and IDE will be installed.

Building the Product

1. In the extracted source code folder is a file called Enabler4Setup.iss. This file is the source code for the Installer product.
2. As explained above in the installation instructions, the tester **must have** the missing folders containing files >100MB (SQL2014,SQL2012 etc.); these cannot be obtained from the repository but are in the zip folder the tester should have if they have followed these instructions.

3. Open the file in the Inno Setup IDE.
4. Click the Compile button. It will create the executable file
Output/Enabler4Setup.exe (relative to the location of the iss file).

The Enabler4Setup.exe file is the installer (our product). Running it will install all the necessary software to operate the client's product.

Commercialisation plan

Future development for this project

As described above, the installer has several areas which need fixes at ITL before this program could be distributed to ITL's professional clients. All bugs and enhancements we are aware of that still need to be fixed/implemented to bring our installer to parity with the Wise version have been entered as future development issues on our GitHub issue tracker. These issues are listed here. These issues have not been assigned to team members.

#63 Brand the Install with ITL Logos;

Tags: enhancement, future development

#60 Add port and domain values to enbweb.config files;

Tags: enhancement, future development

#61 Create all .lnks as seen in Wise script;

Tags: enhancement, future development

#59 Implement Abort() if cannot ping SQL instance if OSQL.exe exists;

Tags: bug, future development

#64 Use function inside EnablerInstall.dll;

Tags: bug, future development

#24 Install API Assemblies;

Tags: bug, future development

#62 Implement Abort() if installed SQL instance is not running;

Tags: bug, future development

#58 SQLInstall.bat does not work on Windows 7;

Tags: bug, future development

#56 If SQL instance exists, correctly obtain instance name list with Instances.bat;

Tags: bug, future development

#53 Implement SQL instance list wizard page when instance name list is obtained;

Tags: enhancement, future development

We are unable to make our GitHub repository public, as it contains proprietary files from ITL which should not be accessible to the public. The repository is owned by Jacob and he can be contacted to grant access (or transfer ownership to ITL) at

jacob.copeland.1@uni.massey.ac.nz

Video Tutorial

How to compile installer, and run from both Wizard and Command line:

<https://www.youtube.com/watch?v=u-qUxFQriiU>

Appendix

Project plan spreadsheets:

Dependencies:

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

A	B	C	D	E	F	G	H	I	J
Task Number	Task	Priority	Dependent Variables	Initialed/set Variables					
1	Create an install log.	#1	Presumably none.						
2	Add notes on installer return codes	#2	Presumably none.						
3	Set installer variables	#3+	Initialization of PC_NAME has dependency on COMPUTERNAME, INSTALL_RESULT, SDK_OPTIONS, COMPONENTS, OS, OS_ARCHITECTURE, OS_ARCHITECTURE32, OPERATING_SYSTEM, FILE_SYSTEM, DRIVERCODE, SQLVER_MAJOR, A						
4	Set command line options	#3+	%CMDLINE%, SA_PASSWORD, SQL_INSTANCE	CMDLINE_LOG, CMDLINEUPPER, CMDOPTION, CMDUPPER, CMDOPTION, CMDLINE, CMDSTART, CMDEND, SHOW_USAGE, UNATTENDED, SILENT, W					
5	Startup processing	#5	INSTALL_RESULT, SILENT, UNATTENDED, COMPONENTS	IEXPLORE_PATH, IEXPLORE_VERSION, SHOW_IE_WARNING, READMEFILE, HTML_RELEASE_NOTES, SYSINST_DRIVE, CMD_PATH, USERNAME, IS,					
6	Sql Setup	#9+							
	~Get available SQL server for CLIENT install		INSTALL_RESULT, SILENT, SQLEXPRESSNAME, INST_DRIVE, UNATTENDED						
7	Return from a restart	#6	COMPONENTS, SQL_NEEDED, SILENT	RESTART_LAST, COMPONENTS, PHASE2, PRE_UPGRADE_BACKUP, SA_PASSWORD, UNATTENDED, SQL_INSTANCE					
8	Install Wizard loop	#9+	COMPONENTS, SQL_NEEDED, INSTANCE_NAME_NEEDED, INSTANCE_NAME_LIST, SQLEXPRESSNAME, CLIENTEMBEDDED,						
9	Get file system value	#9+	UNATTENDED	FILE_SYSTEM					
10	Check disk space	#9+	SILENT	DISK_SPACE					
11	Save config settings	#9+	COMPONENTS, OPERATING_SYSTEM, INSTALL_RESULT, SILENT						
12	MSI 4.5 installer	SQL	COMPONENTS==B						
13	Install .Net 3.5	SQL	SILENT, OPERATING_SYSTEM, IS_WINDOWS_SERVER, INSTALL_RESULT	NET2_0_INSTALLED, NET3_0_INSTALLED, NET3_5_INSTALLED, NET4_INSTALLED, NET4_6_INSTALLED, DOTNET_VERSION, DOTNET350_SP, DOTNET_6					
14	Install .Net 4	SQL	SQLEXPRESSNAME, NET4_INSTALLED, SILENT, INSTALL_RESULT						
15	Service pack install	SQL	COMPONENTS==B						
16	Server component install	SQL	COMPONENTS==B						
17	Sql Server blank password checks	SQL	COMPONENTS==B						
18	Install sql server	SQL	COMPONENTS==B						
19	Install C++ Redistributables	SQL	SILENT, VC2008RUNTIME_STATUS, INSTALL_RESULT						

Tasks/Issues

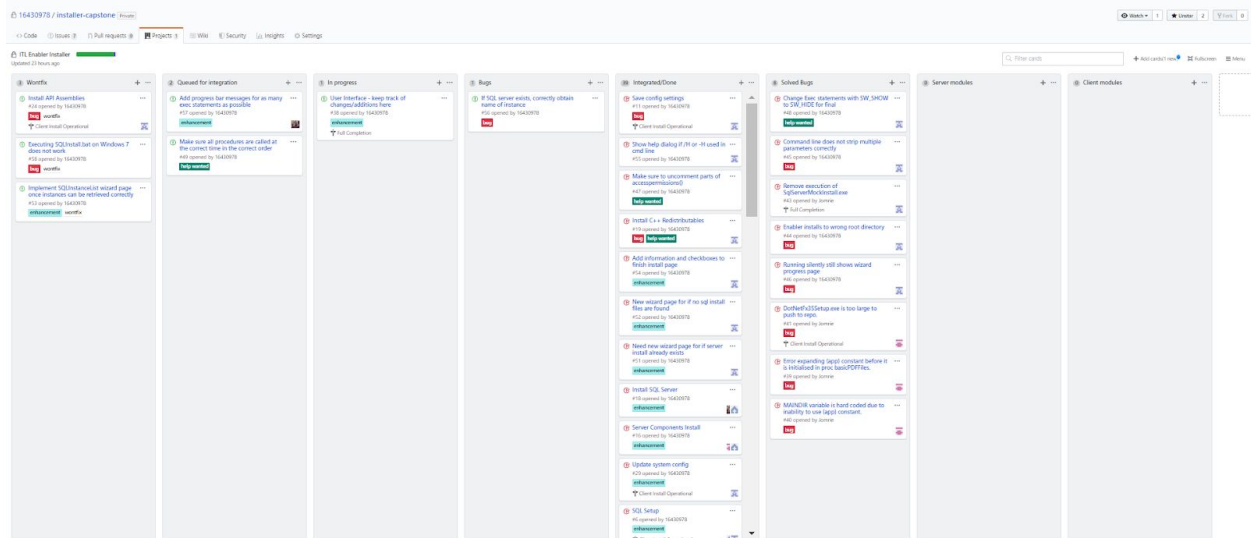
https://docs.google.com/spreadsheets/d/19XezZA3I-hzyD3eZUbFKjSIOMy8_6nIBnk8FFlgPWNo/edit?usp=sharing

Start Line	End Line	Task Number	Task	Comments		
0	0	1	Create an install log.			
1	49	2	Add notes on installer return codes			
50	207	3	Set installer variables	If System doesn't have windows NT running, abort???		
208	348	4	Set command line options			
349	480	5	Startup processing			
481	707	6	Sql Setup			
			->Get available SQL server for CLIENT install			
708	752	7	Return from a restart			
753	827	8	Install Wizard loop	Contains the installer dialogs, Runs once PER DIALOG BOX		
828	842	9	Get file system value			
843	867	10	Check disk space	If 800MB not available, quit install		
868	903	11	Save config settings			
904	1010	12	MSI 4.5 installer			
1011	1115	13	Install .Net 3.5			
1116	1159	14	Install .Net 4	Very low priority		
1160	1192	15	Service pack install.	Only if installing Sql2012.		
1193	1446	16	Server component install.			
			->several sub-modules here			
1447	1463	17	Sql Server blank password checks.			
1464	1763	18	Install Sql Server			
			->several more sub-modules			
1764	1854	19	Install C++ Redistributables			
1855	1882	20	Remove variables in Windows Registry			
			->must be careful here			
1883	1940	21	Install Enabler files			
1941	1951	22	Create licenses.	Might ask client if we need this???		
1951	1985	23	Setup environment variables and registry keys			
1985	2031	24	Install API assemblies			
2032	2048	25	Install security components	Always installed.		
2049	2061	26	Install third party dlls.	For VB6 apps. Looks easy		
2062	2638	27	Install server files.	Has firewall rules		
2639	2776	28	Install user docs.	Client and server installs need this. Looks easy. If SDK selected add more stuff		
2777	2792	29	Update system config.			
2793	3081	30	Setup access permissions.			
3082	3115	31	Add uninstall items to the install log.	Might not have to be done at all in inno		
3116	3201	32	Reboot system			

Jacob Copeland (#16430978), Siqi Zhao (#17140639), Jaime Curnow (#03336247), Giovanni Saberon (#00100188)

GitHub issue tracking and Kanban board (only Shawn and team members can view):

<https://github.com/16430978/installer-capstone/projects/1>



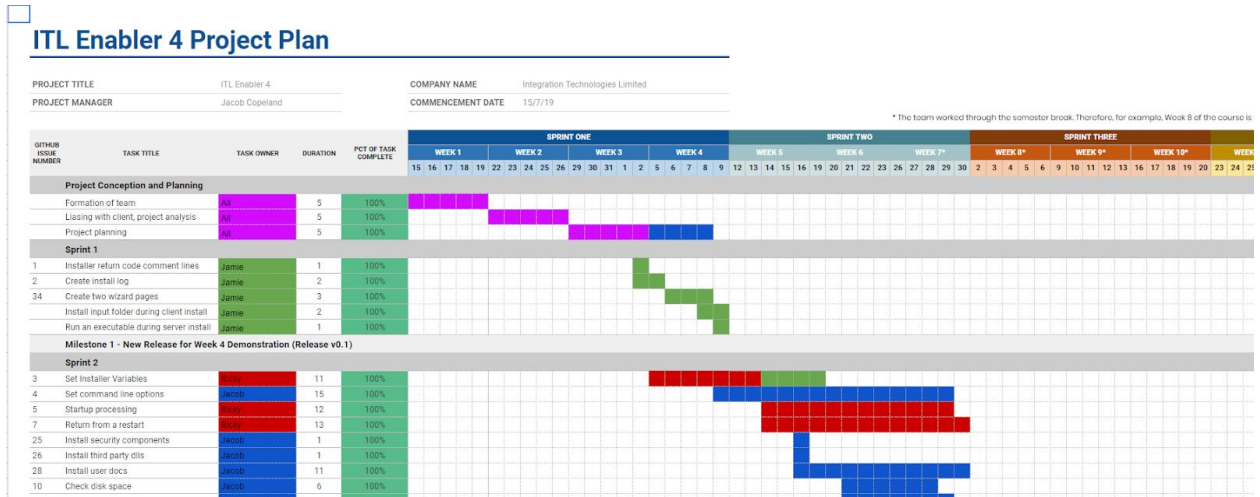
Client requirements:

<https://github.com/16430978/installer-capstone/blob/master/Documentation/Brief%20-%20Enabler%20Installation%20using%20InnoSetup%202019-04-01.pdf>

Jacob Copeland (#16430978), Siqi Zhao (#17140639), Jaime Curnow (#03336247), Giovanni Saberon (#00100188)

Project Plan:

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



Project Test Schedule:

<https://docs.google.com/document/d/1r39H88Zlrl06lVrnEkrgGiOCFEF4MypJvpJ3ESDlj1s/edit?usp=sharing>

Application Testing Checklist (by Jamie)			
Tested By			Date
Application Name	Enabler 4 Installer (Enabler4Setup.exe)		
Procedure	Expected Result	Pass/Fail (P/F)	Actual Results/Comments
Start application wizard.			
Double-click Enabler4Setup.exe.	Welcome wizard page is displayed.		
Cancel application wizard.			
On the Welcome wizard page, click the Cancel button.	A confirmation dialog box opens. Installation is cancelled upon confirmation.		
Perform client install.			

Double-click Enabler4Setup.exe.	Welcome wizard page is displayed.		
On the Welcome wizard page, click the Next button.	Read Me wizard page is displayed.		
On the Read Me wizard page, click View Release Notes button.	The release notes html page is opened in a web browser.		
On the Read Me wizard page, click Back button.	Returns to Welcome wizard page.		
On the Read Me wizard page, click Next button.	Installation type wizard page is displayed.		
On the Installation type wizard page, select Client Install. Tick SDK Add-ons checkbox. Click Next.	The Enabler server name wizard page is displayed.		

On the Enabler server wizard page, enter an Enabler Server name. Click Next.	The SQL server name wizard page is displayed.		
On the SQL server wizard page, enter a SQL server instance name. Click Next.	Ready to install wizard is displayed.		
On the Ready to install wizard page, click Install.	An Installing wizard page is displayed while installation is taking place. When installation is completed, a Setup completed wizard page is displayed.		
On the Setup completed wizard page, click Finish.	Setup completed wizard closes.		
Check that:	SQL Server is installed.		

Check that:	If SQL2014 is installed, check that .Net 4 is installed, otherwise check that .Net 3.5 is installed.		
Check that:	C++ 2008 SP1 Redistributables are installed.		
Check that:	SDK Add-ons are installed (in 'C:\Program Files (x86)\Enabler4\SDK' folder).		
Check:	The log for any errors.		
Disk space test.			
Ensure computer has less than 1480Mb free space. Double-click Enabler4Setup.exe.	Message is displayed telling user the installation failed. Installation quits.		

Non-administrator test.			
Log onto computer as user without administrator privileges. Double-click Enabler4Setup.exe.	Message is displayed telling user they are not an administrator. Installation quits.		
Internet Explorer test.			
Ensure computer Internet Explorer is less than version 5. Double-click Enabler4Setup.exe.	Message is displayed telling user a newer Internet Explorer is required. Installation quits.		
Windows NT test.			
Ensure Windows operating system on computer is older than Windows NT. Double-click Enabler4Setup.exe.	Message is displayed telling user a newer version of Windows is required. Installation quits.		

Command line - showing usage.			
<p>Open a command prompt. Change directory to the location of Enabler4Setup.exe.</p> <p>Enter in the command prompt</p> <p>Enabler4Setup.exe</p> <p>/cmdlineval=/H</p>	<p>Dialog box opens showing usage.</p> <p>Installation quits.</p>		
<p>Open a command prompt. Change directory to the location of Enabler4Setup.exe.</p> <p>Enter in the command prompt</p> <p>Enabler4Setup.exe</p> <p>/cmdlineval=-H</p>	<p>Dialog box opens showing usage.</p> <p>Installation quits.</p>		
<p>Open a command prompt. Change directory to the location of Enabler4Setup.exe.</p> <p>Enter in the command</p>	<p>Dialog box opens showing usage.</p> <p>Installation quits.</p>		

prompt Enabler4Setup.exe /cmdlineval=/?			
Open a command prompt. Change directory to the location of Enabler4Setup.exe. Enter in the command prompt Enabler4Setup.exe /cmdlineval=-?	Dialog box opens showing usage. Installation quits.		
Command line - client install.			
Open a command prompt. Change directory to the location of Enabler4Setup.exe. Enter in the command prompt Enabler4Setup.exe /SILENT /cmdlineval=/CLIENT/ENABLERSDK	Silent mode (/SILENT) means wizard does not run, but installation progress dialogs are still displayed. Client install (/CLIENT) runs. SDK files (/ENABLERSDK) are loaded.		

Check that:	SQL Server is installed.		
Check that:	If SQL2014 is installed, check that .Net 4 is installed, otherwise check that .Net 3.5 is installed.		
Check that:	C++ 2008 SP1 Redistributables are installed.		
Check that:	SDK Add-ons are installed (in 'C:\Program Files (x86)\Enabler4\SDK' folder).		
Check:	The log for any errors.		
Command line - client install with full install.			
Open a command prompt. Change directory to the	Installation quits.		

location of Enabler4Setup.exe. Enter in the command prompt Enabler4Setup.exe /cmdlineval=/CLIENT T/FULL			
Check:	The log for an entry stating "ERROR: Cannot use /CLIENT and /FULL together".		
Command line - client install with password.			
Open a command prompt. Change directory to the location of Enabler4Setup.exe. Enter in the command prompt Enabler4Setup.exe /cmdlineval=/CLIENT T/PASSWORD	Installation quits.		

Check:	The log for an entry stating "ERROR: Cannot use /CLIENT and /PASSWORD together".		
Command line - client install with Enabler Server name and Sql Server name.			
Open a command prompt. Change directory to the location of Enabler4Setup.exe. Enter in the command prompt Enabler4Setup.exe /cmdlineval=/CLIEN T:TestEnablerServer/ INSTANCE:TestSQL Server	Client install runs.		
Check:	Enabler server name is TestEnablerServer.		
Check:	The log for an entry stating		

	"Enabler Server name has been entered: TestEnablerServer" .		
Check:	SQL Server name is TestSQLServer		
Check:	The log for an entry stating "SQL Named Instance - TestSQLServer"		
Command line - client install with apps, simulators, nostart, and backup database options.			
Open a command prompt. Change directory to the location of Enabler4Setup.exe. Enter in the command prompt Enabler4Setup.exe /cmdlineval=/CLIEN T/APPS:/MPPSIM/N OSTART/BACKUP	Client installation runs.		

Check:	The log for an entry stating "The /APPS option was ignored - this option is no longer supported."		
Check:	The simulator executables and dlls are installed, i.e. MPPSim.exe exists in C:\Program Files (x86)\Enabler4.		
Check:	The log for an entry stating "Install Simulator EXE and DLL"		
Check:	Enabler Services have not been started.		
Check:	The log for an entry stating "Not starting Enabler services -		

	NOSTART was selected."		
Check:	The log for an entry stating "INFO: /BACKUPDB option ignored. Cannot backup database on a /CLIENT install."		

Acknowledgement

The team would like to acknowledge the Final Report for 2016 Project Nest, as written by Zoe Purdon-Udy, Francis Greateorex, Sam Hunt and MJ Lee, for providing inspiration for the general structure and layout (though not the content) of this report.