# A GUIDE TO SUCCESSFUL UNIT TESTING

*Tehn Yit Chin*

# A guide to Successful Unit Test

For your next embedded project, here is a guide to getting the right results from your unit tests.

Tehn Yit Chin

This book is for sale at http://leanpub.com/successfulunittest

This version was published on 2017-03-17


This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

# Tweet This Book!

Please help Tehn Yit Chin by spreading the word about this book on Twitter!

The suggested tweet for this book is:

Designing a successful #unittest is really hard. Find out how to do it right.

The suggested hashtag for this book is #unittest.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#unittest

# Contents

# Introduction

## Why am I writing this book

I graduated from the Royal Melbourne Institute of Technology in 1992. Since then I have been working with embedded systems. Most engineering shops practicing this specialised artform are small to medium sized. Their engineering teams are usually small but resource hungry.

My first engineering team was a small team of three engineers. I was a graduate engineer, tasked with writing code for an embedded system. It was for an 8051 derivative from Philips controlling a vacuum replenishing system. At that time, the Australian government just passed a law banning ozone destroying gases, especially from the carâ€™s air conditioner system. So this product would suck out the banned gas from the air conditioner and pump the approved gas into it.

The small engineering team did a back of an envelope design. We coded it up, did some rudimentary testing and shipped it. The project struggled with all sorts of problems. The software was buggy at all different levels. It caused the delivery to be late and the project costs to be over budget. The project finally shipped, but with plenty of bugs and we lost the customer as result. It was a depressing lesson for me.

From that disastrous first project, and the many projects that followed it, I learned to ship a project on time, on budget and on spec. Careful planning must be ensured. A robust testing strategy must be part of that plan. One of the corner stones of the testing strategy is the unit test. With a good unit test plan, the whole project still has the potential to fall down, but that potential is much less.

I usually think of about this as getting all the small bricks that are part of a three storey house correct. The right type of bricks must be specified, and there must be test results that the bricks meet the specifications. Once satisfied with the results, I can safely proceed to build the house. If the bricks are not properly specified or tested, the house has the potential to collapse as the bricks are overstressed and will disintegrate.

In writing this guide, I have hope to close the gap of providing practical ideas to successful unit tests for your code.

## Who should read this book

- **The graduate software development engineer**
  - The graduate engineer will see how a unit test will increase the robustness of code that was created. Firstly it will take the graduate engineer from designing to structuring the code for testability. Secondly, it will show the graduate engineer how to collect

the metrics to prove that the code is correctly implemented. This book will contain the necessary details needed for the graduate engineer to ensure that the software module is complete when it is delivered for integration.

- **The seasoned software engineer**
  - This book will cover topics which a seasoned software engineer should be familiar with, so it should be just a review of a typical process. This book is ordered in a typical process with which the software module is tested and formalises the typical development process for a software module. It will discuss advance concepts such as traceability and test planning.
- **The software team leader**
  - This book will show the software team leader what indicators to measure to ensure that the software modules are written to the necessary quality level. The signs will include the basic measures and items for review. The software team leader will take these metrics to provide a picture of the health of the overall software delivery.
- **The quality assurance engineer**
  - This book will give the quality assurance engineer methods which can be used to monitor quality and derive measurements from the quality of the unit test specifications and the unit test report.

# I am assuming this about you

When writing this book, I am assuming you have been exposed to embedded systems and that you have some idea about the basic concepts of developing software for the embedded system. The concept of writing a device driver for the I2C peripheral means something to you, or at the very least, you are curious enough to find out more information.

You are curious and open to read about ideas and concepts new to you. You are curious enough to take the concepts presented in this book and to further extend them to suit your use case. Your use case will differ from the ones shown in this book. You are willing to experiment and willing to make plenty of mistakes.

You have worked in a team environment where your software modules are expected to be integrated into overall software with other modules. Your software module is also dependent upon other software modules.

# How to use this book

I have presented the information in this book in about the same order which a software is typically tested. However, it does not have to be read in this manner. You could easily have opened the book at any particular chapter to get the necessary info. Each chapter is self contained, let me know if this are some gaps missing for you.

You should make your own annotation so that the concepts are understandable to you. If you have a physical copy of this book, I encourage you to pen down your own notes on the margins of the book. If reading this on a ebook reader, use the notes feature to annotate it.

This book is a guide. It is not a text book that shows everything about unit tests and how it can be applied. It is shows you the basic idea of how a unit test is constructed and how it can be use to deliver robust code.

## Thank you

Finally, before we get into the book proper, I want to thank you for taking the time to learn about unit testing. I hope that you have got what are seeking from this book and can spread the word about the importance of early automated testing.

## Going Forward

If you want to provide feedback and suggestion. You can leave a comment on the website, or alternatively, please send an email to

```
1 tehn.yit.chin@gmail.com
```

# Tools

There are several ways to look at the tools used in unit tests. All the tools are geared towards getting your software code to behave as per the requirements. They can range from self made tools to enterprise level tools.

One of aspects of selecting a tool is the fact that it will have significant effect if it is to be standardise across many projects over a long period of time. This type of technical debt is hard to combat if it is not carefully considered on its long term effect. For example, the implementation of the test case is locked into the unit test framework. Migration is necessary if a different framework is used in the future. This type of technical debt is present in any tool that you choose. It includes the tools used for developing your code. So choose carefully.

The unit test tool forms a framework which the test case can be created and managed. The test engineer would be able to kick the test run, be able to specify which tests to run and have the results available when completed.

## Automatic Stub generator

An automatic stub generator analyses the software module under test. From the analysis, it will get a list of lower level functions that it calls. The signature of the lower level function is also obtained.

The stubs are usually obtained from the header file that is included by software module under test.

Once the stubs are generated, it is up to the test engineer to fill in the details of the stubs. The behaviour of the stub is defined.

The thing to remember is that depending upon the test case, the stub may have a different behaviour. In one test case, the test engineer may want the stub to return a value that is out of the expected range for negative testing. In another test case, the test engineer may want to have the stub return the correct values for positive testing.

## Debugging the software module

During the early stages of the development of the software module, bugs and logical problems will surface. Running the test cases over it will case it to detect a failure.

One method of debugging would be to take the inputs used in the test case and analyse it for the root cause of the bug. If the software module is complex, this debugging strategy is quite intensive. The intensity can lead to many false conclusion. Unless you have plenty of time, the recommendation is to avoid this debugging strategy. However, use it as a last resort if there are no more options.

There are some unit test frameworks around where it mixes languages. In the embedded domain, the software code is usually written in C, C++ or assembly, but the test cases are written in another language such as Python or Java. When a different language is used, it is typical for the C code to be compiled into a shared library. In the Windows world, the shared library would be compiled into a `.dll`. In the Linux world, the shared library would be a compiled into a `.so`. When the test case is written in a different language and it is linked to the shared library, the resultant binary is difficult to debug. The debugging tool will have to cross the language boundary to support all the languages in the binary.

Having said that, I have found that a test case written in Python, linked with a shared library. This combination can be debugged within Windows if the shared library is created using the GNU tools. They will be other combinations as well. However, this adds an additional layer of complication.

To have the ability to debug will allow the software module to be tested faster. It will allow you to get to the root cause of the bug a lot faster.

# Tool Langauge

The language of the unit test framework is extremely important. It is just as important as the language of the software module. In fact, the test scripts that are written is a piece of code that needs to be managed. So ideally, the principles of software engineering also applies to the test scripts as well.

To reduce the complexity in your test scripts, it would be ideal if the test scripts are written in the same language that software module. If you are writing your software module in C, your test scripts should also be written in C. If you are using rust, then rust should be used for the test scripts.

The advantage of using the same language for both the software module and test script, it is compiled for testing on the target. The big assumption is that the language is already supported by your target.

# Metric collecting and logging

Collecting metrics about the unit test is an important aspect of the test strategy. The metric is used to learn whether the software module has correct behaviour or not.

The framework is able to execute each test case and to interpret if its results are a pass or a fail for the software module. As the software module matures over time, the percentage of pass results will get higher and higher. When the software is released, the percentage should be at 100%. However, this will need to take into account of the coverage. The software would not be complete if all the tests are passed, but the software module does not cover all its requirements.

During the capturing of the results for interpretation, it must note down several details about test run. As a minimum, it needs to capture the following details

- date and time of test execution

- the details of the test engineer performing the test
- a snapshot of the environment used for the testing
- the unique name of the software module and its version number.
- the unique name of the test case and its version number.

These details provides a clear picture of the environment of the tests. If the tests are to be recreated, all the details are available.

# Integration

To allow integration of the unit test framework into the build chain, it should be able to be triggered via command line. At the moment, the computer environment most open for tools integration is via the command line. The concept is true if the host is a Windows machine, Linux machine, OSX machine or even one hosted on HP-UX. Command line environment with its large amount of tools, it is the most easiest way of doing this.

The typical setup in a build tool chain is to execute all unit test cases when it is used to do a release. It will compile the the software module, its corresponding test cases are automatically executed afterwards. If the unit tests are not successful, the build is aborted and the engineers notified. Successful testing is to be defined by the project. One possible definition is for all unit test to achieve 80% pass rate as the build is for a prototype. Another definition is for all unit test to achieve 100% pass rate as the build is targeting production.

Once written, the unit tests are available for free in the future. Free means that the unit tests can be reused without any effort. So it make sense to include and execute them as often as possible.

Some unit test frameworks are driven via a graphical interface. This will make it problematic when integrating. A deeper investigation is need on the unit test framework on how it can be integrated. There are several GUI tools available that can be used to automate the execution of the unit tests.

The obvious fall back is to trigger the unit test manually. When you want to execute the unit test cases, the unit test framework would start up manually by you.

For unit test on the target, it would be beneficial if the compiled unit test binary can be downloaded onto the target, executed and the result captured automatically. At the time of writing this book, several of the debugger tool companies are going towards this direction. Their debuggers can be controlled via scripts, making it possible for the automatic execution of the unit test binary.

# Maturity of the framework

There are many unit test frameworks which are designed for a language that is quite new. This usually means that the unit test framework is also quite new. If this is the case, make sure that you pick a unit test framework that is well supported. The support could be in the form of a well

informed and helpful user community. The support could also be purchased from technical experts of the unit test framework.

For support via user community, please do the right thing and contribute your gained knowledge back into the community. They have provided you with valuable help.