Software Engineering:
Software Best Practice
Software Development Tools

EBU5304 Software Engineering
2018/19

Dr Matthew Huntbach
matthew.huntbach@qmul.ac.uk

Software Craftsmanship and Clean Code

- Robert C. Martin (also known as “Uncle Bob”) wrote the book:

  Agile Software Development: Principles, Patterns and Practices 2002 (republished by Pearson Education 2012)

- This book outlines the design principles we discussed previously; it illustrates the design principles and the common design patterns through practical examples of code development
- Another book by Robert C. Martin is

  Clean Code: A Handbook of Agile Software Craftsmanship Pearson Education (Prentice Hall) 2009

- It is about good quality code, stressing the importance of “small” aspects such as good choice of names, appropriate comments, good code layout, as well as “larger” design issues
- The emphasis in both books is on the programmer as a skilled craftsman
- For more on programming as a “craft”, this quote*: “I would describe programming as a craft, which is a kind of art, but not a fine art. Craft means making useful objects with perhaps decorative touches. Fine art means making things purely for their beauty”* is from Richard Stallman

Saying "No"

From a blog by Robert C. Martin:

https://sites.google.com/site/unclebobconsultingllc/blogs-by-robert-martin/saying-no:

- "The difference between a labourer and a professional is that a labourer takes orders from his boss, and a professional provides input to his superiors. Labourers are hired to take direction. Professionals are hired to ensure that the direction chosen makes sense."
- "Programmers are professionals. They know more about designing and implementing software systems than their bosses do. Indeed, they are hired for this knowledge and expertise. And they have a solemn duty to prevent their managers from doing things that would be harmful."
- "All this boils down to one simple thing. Professionals are willing to say 'No'. When their managers come to them with direction that makes no sense, a professional programmer will refuse the direction."
- "Of course saying 'No' is only one side of the coin. Professionals are also expected to explain their positions, and come up with viable alternatives. Professionals *negotiate* with their superiors until both parties are satisfied with the chosen direction."

Learning from Mistakes

From another blog by Robert C. Martin:

https://blog.cleancoder.com/uncle-bob/2013/02/01/The-Humble-Craftsman.html :

• "What does it take to be a craftsman? It takes time. It takes experience. It takes mentoring. And, it takes a *lot* of trial and error. In our industry the best, and possibly the only, way to refine your skill is to make lots and lots of mistakes; and to learn from others who have made lots and lots of mistakes. So *thank goodness* for those mistakes, and thank goodness for the people who made them. Without them, we'd have learned nothing. And *especially* thank the *people* who were willing to expose their mistakes to the world."

From his book *Clean Code*:

• "There are two parts to learning craftsmanship: knowledge and work. You must gain the knowledge of the principles, patterns, and heuristics that a craftsman knows, and you must also grind that knowledge into your fingers, eyes, and gut by working hard and practising".

Microsoft's Best Practices

The following is a list of "best practices" for software development projects which was put together by Microsoft:

- Revision Control System
- Daily Builds
- Build Verification Tests
- Bug Database
- Code Reviews
- Coding and Engineering Guidelines
- Code Analysis Tools
- Globalization and Localization

This is about adopting a disciplined approach to managing the code files and information of a large scale project. There will often be dedicated Computer Aided Software Engineering (CASE) tools for managing these practice

This lecture is based on Microsoft's description of these best practices, with some additional comments noting further developments which have been promoted by the Extreme Programming movement

Revision Control System

A Revision Control System (also called “Version Control”) keeps a record of all the changes that are made to the software of a project

• It will generally incorporate roll-back features, enabling a return to any previous version of the system

• A check-out is when a developer obtains the current version of a file, to have a local copy to work with

• A check-in (or “commit”) is when a developer enters a modified version of the file into the system to become the current version (“master copy”) of that file for the system

• A conflict is when two different developers are trying to make incompatible changes to the same file

• A merge is an attempt to put together separate changes made by two or more developers

Conflicts could be avoided by allowing only one developer to work on any one file at one time (“locking” the file), but this is not always practical or efficient in large scale systems

Daily Build

- A build is the process of producing a complete working version of the software for the whole system, compiling the source code files and linking them
- For all but very small systems this can become complex, so it is good practice to use dedicated tools such for this, which will work with build files that give instruction for the build of a system
- Having a daily build means having a practice of doing this once a day, usually overnight so the team starts work with the latest version of the software
- A build will generally contain at least some verification tests for correct working, sometimes referred to as “smoke tests”
- A build break is when the build or its tests fail, it should be considered a high priority problem that must be fixed immediately

Continuous Integration

- The Extreme Programming movement recommends a complete build of the system for every check-in of a modified file
- Extreme Programming also recommends test-driven development, so there is a full range of tests for every aspect of the system, and regression testing meaning the build incorporates all tests
- This practice of builds involving full testing for every modification is called continuous integration
- Developers will have their own copies of the system code for making and testing modifications, but cannot check-in their modifications unless they satisfy the tests in the build
- With continuous integration it is recommended that developers check-in their work at least once a day
- For this work pattern to be possible, automated build tools which incorporate testing are essential
- A ten-minute build (build script that completes all building including testing in ten minutes) is recommended

Version Control and Build tools

- Commonly used version control tools include:
  - Subversion http://subversion.apache.org
  - Git http://git-scm.com
  - Helix Core https://www.perforce.com/products/helix-core
  - Mercurial https://www.mercurial-scm.org
- Commonly used build tools include:
  - Apache Ant http://ant.apache.org
  - Apache Maven http://maven.apache.org
  - GNU Make https://www.gnu.org/software/make
- Note most of these are Open Source Software

Assert statements

- An assert statement in source code is a statement involving an expression which is always meant to evaluate to true, if it does not an exception is thrown
- Assert statements are used for testing software during development, compilers will have an option to ignore them when producing compiled code for release
- Java has had assert statements since version 1.4
- For example, suppose you had a variable of type `int` called `age` which you expected always to be set to a non-negative number, then you could put the following in your program:

```
assert age>=0;
```

- It would alert you to any case where that variable was wrongly set to a negative number, which might not otherwise be spotted until it caused an error in output somewhere else

Unit testing

- Extreme Programming recommends built-in testing incorporated into all elements of the code, this is called unit testing
- JUnit is a commonly used framework (set of classes) which supports the development of automated tests using the idea of assertion, see http://junit.org
- A unit test is meant to test an individual class or method, it should run very quickly in order to enable a “ten-minute build”
- Unit tests may make use of mock objects, which are simple objects that replace the rest of the code that an object of a class would interact with when it is incorporated into the whole system
- The idea is that the extra work involved in developing automated tests is repaid by the way they ensure modifications made to the code cannot have unexpected effects on other parts of the code, because that would be shown up by an assertion test failing in the build

Bug Database

- A bug database is a central record for bugs that have been identified in the system
- The record for a particular bug will describe it, give tests that demonstrate it, give its status (active, resolved or closed), and who has been assigned to correct it
- Tests that demonstrate a bug should say what is needed to get the bug to occur, what the system should do if there was no bug, and what is observed instead because of the bug
- A bug should also be given a severity and priority rating
- An active bug is one which has been reported but not corrected
- A resolved bug is one where a programmer has submitted a correction
- A closed bug is where the person who originally reported the bug agrees that the correction made has removed it

War Team and Bug Triage

- Microsoft recommends the technique of a war team, which is a small team of senior managers who take control of a project when it is near completion
- The war team has the task of ensuring the system is “good enough” for release
- The war team will triage remaining bugs, meaning they analyse them and decide how serious they are
- Some reported bugs may be duplicates of others
- Some reported bugs may be “features” (agreed as acceptable according to the requirements of the system)
- Some reported bugs may be low priority and can be left to be resolved in a future release
- The war team identify responsibility for fixing bugs which it is agreed cannot be allowed to be shipped in the next release

Code reviews and coding guidelines

- A code review is a systematic examination of source code, it may involve a formal meeting with a structured format to an informal agreement for team members to check over each other’s code
- Microsoft recommends code reviews because:
  - “An extra pair of eyes can go a long way”
  - It “disciplines new hires very quickly on coding style”
- Note that the Extreme Programming practice of “pair programming” can be considered a continuous code review
- There are software tools to help conduct and manage code reviews, enabling colleagues to view and comment on code
- Automated code review involves tools which analyse code to check for compliance with predefined sets of coding guideline rules or best practices

Static and dynamic analysis tools

- A static analysis tool is a tool which examines the source code without running it, for example:
  - Checkstyle – a tool to help programmers write Java code that adheres to a standard http://checkstyle.sourceforge.net
  - FindBugs – a tool which uses static analysis to look for bugs in Java code http://findbugs.sourceforge.net
- A dynamic analysis tool is a tool which examines what happens when source code is run
- A profiler is a dynamic analysis tool which analyses the time and space usage of a program, in particular which parts of the program are most frequently used, or which parts are taking up the most memory – these are the parts where any code change to improve efficiency should be made, it is often hard to predict where they are without using such a tool. The VisualVM profiler for Java https://visualvm.github.io is an example

Debuggers

- A debugger is a tool for assisting debugging
- There are a variety of types of debugging tools, but typically a debugger will enable a programmer to run a program and cause it to suspend at times in the execution, it will have features which enable the programmer to investigate the dynamic state of the program execution when it is suspended, looking at the value of variables and possibly changing them
- A debugger may run code step-by-step (suspending after every execution step) or may enable the programmer to set breakpoints at which it suspends
- A simple way of debugging is to put statements at various points in your program which print out the value of variables when they are reached, if the value of a particular variable goes from an expected one to an unexpected one between two print statements, you can removed the other print statements and put more in between these two
- Systematic use of a debugger enables you to do the same thing without having to make changes to the code

9 essential debugging rules

These were developed by Dave Agans:

http://gipsysoft.com/articles/debug-rules

- Understand the system – make sure you understand how any code you are re-using is meant to work
- Make it fail – find an example which consistently shows the bug happening
- Quit thinking and look – run the code using debugging tools to get information that will help you debug
- Divide and conquer – narrow your search until you reach the exact place where the bug occurs
- Change just one thing at a time – change just one thing which you think causes the bug, if the bug still happens, change it back to what it was
- Keep an audit – keep a record of the bug and what you have done to try to find and correct it
- Check the obvious first – for example, is the code you are looking at the same code you are running?
- Ask someone else – a fresh viewpoint can often help
- If you didn’t fix it then it’s not fixed – make sure any changes you made did actually correct the bug

Globalisation and Localisation

- Globalisation is the process of making your software work in different languages and cultures
- Localisation is the actual translation work for individual languages
- The idea is so that your software product can easily be used by different people across the world
- Software needs to be written in a way where aspects to do with display of text and other issues where they may be differences in different countries are separated out so they can easily be modified without any changes having to be made to the rest of the code
- Issues include:
  - Different languages
  - Different scripts
  - Some languages read text right-to-left rather than left-to-right
  - Some languages have complex rules for combining characters
  - Different formats for dates (UK and USA formats are different)
  - Different paper sizes (standard paper size in USA is 279×216mm, standard paper size in most of the rest of the world is 297×210mm, known as "A4")

Documentation Generators

- A documentation generator is a software development tool that generates documentation
- An example is Javadoc, this is distributed with Java, and is a standard format for Java comments together with a tool which processes those comments and produces documentation in an HTML format that can be viewed as web pages
- Consistent use of a code documentation generator like Javadoc means the documentation keeps in track with the code
- The format of the output of Javadoc can be changed by writing a program called a doclet, for example ApiViz produces UML-like class diagrams (http://code.google.com/p/apiviz)
- Doxygen (http://www.doxygen.org) is an open-source documentation generator for C++ and other programming languages

“Eat your own dog food”

- The phrase “eat your own dog food” originated from the idea that if you ran a company that sold food for dogs, but you kept dogs and you did not feed those dogs on the dog food you sold, it suggests you think the dog food you sell is poor quality
- In general it means if you are manufacturing and/or selling a product, but you do not use that product yourself, instead when you need something of that sort you use another product, it suggests you do not think your product is of high quality
- Making use of your own product also helps make sure you understand it, and an easily relate to any comments about it from your customers
- The phrase originated from television advertisements for a brand of dog food, but achieved widespread usage within Microsoft, and from there came to be used in general for software products, after an internal campaign within Microsoft to encourage the use of Windows and other Microsoft products when developing Microsoft software

Interactive Development Environments

- Software tools are often put together into one system which is intended to be used for all aspects of software development, these systems are called Interactive Development Environments (IDEs)
- Examples include:
  - ➢ Eclipse – written in Java but with plug-ins enabling it to be used to develop software in a variety of programming languages. It is Open Source and governed by the Eclipse Foundation, a consortium of software industry vendors
  - ➢ Netbeans – written in Java and oriented mainly towards developing Java code. It was governed by Sun Microsystems, the company which developed Java. In 2010 Sun, and hence Netbeans, was acquired by the Oracle Corporation
  - ➢ Visual Studio – developed and governed by Microsoft, supports a variety of programming languages. Visual Studio, like other Microsoft products, is not Open Source

Command line operation v IDE

- Command line operation means manipulating your files using instructions typed directly into a console window which interact directly with the operating system
- Before IDEs came into common use, software development involved extensive interaction through command line operations
- Some advanced programmers prefer command line operation because it gives closer control over the computer
- IDEs are controlled mainly by pointing and clicking rather than typing instructions, for many programmers this is more convenient and automates many of the routine tasks needed for managing your files while developing software
- Automation in command line interfaces can be done by writing scripts (programs whose basic commands are operating system instructions)
- Use of an IDE can mean you are not aware of the various tools it uses underneath, and have less flexibility over use of tools
- It is a good idea to gain some experience in command line operation in order to better understand how your computer works

Summary

- In this section we have given some examples of the vast range of tools that are available to assist in software development
- We have also given some examples of best practices in software development
- Use of good software tools helps the organisation of software development, it means that tasks that would otherwise be time-consuming are done quickly and in a consistent manner
- It is important to understand that the informal approach to tasks such as testing, debugging, documentation, building an executable system, which we use when we first learn to program, leads to complexity which is difficult to manage when we move to the development of large scale software
- So moving to a more disciplined use of tools and standards for operation is similar to moving to a more disciplined approach to code structure, as we considered with the design principles and design patterns – it takes effort to adopt the practice, and we have to learn more to do it properly, but it is essential to manage large scale systems
- Experts make good use of tools, but can also “hand craft” when necessary
- Experts often build their own tools