CompSci 230
Software Design and Construction

Software Quality 2015S1
Key developer practices

Lecture plan

Week 1:	*No class - Anzac Day* What is software quality? Some key developer practices (version control, testing).
Week 2:	Black box testing. White-box testing. Myers' testing principles.
Week 3:	Traditional approach to testing (Waterfall). Agile approach to testing (XP). Famous failures.

Myers Ch. 1, pp. 1-4 ; Ch. 2, pp. 5-8

Learning goals for today

- Develop a working understanding of
  - version control best practices
  - testing (what it means to test, psychology of testing).
- Our goal:
  - I find it easy to test someone's memory. Tests of understanding are more difficult and error-prone. Even so, that's my primary goal: to guide (and then to evaluate) your understanding, not your ability to memorise!
    - I'd say memorisation is not useful -- unless you can apply what you have memorised.
    - You won't be able to apply any of this information, until you do the hard-yards of trying (and failing, and trying again ;-) to apply it!

Recap

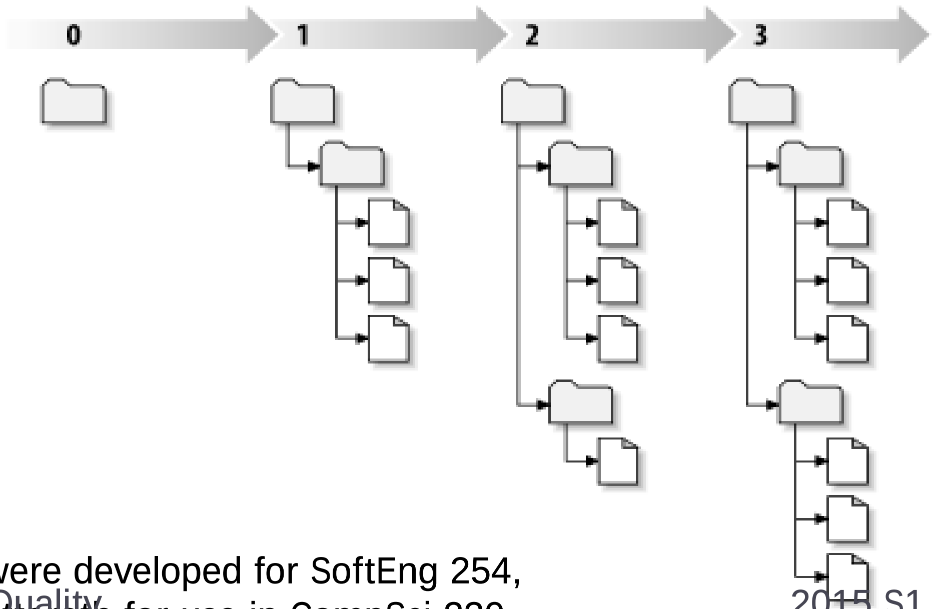
- We understood that software quality
  - has a number of possible characteristics, described in a quality model
  - means different thing to different users
  - for software of any useful size, requires a sound set of practices to be in place
- The developer’s task is to:
  - understand the requirements and which quality characteristics are important
  - ensure any software (s)he delivers (to the user, testers, other developers) is of high quality

Recap

- Two key developer practices that support quality outcomes are:
  - version control
  - testing

Subversion (SVN)

- Centralized open-source version-control system; started in 2000
  - 1986: Concurrent Versions System (CVS)
  - 1977: C-language SCCS released by Bell Labs, in PWB/Unix
  - 1972: SCCS developed at Bell Labs, in SNOBOL, by Marc Rochkind
- A "three dimensional" filesystem: (Revision × Folder × File)
- Each change creates a new revision of the whole file/folder structure
- Revision names are sequential natural numbers (0, 1, 2, …)

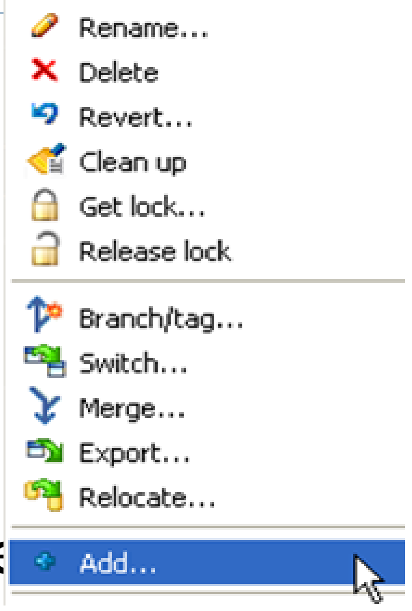


Acknowledgement: The slides on SVN were developed for SoftEng 254, and were provided to me by Christof Lutteroth for use in CompSci 230.

Subversion Features

- Supports merging (recommended) as well as locking
- Changes are transactions
  - Atomic: completely or not at all
    Change is either committed and becomes the latest revision, or is aborted
  - Interrupted commits do not corrupt the repository
- Complete file and folder structure is versioned, including renames and file metadata
- Costs are proportional to change size, not data size
- Delta encoding and merge algorithms work also with binary data
- Works with HTTP server: WebDAV/DeltaV protocol makes it possible to read repository with just a web browser

Add, Delete, Rename, Revert



Add file/folder to the repo

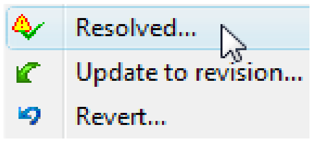
- All new files/folders need to be added explicitly to the repo
- Only add source files (e.g. not `.class` files)

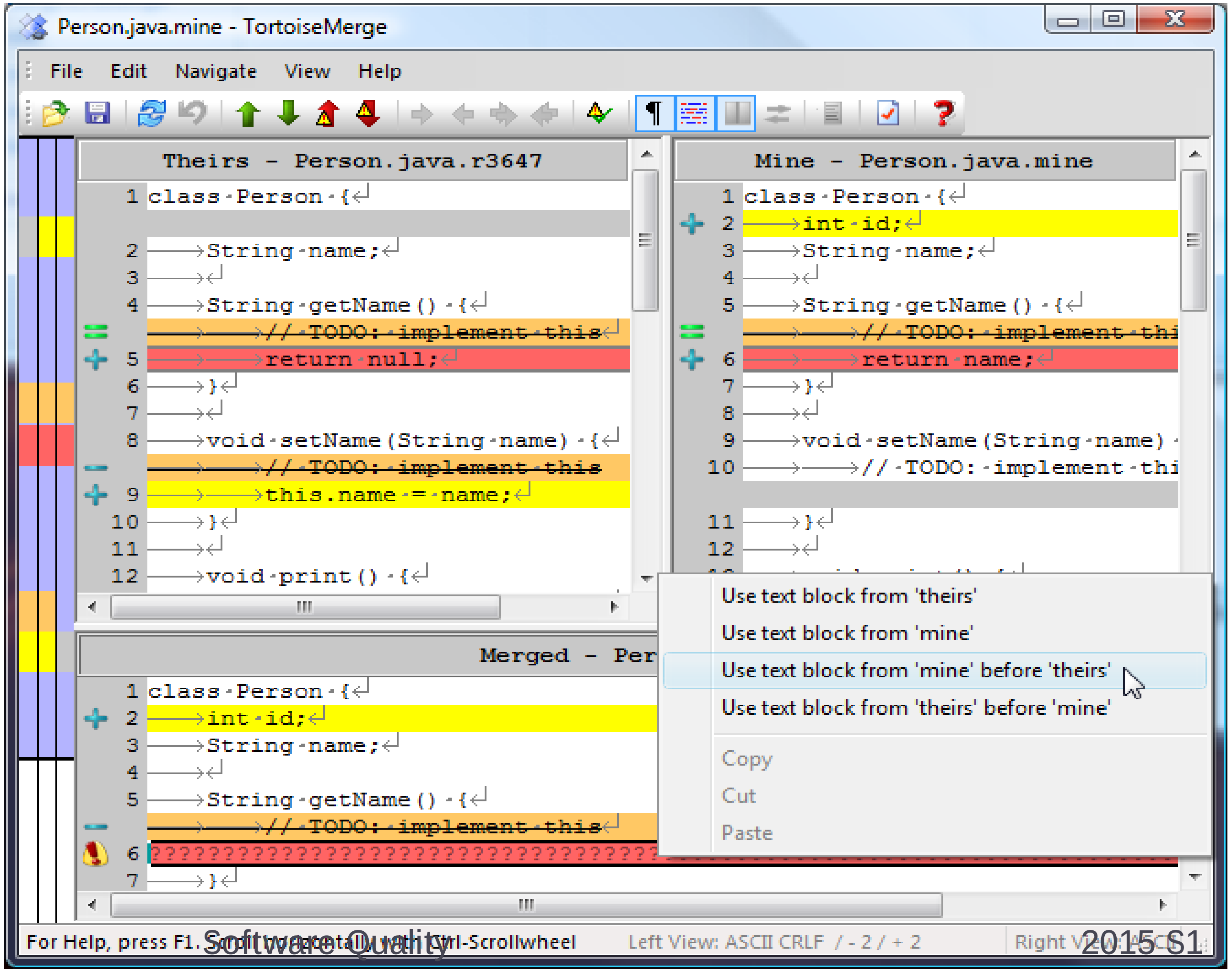
For deleting and renaming files/folder in the repo use the commands (don't delete/rename directly)

Revert local changes in a file/folder if you want to go back to the last version you got from the repo (i.e. throw away local modifications)

Resolving Conflicts

- After updating, SVN may tell you that someone else committed a change which conflicts with your local changes
- Need to decide how to merge the conflicting change
- Use merge tool, e.g.TortoiseMerge
- When a conflict is resolved, you must tell SVN





Person.java.mine - TortoiseMerge
File Edit Navigate View Help
Use text block from 'theirs'
Use text block from 'mine'
Use text block from 'mine' before 'theirs'
Use text block from 'theirs' before 'mine'
Copy
Cut
Paste

Theirs - Person.java.r3647

```
class Person {
	String name;

	String getName() {
~~		// TODO: implement this~~
		return null;
	}

	void setName(String name) {
~~		// TODO: implement this~~
		this.name = name;
	}

	void print() {
```

Mine - Person.java.mine

```
class Person {
	int id;
	String name;

	String getName() {
~~		// TODO: implement thi~~
		return name;
	}

	void setName(String name)
		// TODO: implement thi

	}

```

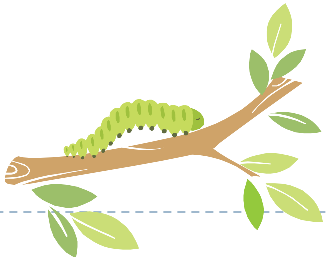
Merged - Per

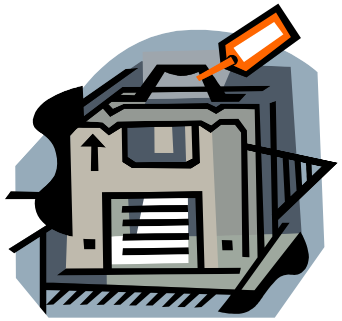
```
class Person {
	int id;
	String name;

	String getName() {
~~		// TODO: implement this~~
????????????????????????????????????
	}
```

For Help, press F1. Scroll horizontally with Ctrl-Scrollwheel   Left View: ASCII CRLF / - 2 / + 2   Right View: ASCII

Branching / Tagging

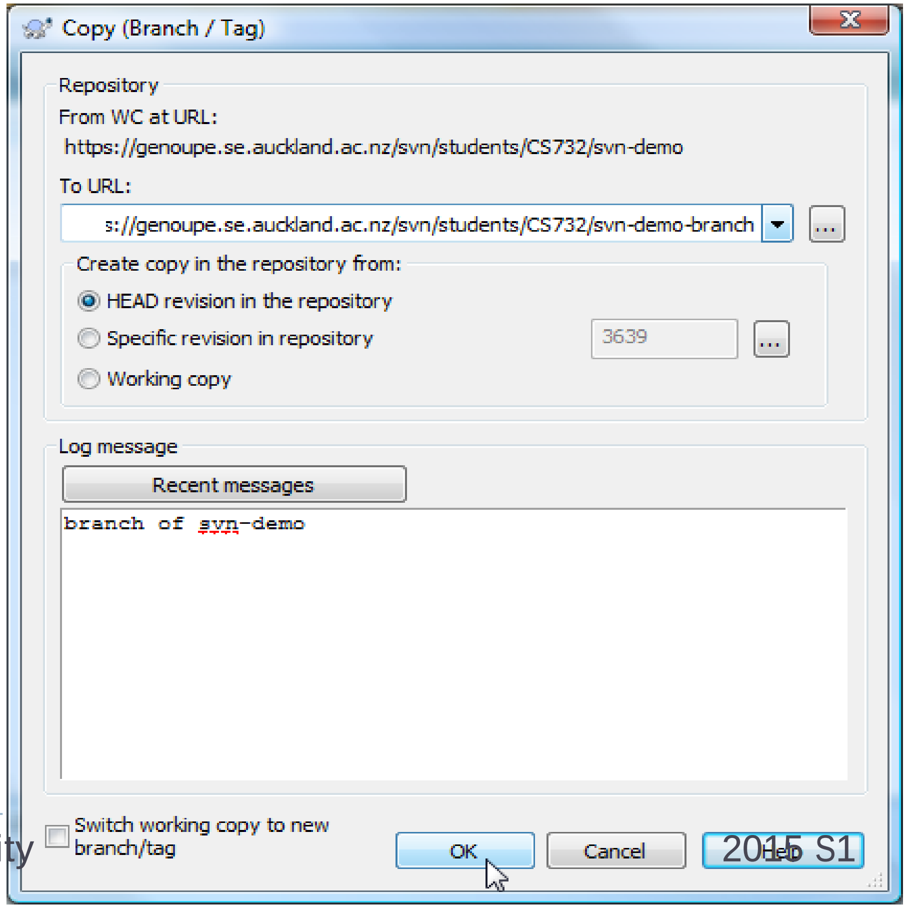




- Creates a copy of a folder in your repository
- Branch: the copy will be used for further development
- Tag: the copy is archival and will remain unchanged

How to do it:

1. Select folder to copy from (right-click on it, use menu)
2. In the dialog: select new folder to copy to
3. Select revision of that folder (usually HEAD)
4. Enter log message
5. Update parent folder of branch or tag to load it in the local working copy



Copy (Branch / Tag)

Repository

From WC at URL:

https://genoupe.se.auckland.ac.nz/svn/students/CS732/svn-demo

To URL:

s://genoupe.se.auckland.ac.nz/svn/students/CS732/svn-demo-branch

Create copy in the repository from:

- (●) HEAD revision in the repository
- ( ) Specific revision in repository 3639
- ( ) Working copy

Log message

```
branch of svn-demo
```

[ ] Switch working copy to new branch/tag

Version Control
Best Practices

1. One Change at a Time

Complete **one change** at a time and commit it

- If you commit several changes together,
  you cannot undo/redo them individually
  - Sometimes individual changes are needed
  - Sometimes individual changes need to be excluded

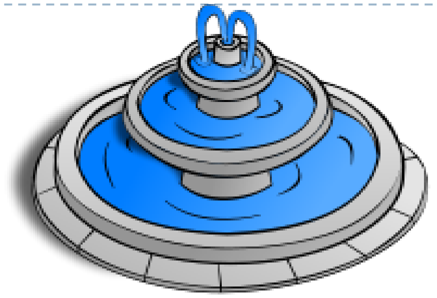
- Continuous integration (see also XP practice)
  - If you make several changes, then conflicts are much more likely
  - Merging a simple change is *much* easier than merging a complicated one

- If you don't commit and your hard disk crashes…
  - Your repository is your backup system
  - Even if the repo is destroyed, other developers will probably have their own local copy. (The Git VCS takes this idea to an extreme: it has no central repo!)

2. Don't Break the Build

- Only commit changes that preserve system integrity:
  - Never commit a "breaking change" (which won't compile, or which fails a test)
- Test-driven development (see also XP practice):
  - Write tests for every change
  - Run tests before committing (at least some of them)
- Think of others:
  - All other developers will download your changes
  - Any problem that was introduced will suddenly be everyone's problem

3. Only the Source



Commit only **source files**

- I.e. files that are actually necessary for your software (including documentation)
- Not generated files (e.g. `.class, .exe`)
- Not temporary files (e.g. irrelevant data or log files)
- Source files are often textual and generated files binary

Why?

- Unnecessary files waste space (other people need to download them when checking out / updating)
- Most binary files are unmergeable (easily lead to conflicts that can't be resolved manually)

4. Use the Log

Write a log entry for each change

- What has been changed and why
  - Twitter style: short is sweet!
  - You are communicating with the other devs
- Logs make it easier for devs to find changes (good and bad ones)

Revision	Time	Author	Description
4	1am	CodeCowboy	Added the files
5	1pm	CodeCowboy	More code
6	2pm	CodeCowboy	Minor change

Revision	Time	Author	Description
4	1am	CodeSheriff	Added files from our old repo at http…
5	1pm	CodeSheriff	Added Order.sort() for sorting OrderItems
6	2pm	CodeSheriff	Bugfix for #67: initialized variable

5. Communicate

Communicate with the other developers

- Before changing existing code
  - See who else is working on it / has worked on it
  - Ask that person about your change before committing (possibly show them a patch)

- Before starting something new
  - Discuss with co-developers and agree on a design
  - Make design proposal, point out design alternatives

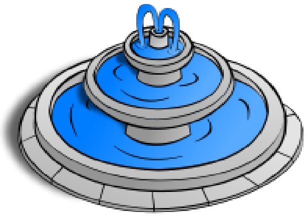
- Always follow the project guidelines & specifications

Version Control Best Practices

1. Complete **one change at a time** and commit it
   - If you committing several changes together you cannot undo/redo them individually
   - If you don't commit and your hard disk crashes…

2. **Don't break the build**
   - Test your changes before committing

3. Commit **only the source** files (e.g. not **`.class`** files)



4. **Use the log** by writing a summary for each commit
   - What has been changed and why

5. **Communicate** with the other developers
   - See who else is working on a part before changing it
   - Discuss and agree on a design
   - Follow the project guidelines & specifications

Software Testing

- The following slides are based on Myers' classic book, *The Art of Software Testing*.
- Many of these slides contain quotations from his book. Some contain questions. Only some of these questions are answered.
  - Myers is a guru. Let's try to understand his insights.
  - If you don't agree with him, that's great! Let's talk about it! If you're not questioning, you're not learning.
  - If you don't understand something, formulate a relevant question and then try to answer your own question. How? By thinking, by reading more from Myers, by reading from another source, by experimentation, …

Psychology of Software Testing

- "One of the primary causes of poor program testing is the fact that most programmers begin with a false definition of the term.
  - "They might say:
    - 'Testing is the process of demonstrating that errors are not present.'
    - (Is this what you would say? Why does Myers think this is false?)
  - "Or
    - 'The purpose of testing is to show that a program performs its intended functions correctly.'
  - "or
    - 'Testing is the process of establishing confidence that a program does what it is supposed to do.'
- "These definitions are upside-down."

Adding Value through Testing

- "Adding value through testing means raising the quality or reliability of the program.
- "Raising the reliability of the program means finding and removing errors.
- "Therefore, don't test a program to show that it works;
  - "Rather, you should start with the assumption that the program contains errors…
  - "and then test the program to find as many of the errors as possible."
- "Thus a more appropriate definition is this:
  - "Testing is the process of executing a program with the intent of finding errors."

Two Definitions from Myers?

- "Software testing is a process,
  - Or a series of processes,
- "designed to make sure computer code
  - Does what it is designed to do and
  - That it does not do anything unintended." (p. 1)
- "Testing is the process of executing a program with the intent of finding errors." (p. 6)
- Is Myers' second definition a restatement of the first one?

Why does this definition matter?

- Myers asserts that "establishing the proper goal has an important psychological effect.
  - "If our goal is to demonstrate that a program has no errors,
    - then we will subconsciously be steered toward this goal;
    - That is, we tend to select test data that have a low probability of causing the program to fail."
  - [but] "… if our goal is to demonstrate that a program has errors,
    - [then] our test data will have a higher probability of finding errors."
- Do you agree with his logic?
  - Do you think his assertion (about the "higher probability of finding errors") could be tested, scientifically?
    - He doesn't offer any experimental evidence, so why should we believe him?
  - The title of his book is "The Art of Software Testing"!
    - Do you think he's defining an artistic style?
    - Does he strongly imply that his style is the only acceptable way to test?

Do you believe Myers?

- Myers supplies no experimental evidence for his claim that a tester using the "wrong" definitions will write test sets with a "lower probability" of causing program failure.
  - This claim has not yet been validated by experiment (AFIK), but it is generally accepted by software engineers.
  - "Testing is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems." [SWEBOK 2004]
- My belief: people who use one of Myers' "wrong" definitions for testing should be working in the sales department!
  - "establishing confidence that a program does what it is supposed to do" = convincing people that they should "trust a program"!
  - If someone trusts a program, their trust **might** be misplaced.
  - Testers can help someone determine whether a program is trustworthy.

Is Testing A Destructive Activity?

- Do you think the job of a programmer is to create "good programs", and the job of the tester is to destroy "bad programs"?
  - Myers thinks "Most people are inclined toward making objects rather than ripping them apart."
- I agree, in part: testing isn't for everybody.
  - But most testers do not think that their task is to destroy stuff.
  - I'd say that good testers tend to be critical thinkers: they enjoy "looking for the bugs", and get great satisfaction from finding them.
    - Some people get satisfaction from fixing bugs, but this is not part of a tester's job.
    - Some developers resolve bugs (as part of the QA Team); some devs add features; some devs review code written by other devs; …
  - In my experience, the best programmers will …
    - cheerfully add new features to a codebase that is throwing a bazillion warning messages during compilation, and has a long list of non-critical bug reports.

What is a Successful Test?

- "A test case that finds a new error can hardly be considered unsuccessful; rather, it has proven to be a valuable investment.
- "An unsuccessful test case is one that causes a program to produce the correct result without finding any errors." [Myers]
  - Warning: this usage of the word "unsuccessful" may confuse your manager!
- SWEBOK 2004 disagrees with Myers, by distinguishing two types of testing:
  - "In testing for defect identification, a successful test is one which causes the system to fail.
  - "This is quite different from [acceptance] testing to demonstrate that the software meets its specifications or other desired properties, in which case testing is successful if no (significant) failures are observed."
- In any event, a test case should only be run if it has a non-zero chance of revealing a defect.
  - Otherwise you are just wasting time with your test.
- By analogy: A doctor who orders an unnecessary test has been "unsuccessful".
  - The doctor's unnecessary test has wasted some resources. Furthermore,
    - Any delay in an accurate diagnosis may endanger the patient by delaying their treatment.
    - If the test (erroneously) reveals an illness, then the misdiagnosis may seriously endanger the patient!
  - Software is always a "patient with an illness" to a tester! (All software has bugs, right?)
  - A doctor who calls for a rarely-used test is very "successful" if this test reveals a treatable disease!
- Should your test report say that software meets its specifications because it has passed 99.9% of your defect-detection tests?
  - Or should your test report describe the bug that was revealed by one of your thousand test cases?

More on the Psychology of Testing

- Myers asserts that "… people perform poorly when they set out on a task that they know to be infeasible or impossible."
  - "Defining program testing as the process of uncovering errors in a program makes it a feasible task."
  - "… the process of demonstrating that errors are not present" is "impossible to achieve for virtually all programs".
    - (Do you agree?)
- If we define testing as "the process of demonstrating that a program does what it is supposed to do", then
  - it can still have serious bugs (e.g. security faults) on inputs that cause it to do something it is not intended to do.
- Myers argues that a tester who is trying to "find errors" is more likely to develop test cases for unintended outputs than is
  - a tester who is considering only "what the program is supposed to do".

A Definition of Software Testing

- "Software testing is a process, or a series of processes, designed to make sure computer code does what it is designed to do and that it does not do anything unintended." [Myers 2004, pp. 1-2]
- Let's examine this carefully. We can perform a grammatical analysis: find the subject and verb (usually these are clauses)…
  - Software testing
  - is a process, or a series of processes, designed to make sure computer code does what it is designed to do and that it does not do anything unintended."
- Hmmm… hardware testing isn't being defined here, nor is any other type of testing. We won't explore this in COMPSCI 230; instead we'll focus on the verb clause or "body" of the definition.

A Definition of Software Testing

- "Software testing is a process,
  - Or a series of processes,
- "designed to make sure computer code
  - Does what it is designed to do and
  - That it does not do anything unintended."

- Hmmm….
  - Testing may be a "series of processes", or just a single process.
    - What sort of process or processes should be used, under what conditions?
  - Two types of test: "does what it is designed to do" and "does not do anything unintended"
    - Are some testing processes better for the first (or second) type of test?

Applying a definition

- "Software testing is a process, or a series of processes,
  - designed to make sure computer code
    - does what it is designed to do and
    - that it does not do anything unintended."
- Let's apply this definition to a simple example (in Myers' book):
  - "The program reads three integer values from an input dialog.
  - "The three values represent the lengths of sides of a triangle.
  - "The program displays a message that states whether the triangle is scalene, isosceles, or equilateral."
- Question: is this enough information to start designing tests?
  - Scientific approach: make a guess, then do an experiment to see if you guessed right!

Active Reading

- If something is difficult to understand, and seems to be written carefully… take the time to
  - Ask questions
  - Analyse what you're reading
  - See if the subsequent text answers your questions
- If you become proficient in "active reading", you'll be able to handle stage-2 and stage-3 papers here. If not, you'll probably struggle…
  - You might ask yourself: why am I taking COMPSCI 230?
  - Analyse your behaviour: are you answering your own questions? If not, you might be wasting your time!
  - See if your subsequent behaviour has answered your questions. If so, you might want to ask some new questions…

Test cases

- Missing information: what sort of "testing process" are we using?
- A partial answer (just before the program description):
  - "We want you to write a set of test cases – specific sets of data – to properly test a relatively simple program."
  - "Create a set of test data for the program – data the program must handle correctly to be considered a successful program."
  - "Here's a description of the program."
- Is this a complete description of a testing process?
  - Hmmm… ? I write test cases; I run the program on the cases; then I determine whether the program handled each case correctly.
    - What if I'm not sure whether a case is handled correctly?
    - How can I tell if a set of test cases has properly tested the program?
    - Keep reading… actively! Ask questions, see if you can answer them…

Tonight: Write some test cases!

- Program description:
  - “The program reads three integer values from an input dialog.
  - “The three values represent the lengths of sides of a triangle.
  - “The program displays a message that states whether the triangle is scalene, isosceles, or equilateral.”
- Recall:
  - A scalene triangle is one where no two sides are equal.
  - An isosceles triangle has two equal sides.
  - An equilateral triangle has three sides of equal length.