Stages in Teaching Software Testing

Tony Cowling

Department of Computer Science
University of Sheffield

Overview

The Problem

- Software testing skills should be built up in stages;
- What are these stages?

The Approach - Compare Two Main Models

- The course sequences in the SE2004 model curriculum;
- A theoretical framework for overall SE skills development, plus a separate maturity model for software testing.

Structure of Presentation

- Background:
  SE2004, the theoretical framework, the Sheffield curriculum;
- Teaching testing in each stage of the theoretical framework:
  programming, software development, software engineering (2);
- Conclusions

Computing Curricula 2001

A Set of Volumes

- One for Computer Science (CS2001):
  - partially revised (CS2008), and a new version being developed;
- One for Software Engineering (SE2004);
- Three for other branches of computing;
- Plus an Overview Report (CC2005).

Common Features

- Each defines a hierarchical body of knowledge:
  - knowledge areas, knowledge units and topics;
- Each defines sets of course structures;
  - typically at three levels,
  - introductory, intermediate and advanced;
- Each discusses other pedagogic issues.

The SE2004 Model (1)

Body of Knowledge

- It imports material from the CS volume:
  - CC2001 knowledge areas become SE2004 knowledge units;
- Two of these units include software testing material:
  - Programming Fundamentals includes testing and debugging,
  - for various kinds of programming constructions:
    - imperative programs, abstract algorithms, recursive functions, etc
  - Programming Languages similarly,
    - object-oriented and functional programs;
- It defines a Software Verification and Validation area;
- This defines some core units:
  - foundations of verification & validation (VAV.fnd),
  - functional testing (VAV.tst),
  - HCI testing and evaluation, eg for usability (VAV.hct),
  - reviews and inspections (VAV.rev), and
  - problem analysis and reporting (VAV.par).

The SE2004 Model (2)

The Basic Structure of Courses

- Introductory course sequences:
  - two versions are defined,
  - a main one and an alternative one;
- Core course sequences:
  - these follow on from the introductory ones,
  - two different packages are defined,
  - which take different approaches;
- Other course, which can cover:
  - other intermediate CS topics,
  - mathematics or other sciences, or
  - non-technical aspects of SE, or other general topics,
  - none of these need be considered further here.

The SE2004 Model (3)

Introductory Sequences and Software Testing

- Main version – four courses:
  - SE101 – programming, testing and debugging,
  - SE102 – requirements, program development, graphical UIs, testing and reviews, basic problem analysis,
  - SE103 – data structures, algorithms, some testing,
  - SE104 – mainly development processes, a bit more testing;
- Alternative version – four courses – more focus on CS:
  - CS101 – programming, testing and debugging,
  - CS102 – object-oriented concepts, basic HCI, basic VAV,
  - CS103 – data structures, algorithms, some testing,
  - SE201 – requirements analysis, use cases, testing against requirements.
The SE2004 Model (4)

Core Sequences and Software Testing
- Package 1 – bottom-up approach – includes:
  - SE211 – software construction – no testing,
  - SE321 – software quality assurance and testing,
  - 3 courses that do not cover testing (SE311, SE322, SE323);
- Package 2 – top-down approach – includes:
  - SE13 – architecture and design – no testing,
  - SE212 – HCI (interface design & evaluation),
  - SE221 – software testing, particularly from requirements,
  - 3 courses that do not cover testing (SE312, SE313, SE324).

The Theoretical Framework (1)

The Structure of the Framework
- This focuses on skills rather than knowledge;
- It identifies four stages.
- Stage 0 – Introductory Programming
  - This provides a necessary underpinning.
- Stage 1 – Software Development (SD)
  - This covers basic methods for producing software systems:
    - restricted to producing feasible solutions
    - for sets of functional requirements
  - It largely ignores non-functional properties, such as:
    - software quality (except for basic usability);
    - process quality, etc.

The Theoretical Framework (2)

Stage 2 – Software Engineering (SE)
- This extends coverage to full range of engineering issues – e.g:
  - requirements for quality, uncertain requirements & requirements conflict;
  - designing for quality, managing risks;
  - selecting and optimising processes;
- It introduces quantitative approaches.
- Stage 3 – Professional Development
  - This stage must scale up to nearer industrial size systems
    - difficult to fit into undergraduate programmes,
    - but important for postgraduate ones.

The Sheffield Curriculum

Stage 2 – Software Engineering (SE)
- The previous stages only cover small systems:
  - typically up to 2 or 3 subsystems, 5 to 7 business classes;
- This stage must scale up to nearer industrial size systems
  - by introducing quantitative approaches.

Software Testing Maturity Model

Background to The Model
- Developed by the software testing research community;
- Based on the different purposes for testing;
- Also reflects the historical development of testing;
- Defines four maturity levels.

Model Structure
- Level 0: testing as an integral part of debugging;
- Level 1: testing to demonstrate correct operation of a system,
  - so failed tests are probably followed by debugging;
- Level 2: testing to find faults in a system,
  - so successful tests are probably followed by debugging;
- Level 3: testing to reduce the risks of operating a system,
  - by identifying situations of correct operation or faults.

Testing in the Programming Stage

Basic Testing Skills
- Identifying single test cases:
- Writing and analysing test cases for code units:
  - for unit testing only;
  - debugging programs;
  - so this activity is essentially all at maturity level 0;
- Skills are developed in CS101/2 and SE101/2.

Other Aspects of Testing
- Very basic HCI testing;
- Mainly for graphical UIs in CS102/SE102;
- Testing and system requirements;
  - basic concepts of use case testing, in SE102/201;
- Code reviews and problem analysis as part of testing,
  - Introduced in SE102/202/203
Testing in Software Development (1)

Increasing the Maturity Level
- At least, testing to demonstrate correct operation (level 1);
- Ideally, testing to find faults (level 2);
- Hence, need to consider test sets, not just individual test cases;
- Also need some consideration of integration and system testing, not just unit testing.

Limiting the Scope of Testing
- SD is mainly just concerned with functional correctness – plus very basic usability;
- Other kinds of testing are beyond its scope.

Testing in Software Development (2)

Key Topics for Correctness
- The concepts of test sets, and the methods for creating them:
  - at least one method for functional testing,
  - and one for structural testing;
- Testing larger scale constructions:
  - use case testing,
  - integration testing,
  - basic system testing;
- Management of test sets:
  - basic regression testing and tool use.

These Imply a Progression
- From concepts to methods that use them:
  - from equivalence classes to generating test frames,
  - from structural units to identifying ones not covered;
- From methods in isolation to relating methods:
  - but not actually comparing them.

Testing in Software Development (3)

Key Topics for Usability
- Usefulness and Usability in HCI;
- User testing of an HCI.

Coverage in the SE2004 Courses
- All beyond the programming material is in core courses:
  - SE221 Software Testing;
  - SE231 Software Quality Assurance and Testing;
- None of it is in the introductory courses:
  - not even the last ones, SE200 or SE201.

Testing in Software Development (4)

Compare with the Sheffield Curriculum
- Nearly all of this material is needed in stage 1:
  - for the first year “crossover” project,
  - where teams of students actually build simple systems;
- To cover this the curriculum requires four courses, not three.

Implications for the SE2004 Model
- Its introductory course sequences go beyond programming;
- So they should cover the software development material:
  - as they more-or-less do for some other key topics,
  - in particular software design;
- Currently they do not;
- Doing so would need a longer sequence of courses.

Extending Testing to SE (1)

Key Features
- Issues of quality must be introduced:
  - product quality (what testing is done), and
  - process quality (how testing is managed);
- Hence, the scope of the skills must be extended.

Extending Test Methods
- Needs to cover a variety of methods for functional testing:
  - various approaches to representing partitions and constraints,
    - logical, algebraic, graphical, hierarchical, etc;
- Needs to cover more advanced kinds of structural units:
  - to consider paths and data flow.

Extending Testing to SE (2)

The Quality of Test Sets
- Needs to introduce the relevant concepts:
  - basic selection criteria; test adequacy;
- Needs to introduce their use:
  - to perform basic measurements and comparisons.

Broadening the Scope of Testing
- Needs to introduce the various objectives for testing:
  - usability testing, performance testing, stress testing, etc;
- Needs to link these with the system operation:
  - i.e. moving up a level in the testing maturity model;
- Needs to combine different testing approaches:
  - e.g. functional with structural;
  - to achieve more thorough testing.
Extending Testing to SE (3)

Testing and Other Knowledge Areas
- Testing needs to link with Software Quality:
  - how testing can improve quality;
- Testing needs to link with Software Measurement:
  - so that improvements in quality can be characterised.

A Contrast with SWEBOK
- SWEBOK uses a theoretical order for testing topics:
  - from the most general and fundamental,
  - to the practical applications;
- Teaching the topics needs a different order:
  - start with simple versions of the practical applications,
  - then develop to applying more fundamental theory.

Comparing The Main Models (1)

A Problem with Alignment
- The SE2004 and theoretical models do not align well;
- SE2004 introductory courses cover testing of programs:
  - this matches the programming stage of the framework,
  - the relevant skills are developed in CS101/2 or SE101/2;
- They do not cover much about testing of systems:
  - basic topics are introduced in SE102, SE200 or SE201,
  - little development of the relevant skills;
- So the main boundaries in the two models do not align:
  - is the boundary ‘introductory sequences to core sequences’
    does not match the one ‘programming stage to SD stage’;
- Hence the introductory courses do not support the SD stage:
  - their coverage of testing is inadequate for developing systems,
  - the necessary skills are only developed in the core courses.

Comparing The Main Models (2)

Aligning These Models Better
- A longer introductory sequence would be needed:
  - in order to equip students better for testing software;
- But fewer core courses:
  - because testing material moves to the introductory sequence;
- Learning objectives in the introductory sequence:
  - should include an objective for ‘doing SD’:
    - most appropriately in SE200 or SE201,
    - or the proposed additional course;
- Teaching method:
  - a complete SD project would be a valuable activity,
    - ideally as part of SE200 or SE201,
    - or the proposed additional course.

Comparing The Main Models (3)

Validating These Comparisons
- Would these proposed changes give a better curriculum?
- A controlled experiment would be impossible:
  - this how the Sheffield curriculum is organised;
- But the proposals have been shown to work:
  - the emphasis on project-based courses;
  - involving a series of incremental changes;

Possible Confounding Factors
- The Sheffield curriculum involves other features:
  - notably the emphasis on project-based courses;
- But the theoretical framework abstracts from these:
  - its benefits, and the benefits of project-based courses,
  - are independent of each other.

Conclusions

Basic Results
- Two models have been described and compared;
- Experimental validation of this comparison is probably not feasible;
- The comparison has been validated by action research:
  - during the development of the theoretical framework.

Alignment of the Main Models
- Alignment for knowledge is difficult to characterise:
  - for testing programs it is largely imported from CS2001;
- They do not align well for skills development:
  - particularly comparing the SD / SE boundary with the
    introductory sequence / core sequence boundary;
- Providing a better alignment would not need major changes.

Future Issues
- How software measurement relates to these various models.
The End

Thanks for your attention!!

Any Questions?