Finding errors in programs: testing and debugging
FIT1040 Programming Fundamentals (Week 2)
Lecturer, Campus
Faculty of Information Technology

Learning objectives
By the end of this week students should:
1. Describe the kinds of tests that are used in software development
2. Be able to create test cases for unit testing of basic input screens
3. Be able to formulate and execute test plans for simple software applications
4. Understand the difference between syntax errors, logic and run-time errors in program code
5. Understand how to debug a software application in Scribble

Today’s agenda
Today’s agenda

- Being human and making mistakes
- Why “testing” software is impossible
- What industry does to make sure software works as expected
- Technical reviews
- Testing
- Definitions of different types of software tests

We see and hear what we want to

- Confirmation bias “tendency of people to favor information that confirms their beliefs or hypotheses”
- This basic human attribute makes it hard for us to find errors in work we do
- What can we do?
  - Make assumptions explicit (technical reviews, pair programming)
  - Think about testing before we have an emotional investment in the outcome
    - Design tests before coding
  - Get somebody other than the programmers to do the testing
  - “Testing” is a profession
Why “testing” software is impossible

Some definitions

- Validation
  - Making sure a software artifact does the right thing
- Verification
  - Making sure a software artifact works correctly

- Validation means making sure we have done through analysis – making sure we understand the problem (avoiding type III errors)
- Verification compares performance of an application to the expected performance (do we get the expected output for a given input)

Can we every be sure a system is going to work 100%?

- No, it’s impossible for most moderately complex systems to guarantee they will work exactly as expected. There are just too many potential execution paths to test them all.

- So what do we do?

What industry does to make sure software works as expected
The software development process

- Aimed at making sure software is addressing the right problem, two general approaches:
  1. Spend a lot of time up front doing analysis (traditional waterfall approach to software development)
  2. Build a little bit of a big application at a time, making sure the development is heading in the right direction before committing to the next stage (agile development)

Agile Development


Risk management and software QA

- How much testing would you do for?
  - A spreadsheet you have put together to help you choose an ISP
  - An Access database you are using to maintain the membership list at your tennis club
  - A student records database at a University
  - An on-line learning system at a University
  - A fire-alert system for a state-based fire agency
  - Heart and blood pressure monitor in a hospital ICU
Quality assurance (QA)

- Process of ensuring information system meets minimum quality standards
- Determined by users, implementation staff, management
- Identification of gaps or inconsistencies in system requirements
- QA is integrated into project throughout the software development process
- Cost of fixing errors rise as project progresses

Typical industry responses to QA risks

- Application of development methodology
  - Like the waterfall model, or agile development
- Redundancy
- Layers to reduce complexity
  - Data, business, interface, network, security
  - Standard operating environments (SOA)
- Labor specialization
  - Usually related to hardware layers
- Hardware certification
- Reuse of “known” software components
- “Issue” and version management
  - Separate production from development
  - Hot fixes, patches, maintenance releases, service packs
- Pair programming
- Code reviews and walkthroughs
- Testing

Technical reviews

- Formal or informal reviews of design or construction details by group of developers
- Open design and construction process to input from other people
- Other programmers can frequently see errors missed by original programmer
- Similar to author writing and editor reviewing
- Walkthroughs and inspections
  - Reduce number of errors by factor of 5 to 10
  - Reduce testing costs by 50%
Testing

- Process of examining a product to determine if any defects exist
- Testing levels are related to specific SDLC phases
- Testing activities spread throughout SDLC
- Most of testing takes place following software construction and definition of defect standards

Generic model of software testing

Correspondence between SDLC phases and various types of testing

SDLC phases and testing activities performed within each phase

- **Analysis**
  - Plan system tests

- **High-level (architectural) design**
  - Plan integration tests
  - Develop system tests

- **Low-level (implementation) design**
  - Plan unit tests
  - Develop integration tests

- **Implementation**
  - Implement unit tests
  - Implement integration tests

**Test planning**
- Development order
- Testing order
- Data used to test modules, module groups, methods, classes, programs, and subsystems
- Acceptance criteria
- Relevant personnel assignments (construction and testing)

**Test cases**
- Important part of testing is specifying test cases and test data
- A test case is a formal description of
  - Starting state
  - Events to which software responds
  - Expected response or ending state
- Analysis phase documentation is useful in preparing test cases (use-case driven)
- Test data is defined to be used with a test case

**Definitions of different types of software tests**
Unit testing

- Tests individual modules of code or methods before integrating with other software
- Driver module used for testing
  - Sets values of input parameters
  - Calls module to be tested and passes input parameters
  - Accepts return parameters from tested module
- Stub testing – test module simulates module not yet developed
  - In *Scribble* – this means testing an individual script or sprite

Integration testing

- Tests the behavior of a group of modules or methods
- Tests both normal processing and exceptions
- Errors can include
  - Interface incompatibility
  - Incorrect parameter values
  - Run-time exceptions
  - Unexpected state interactions
  - In *Scribble* – this means testing the interaction between sprites or sprites and the canvas or events

System testing

- Tests the behavior of the entire system
- Build and smoke test is performed daily to discover any problems with daily builds
  - System is completely compiled and linked each day
  - Battery of tests are run to smoke out problems
  - Any errors must be from changes made the prior day
- Complete system testing also performed before acceptance testing
  - In *Scribble* – this means testing the entire (final) application

Other whole of system tests

- Performance test checks time-based requirements
  - Response time
  - Throughput
- Acceptance test is system test performed to determine whether system meets user requirements
Test planning

A test plan is a project plan describing how the testing will be done
- What will be tested
- How long it will take (although this may vary, depending upon resource availability).
- What the test coverage will be, i.e. what quality level is required

- Often (and best) integrated with the SDLC being used

What to test?

- Test cases
  - Typical events and scenarios
  - Typical inputs
  - Weird events
  - Weird inputs
  - Bash testing
- What is in the acceptance test is (usually) determined when the contracts are signed
  - And will be the bare minimum the system will do!

The typical test plan structure

1. Introduction
   - Description of this Document
   - Related Documents
   - Schedule and Milestones
2. Resource Requirements
   - Hardware
   - Software
   - Test Tools
   - Staffing
   - Responsibilities
   - Training
3. Features To Be Tested / Test Approach
   - New Features Testing
   - Regression Testing
4. Features Not To Be Tested
5. Test Deliverables
6. Dependencies/Risks
7. Entrance/Exit Criteria
   (from SQAtester.com)
Test cases

- Important part of testing is specifying test cases (and test data)
- A test case is a formal description of
  - Starting state
  - Events to which software responds
  - Expected response or ending state
- Analysis phase documentation is useful in preparing test cases (use-case driven)
  - Should this be done by the analysts?
- Test data is defined to be used with a test case

Developing test data

- Create data
  - for normal usage scenarios,
  - improper usage scenarios,
  - weird usage scenarios.
- Test cases comprise the actions to be taken in the steps of the scenario, and the expected system responses
  - Test data will include inputs and expected outputs
  - For invalid data the expected output might be an error message
- Test data should cover all significantly different options

Significantly different if …

- Triggers a different (alternative) flow
- Triggers an error message
- Changes the user interface
- Causes different options to be available in a drop-down
- Is an input to a business rule
- Border condition
- Changes a default
- Entry format is not clearly defined
- International differences in formats (dates)
Example: Significantly different options

Identify the significantly different options for a license plate.

- Three letters, three numbers.
- Include spaces, negatives, special characters, too many letters, too many numbers, too few letters, too few numbers, boundary conditions (AAA000, ZZZ999), numbers first then letters, … and so on.

Testing in industry

- Often poorly done, undervalued, not well planned
- Can be a career (check seek.com.au)
- Many specialist consulting firms that just do testing
- Becoming increasingly important
  - Test-driven development
  - More development being done by specialist organizations
  - Cloud-based computing (service level agreements being used to manage IT functions)
  - Software architecture is becoming increasingly fragmented and specialized
  - Reputations can be quickly damaged by mistakes (competitors are a click away).
Automated testing tools

- Document and create test cases
- Automated running of test cases
  - Including GUI scripts
  - Load testing (might run on hundreds of client machines to test typical load)
- Recording of results
  - Comparison of actual to expected results
  - Comparison of different test trials
- Examples
  - JUnit, Selenium, Twist

Bugs and errors in software

- A “bug” is a program error, something that doesn’t work as it should
- Aim of testing is to identify errors
- Aim of programmers should be to avoid errors all together
  - Coding in a way that means that mistakes aren’t made
  - Coding in a way that means if they are made their impact is minimized, and that they will be easy to find

  In Scribble we do that by trying to structure our programs into a number of small manageable scripts, rather than few large ones
- In broad terms, there are three types of software bugs: syntax errors, logic errors and run-time errors

Error types

- Syntax errors
  - Not in Scribble
- Logic errors
  - “… an error that occurs because of a mistake on your part in the implementation of the programming logic you applied to solving a problem or performing a task.” – Ford (2009)
- Run-time errors
  - “… an error that occurs when a [Scribble] script attempts to perform an illegal action” – Ford (2009)
  - Scribble traps run-time errors for us – other languages/applications might simply “crash”
Example logic error
- Supposed to count from 1 to 5
- This version counts from 0 to 4
- This one counts from 1 to 5

Example run time error
- Divide by zero
- This is OK
- This isn’t
- Scribble looks after you. Using red to highlight the problem

Debugging strategies for Scribble
- Making noise - (Ford 2009)
- Display informative messages - (Ford 2009)
- Slow things down - (Ford 2009)
- Test scripts individually (unit testing) - (Ford 2009)
- Break things down into smaller pieces - (Ford 2009)
Some *Scribble* tips (cont).

- Get help from other people - POD
  - Use the Moodle discussion forum.
- Annotate code with comments – (Brennan 2010)
- Print out scripts to see them side by side - (Brennan 2010)
- Don’t make the mistakes (plan, code incrementally, test as you go) - (Brennan 2010)

**Summary**

- Testing is best thought about before coding
- Test data can be created before programs are finished
- Test data should include valid, invalid and weird value
  - Significant differences
- Unit tests test individual code elements (Scripts, Sprite)
- System tests test the complete system
- Lot’s we can do to avoid errors and track them down
  - Think and plan our code
  - Develop good structure
  - Debugging strategies
- Remember we often can’t see errors, another pair of eyes often helps
Reminders

- 1% in-class tests start this week (in both tutorials and laboratories)
- Start your good study habits now
  - If you revise your lecture notes and make a summary within 3 days retention is dramatically improved
- Keep playing with Scratch/BYOB/Scribble example projects
- Ask questions if you have them