Model-Based Testing

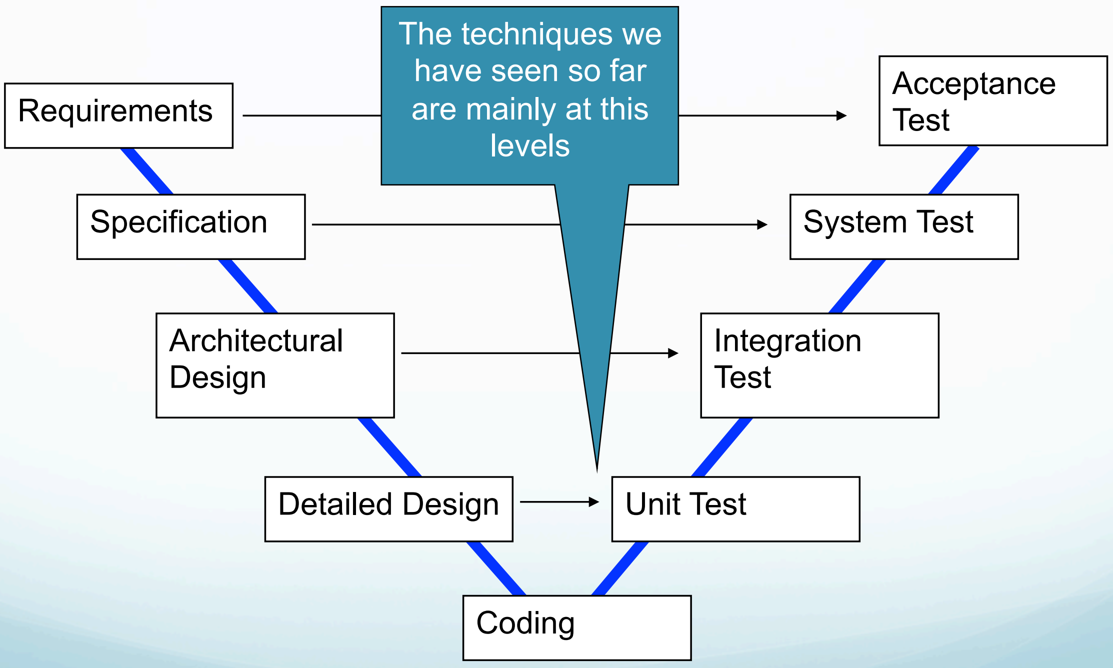
(DIT848 / DAT260)
Spring 2012

Lecture 4
Testing: The Bigger Picture

Gerardo Schneider
Department of Computer Science and Engineering
Chalmers | University of Gothenburg

Some slides based on material by Magnus Björk and Thomas Arts

The Bigger Picture



Unit tests

- Test the smallest components individually
- Often done by the programmer who wrote the code
- Less strict requirements of documentation
  - Large part of documentation replaced by executable test suites (Cunit, Junit or similar), which must therefore be clearly written
- No less important than any other test!
- In fact, maybe the most important test:
  - Unit tests easier to do than other tests – well invested time
  - Bugs discovered early easier to fix
  - So spending effort on unit tests reduces work later
- Recommended effort: equal amount of time spent writing code and unit tests

Unit tests – typical flow

Programmer:

- writes code
- runs static verification tool such as splint (for C)
- writes and runs unit test suite to test the code
  - Using framework such as Junit, CUnit
- complements black box test suite with white box techniques
  - Coverage checking (identify missing test cases) – Gcov, Emma
  - Valgrind: Monitor memory behaviour of C/C++ programs

The colleagues of the programmer do:

- Code review

Unit tests
Test Driven Development (TDD)

Programmer:

- writes unit test cases
  - Runs test suite, makes sure it fails
- writes code until test suite does not fail
  - Adds more test cases if needed
- runs static verification tool such as splint (for C)
- complements black box test suite with white box techniques
  - Coverage checking (identify missing test cases) – Gcov, Emma
  - Valgrind: Monitor memory behaviour of C/C++ programs

The colleagues of the programmer do:

- Code review

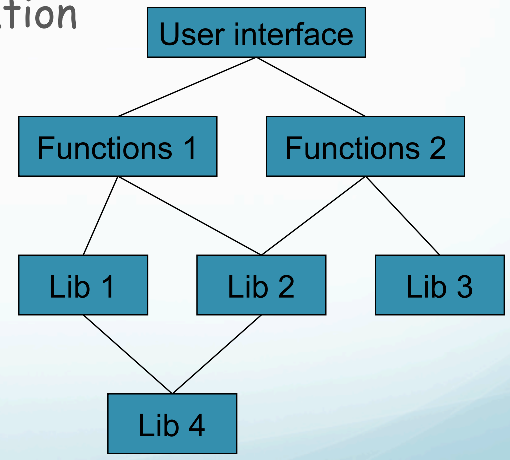
Benefits of TDD

- Focus on what the code does before implementation
  - Helpful when writing the code
- Programmer gets very quick feedback
- Easier to maintain
  - results in better coverage of unit test suite
- Note: TDD mandatory in XP

Integration tests

- Test different combinations of components

- Different strategies for integration
  - Big bang
  - Bottom-up
  - Top-Down
  - Sandwich



Thanks to Bruegge & Dutoit for material (through Magnus)

~~Integration tests: Big bang approach~~

- ~~After unit tests, integrate all components at once~~

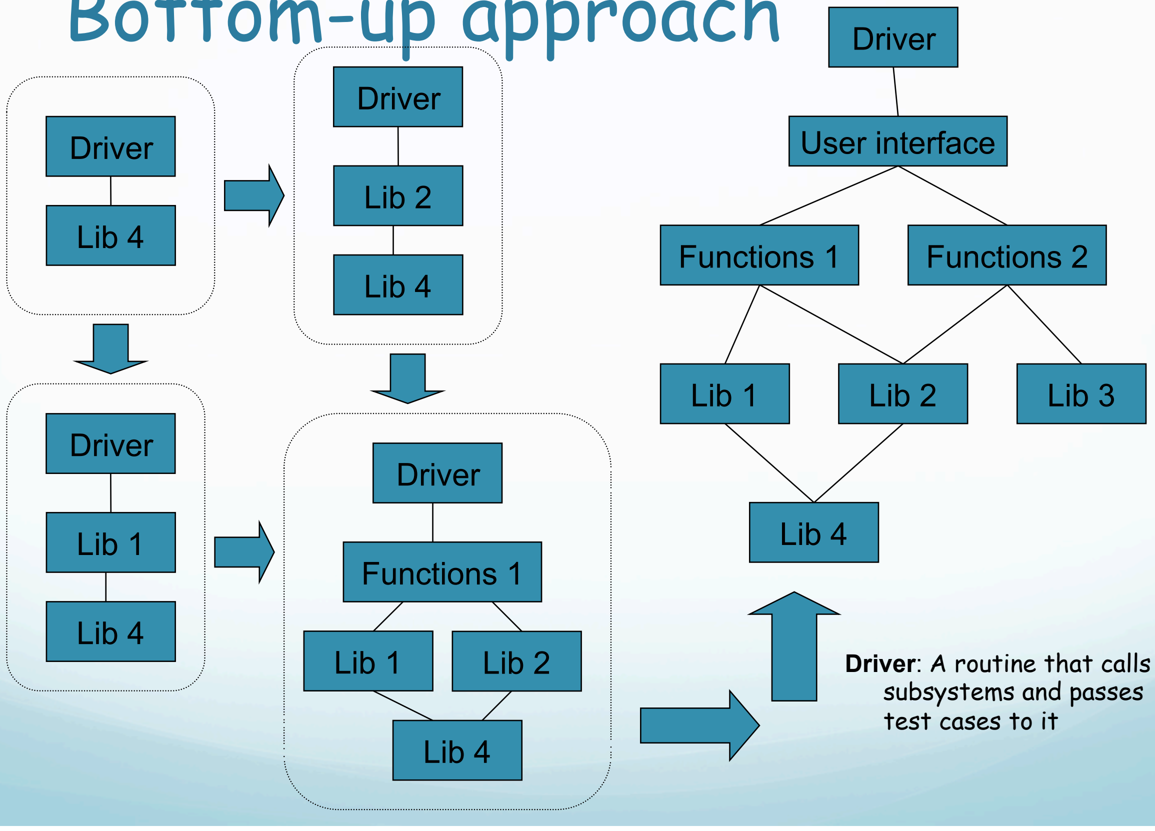
- ~~Essentially a system test~~

- ~~Bad idea! Don’t use it.~~
  - ~~Hard to locate bugs (have to search the whole system)~~
  - ~~Critical and peripheral components get the same attention~~
  - ~~Only possible very late in development cycle~~

Integration tests: Bottom-up approach

- Start with the subsystems in the lowest layer of call hierarchy
- Integrate such components with components that use them
- Done repeatedly until whole system is integrated
- Special code needed: Test driver
  - A routine that calls subsystems and passes test cases to it
- Can be done in frameworks such as CUnit

Integration tests: Bottom-up approach

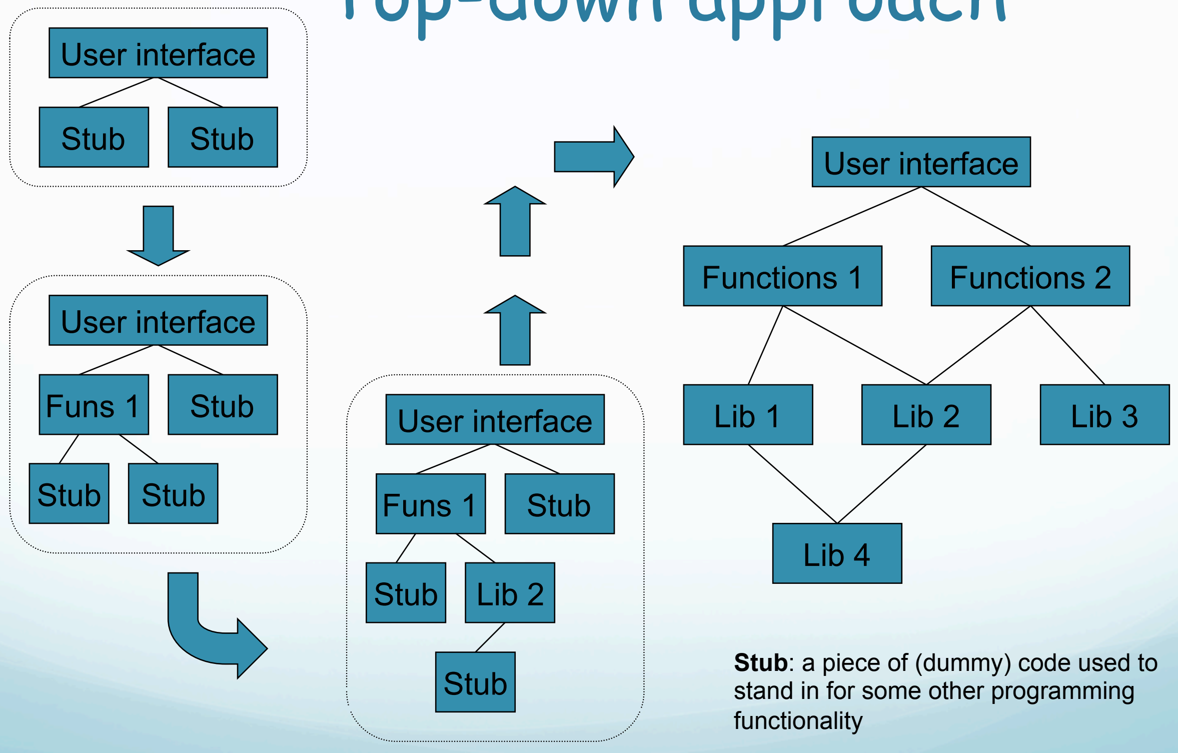


Driver: A routine that calls subsystems and passes test cases to it

Integration tests: Top-down approach

- Test top level components first, iteratively integrate components that are called by the components that already included. Repeat until the whole system integrated
- Special code needed: Stub
  - Has the same interface as the component it replaces
  - Returns fake data (probably described in the test case)
  - Passes information of the call to the test framework
- XUnit may be useful
- Pros:
  - Test cases defined in terms of program spec.
  - Easy to see behaviour at each stage (user interface)
- Cons:
  - Writing stubs difficult and tedious
  - Making automated test suite may be harder (e.g. if GUI)

Integration tests: Top-down approach

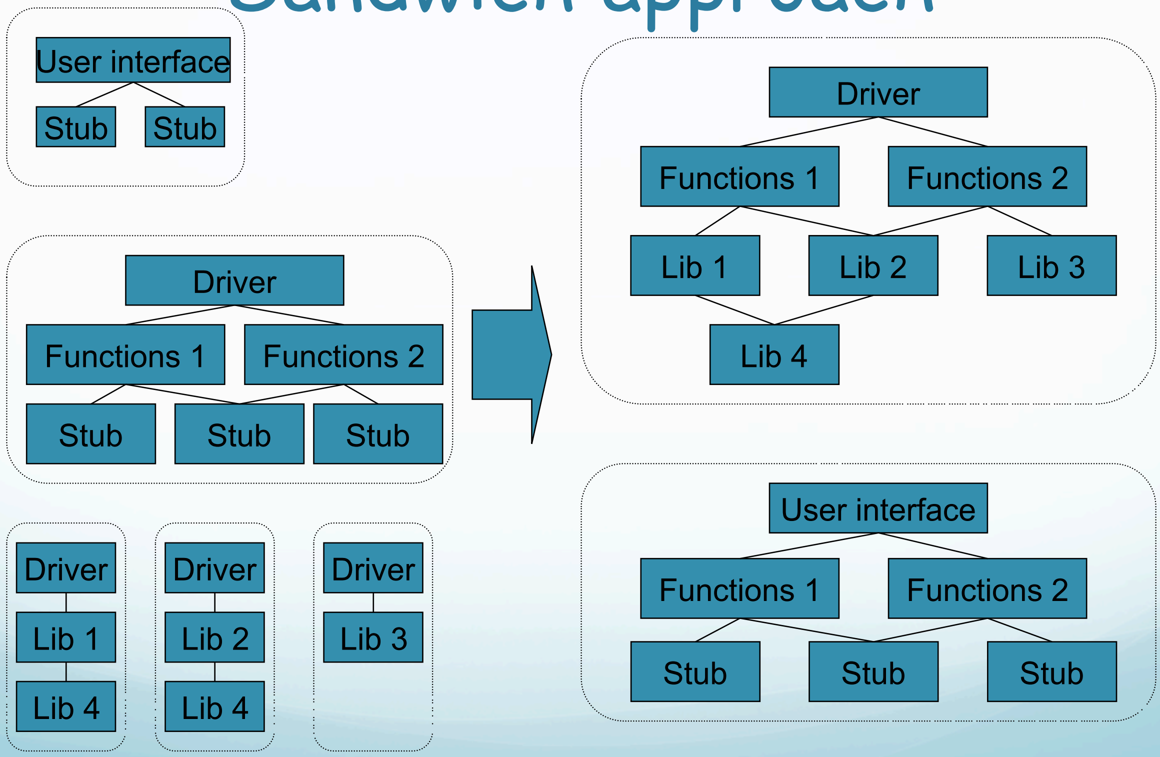


Stub: a piece of (dummy) code used to stand in for some other programming functionality

Sandwich approach

- Do both bottom up and top down, meet in the middle
- Much parallelization:
  - First phase:
    - Top layer with stubs
    - Middle layer with drivers and stubs
    - Bottom layer with drivers
  - Second phase:
    - Top and middle layer (top layer replaces drivers)
    - Middle and bottom layer (bottom layer replaces stubs)

Integration tests: Sandwich approach



Integration Tests: what to consider when choosing approach

- Which parts of the system are most critical?
  - Choose strategy that reveals error in critical parts early, and includes critical parts in many tests
- Which approach means less work?
  - Top level test may be harder to automate (e.g. GUIs)
  - How to minimize work spent writing drivers and stubs
- Availability of components
  - If coding done bottom-up, then bottom-up integration tests can be started earlier

To make integration tests easier

- Do thorough unit tests
- Make well defined interfaces between modules

System tests

- Test the full system
- Cover full specification
- Test automation may be hard to achive
  - System tests may be expensive and tedious

- Combine black and white box testing as before
- Test both normal and abnormal uses of the system
  - Performance testing
    - Push system to its limits
    - The goal is to try to break the system
    - May be used to identify bottlenecks, to be dealt with in next iteration of development

System tests: Performance testing

- **Stress testing:** exceed parameters: number of requests, …
- **Volume testing:** large amounts of data
- **Configuration testing:** different combinations of HW & SW
- **Compatibility testing:** use with older systems
- **Security testing:** try to break in
- **Timing testing:** time responses & functions
- **Environmental testing:** effects of temperature, movement, …
- **Quality testing:** reliability, maintainability, availability
- **Recovery testing:** erroneous or missing input
- **Human factors testing:** test user interface on users

Acceptance tests

- Customer mainly responsible for acceptance test
- **Alpha** testing
  - Done by customer under supervision of developer
  - Usually done in controlled environment (developer's systems)
  - Developer can quickly fix bugs
- **Beta** testing
  - Product used by customers in real environment
  - Developers typically not present
  - Difference from rest of product lifetime:
    - Often only selected customers
    - Customer cannot rely on software

Fixing bugs

- Action depends on severity of bug
  - Low-priority failures may be put on "known bugs" list, included in release notes
- Always do regression test after fixing bugs!
  - Bug fixes are likely to break something else
- Bug tracking tools often useful (Example: Bugzilla)
  - Maintains list of bugs
  - Assigns priorities and responsible people for each bug
  - Keeps reminding people about their high priority bugs
  - Searchable bug index (with history)

Regression tests

- Must be done after every change to source code
- Regression tests significantly cheaper if test suite is automated
- Sometimes not feasible to redo all tests. If so, identify a subset of cases that cover as much as possible.
- Tool: Tinderbox
  - Automatically checks out committed code, compiles it and runs test suite (needs other tool for that, such as DejaGnu)
  - Identifies compilation errors and failing test cases
  - Points out who's responsible
  - Maintains history
  - Often runs 100% of the time on a bunch of dedicated machines

Test in General: a test…

Determine whether the statements are true or false. If a statement is false, justify your answer

1. There are two kinds of testing: dynamic and static.
2. If you get 100% code coverage then you can guarantee that your software has been thoroughly tested and can stop testing.
3. XUnit is better than JUnit since you can get better tests.
4. The V model teaches us that we can do acceptance tests as soon as we have the requirements, even before we start developing.
5. Different testing methods and techniques apply to each test level (as presented in the V model).
6. Testing and debugging are the same.
7. One good thing about the sandwich approach for integration testing is that enhances parallelization (that is, developers and testers can work in parallel).
8. Performance testing is one kind of test, part of the so-called system testing.
9. The best way to do integration test is the sandwich approach.

Groups 2-5 persons: 15 min

Test in General: solution…

1. F – Testing is by definition *dynamic*
2. F – Code coverage is only one aspect; there is no guarantee in general to get 100% confidence
3. F – Xunit is a family of test units, including CUnit (see lect.5 sl.8)
4. F Acceptance test is only done after there is something to test against the requirements. Done by customer.
5. T
6. F – Testing: establish the existence of defects; debugging: locating & repairing those errors found during testing
7. T
8. T
9. F – depends on how the system is built.

Assignment 2

- Continue with last week's assignment – You get a correct implementation of the calculator
    - Use EclEmma (coverage) to identify missing test cases and add them.