

Test-driven development



**Idaho State
University**

Computer
Science

Isaac Griffith

CS 3321
Department of Computer Science
Idaho State University

ROAR



Topics covered

- Development Testing continued
- Test-driven development

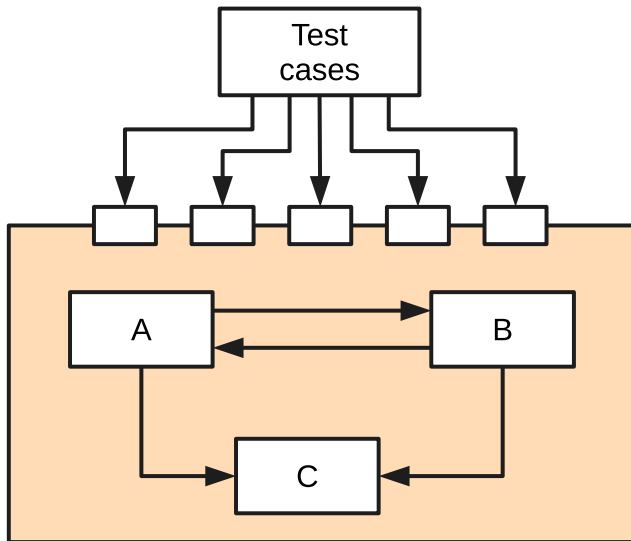


Interface testing

- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.
- Interface types
 - Parameter interfaces Data passed from one method or procedure to another.
 - Shared memory interfaces Block of memory is shared between procedures or functions.
 - Procedural interfaces Sub-system encapsulates a set of procedures to be called by other sub-systems.
 - Message passing interfaces Sub-systems request services from other sub-systems



Interface testing





Interface errors

- Interface misuse
 - A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order.
- Interface misunderstanding
 - A calling component embeds assumptions about the behavior of the called component which are incorrect.
- Timing errors
 - The called and the calling component operate at different speeds and out-of-date information is accessed.



Interface testing guidelines

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- Always test pointer parameters with null pointers.
- Design tests which cause the component to fail.
- Use stress testing in message passing systems.
- In shared memory systems, vary the order in which components are activated.



System testing

- System testing during development involves integrating components to create a version of the system and then testing the integrated system.
- The focus in system testing is testing the interactions between components.
- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.
- System testing tests the emergent behavior of a system.



System and component testing

- During system testing, reusable components that have been separately developed and off-the-shelf systems may be integrated with newly developed components. The complete system is then tested.
- Components developed by different team members or sub-teams may be integrated at this stage. System testing is a collective rather than an individual process.
 - In some companies, system testing may involve a separate testing team with no involvement from designers and programmers.



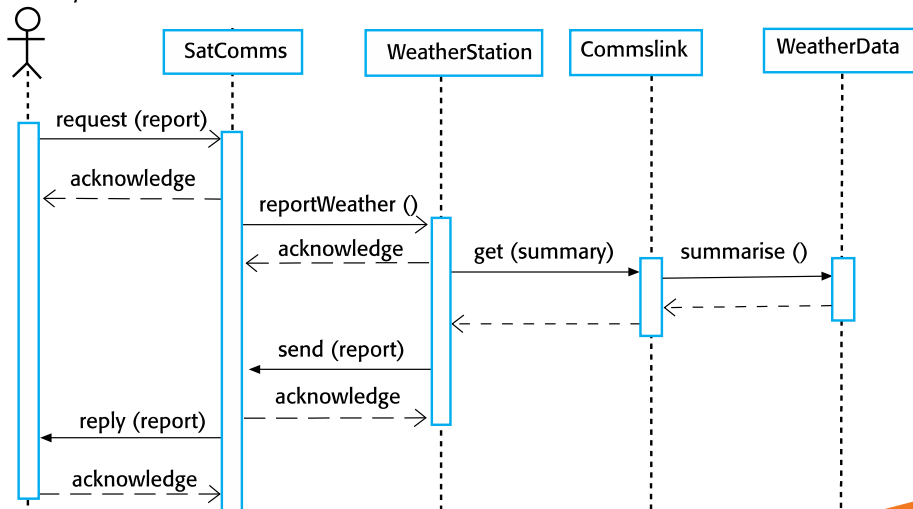
Use-case testing

- The use-cases developed to identify system interactions can be used as a basis for system testing.
- Each use case usually involves several system components so testing the use case forces these interactions to occur.
- The sequence diagrams associated with the use case documents the components and interactions that are being tested.



Collect weather data sequence chart

information system



Test cases derived from sequence diagram

- An input of a request for a report should have an associated acknowledgment. A report should ultimately be returned from the request.
 - You should create summarized data that can be used to check that the report is correctly organized.
- An input request for a report to WeatherStation results in a summarized report being generated.
 - Can be tested by creating raw data corresponding to the summary that you have prepared for the test of SatComms and checking that the WeatherStation object correctly produces this summary. This raw data is also used to test the WeatherData object.



Testing policies

- Exhaustive system testing is impossible so testing policies which define the required system test coverage may be developed.
- Examples of testing policies:
 - All system functions that are accessed through menus should be tested.
 - Combinations of functions (e.g. text formatting) that are accessed through the same menu must be tested.
 - Where user input is provided, all functions must be tested with both correct and incorrect input.

Test-Driven Development

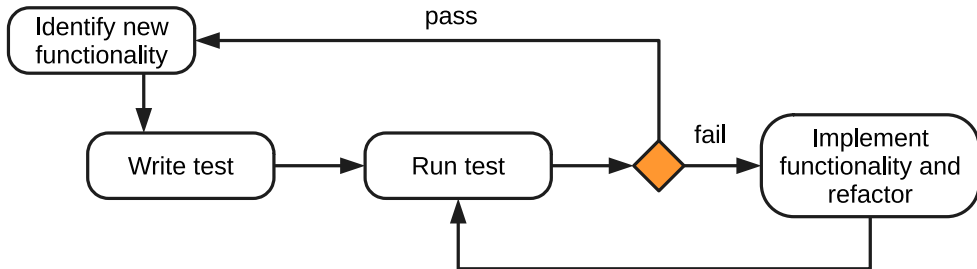


Test-driven development

- Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development.
- Tests are written before code and 'passing' the tests is the critical driver of development.
- You develop code incrementally, along with a test for that increment. You don't move on to the next increment until the code that you have developed passes its test.
- TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.



Test-driven development





TDD process activities

- Start by identifying the increment of functionality that is required. This should normally be small and implementable in a few lines of code.
- Write a test for this functionality and implement this as an automated test.
- Run the test, along with all other tests that have been implemented. Initially, you have not implemented the functionality so the new test will fail.
- Implement the functionality and re-run the test.
- Once all tests run successfully, you move on to implementing the next chunk of functionality.



Benefits of test-driven development

- Code coverage
 - Every code segment that you write has at least one associated test so all code written has at least one test.
- Regression testing
 - A regression test suite is developed incrementally as a program is developed.
- Simplified debugging
 - When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified.
- System documentation
 - The tests themselves are a form of documentation that describe what the code should be doing.



Regression testing

- Regression testing is testing the system to check that changes have not 'broken' previously working code.
- In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward. All tests are rerun every time a change is made to the program.
- Tests must run 'successfully' before the change is committed.



Key points

- Test-first development is an approach to development where tests are written before the code to be tested.
- Scenario testing involves inventing a typical usage scenario and using this to derive test cases.



Are there any questions?