Real Time System Testing - MIT 16.070

Real Time System Testing - Lectures 30-32

Why are we learning this?

- A Real Time System must run continuously until it has been powered off
- Failure to run properly can mean great economic loss and/or loss of human life

A LARGE PART OF REAL TIME SOFTWARE DEVELOPMENT MUST FOCUS ON AVOIDING FAILURE

Real Time System Testing

- The next three lectures will focus on:
  - Lecture 30: *(R 11.3)*
    - How to minimize failure in real time systems
    - Methods used to test real time systems
  - Lecture 31: *(R 13)*
    - What is Software Integration?
    - Test Tools
    - An example approach for integration and test of the MIT 16.070 final project
  - Lecture 32: *(R 11.4)*
    - Fault Tolerance
    - Exception Handling
    - Formal Test Documentation

Real Time System Testing - MIT 16.070

Lecture 30

Real Time System Testing

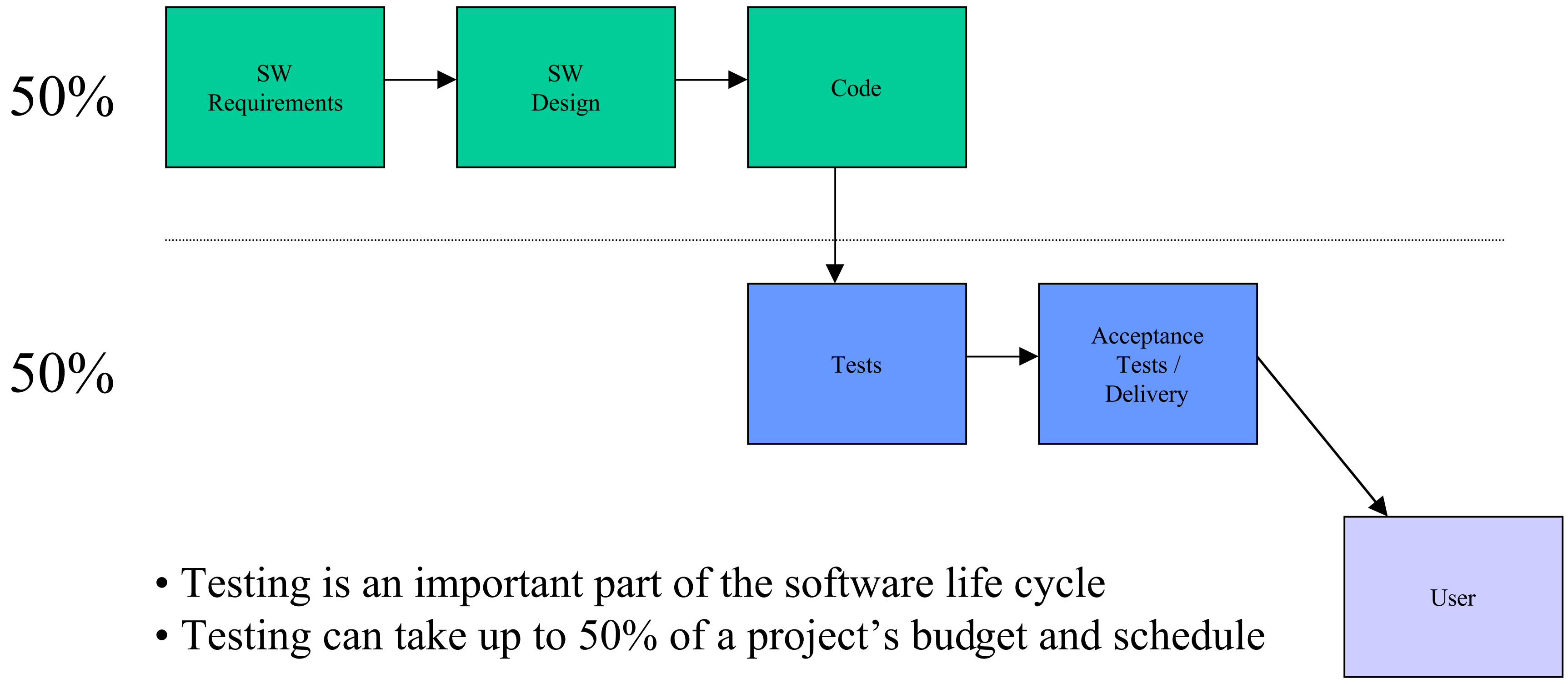
- What are some reasons that Real Time software can fail?

How do we minimize failure in real time systems?

- Testing, Testing, Testing...
  - Software Test (various levels)
  - System Test

- Build test and recovery software and/or hardware into the design
  - Exception handling (limit checking, etc.) to recover from bad data inputs
  - Redundant hardware solutions *(e.g. 5 processors on Space Shuttle)*
  - Fault Tolerant software and hardware

Real Time System Testing



- Testing is an important part of the software life cycle
- Testing can take up to 50% of a project's budget and schedule

Real Time System Testing

- The goal of software testing a program is to find and fix errors prior to delivery to the end user

- Testing:
  - Uncovers errors
  - Fixes errors
  - Measures requirements conformance
  - Provides an indication of quality

- Testing a real time system is often difficult because of the very nature of real time systems

Why Test? - Some Real World Examples

- An F-16 pilot was sitting on the runway doing the pre-flight and wondering if the computer would let him raise the landing gear while on the ground….it did!
- When initially developing the sidewinder missile mounting, there were a few problems. The software would release the latch and fire the missile. Initially, the latch was closed too quickly and the missile could never leave the wing. Imagine the pilot's dismay when there was suddenly extra thrust on one of the wings!!
- The F-16 has a sophisticated s/w system that performs load balancing to optimize flight performance. This includes dropping empty fuel tanks in such a way as to balance the plane. A minor prerequisite to dropping the tank was overlooked in the software - it is usually a good idea to be upright when releasing the tanks. Imagine flying upside down and having empty fuel tanks come flying off!

From a talk by Nancy Leveson, Dept. of CS and Eng, Univ of Washington on
SW Safety and Reliability (IEEE and ACM sponsored 4/20/94)

Testing Considerations

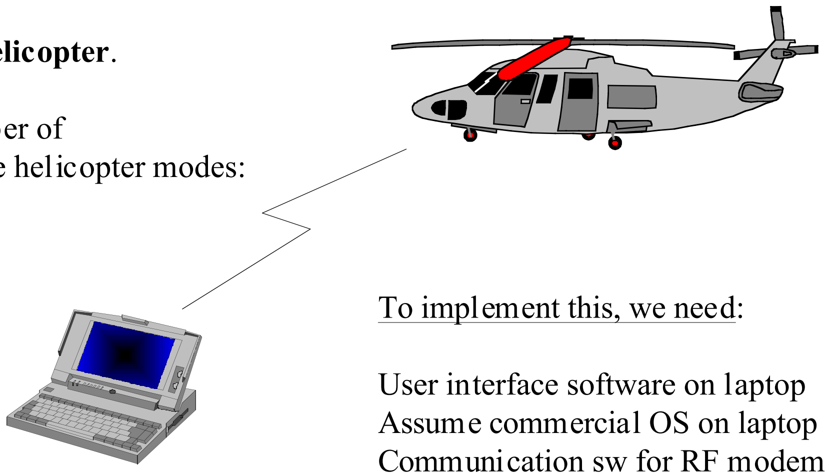
- Testing Techniques - the different approaches to testing software
- Types of Tests - gaining different perspective from a suite of tests
- Levels of Testing - determining how much granularity is needed?

Let’s apply what testing means in an example system:

A remote controlled helicopter.

A user can enter a number of commands to control the helicopter modes:

- Take off
- 30-degree right turn
- 30-degree left turn
- Forward flight
- Hover
- Land



The helicopter monitors its fuel remaining and communicates its mission time remaining back to the laptop.

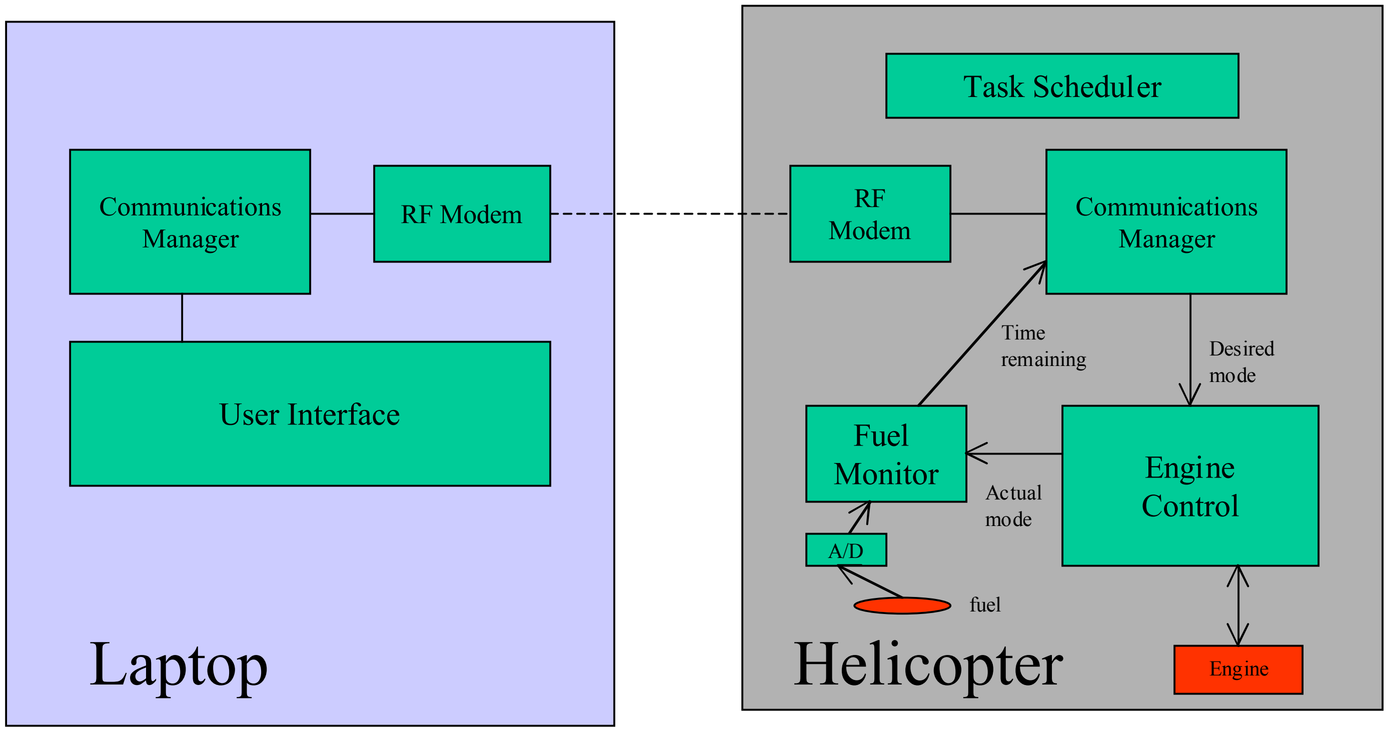
Communication between the vehicle and the laptop is handled by a radio frequency (RF) modem.

The helicopter must work autonomously if the laptop communication link is lost.

To implement this, we need:

User interface software on laptop
Assume commercial OS on laptop
Communication sw for RF modem (driver) on both
Comm mgr (decodes messages for application) on both
Engine control software to process commands on helicopter
Fuel monitoring software on helicopter
Task scheduler on helicopter

Remote Controlled Helicopter Example



Testing Techniques

- Black Box Testing
  - Only inputs and outputs of functions are considered
  - How outputs are generated based on a set of inputs is ignored
  - Run a suite of test cases
    - Exhaustive combination of all inputs
    - Corner cases (min, max, avg)
    - Pathological cases (inputs likely to result in error)
  - Disadvantage: Often bypasses unreachable code

- Helicopter Example: Fuel Monitor
  - Test to verify mission time remaining outputs given varied fuel level inputs and helicopter modes.

Testing Techniques

- White Box Testing
  - Exercises all paths in a module
  - Driven by logic
  - Static Example: Code Inspections
    - group walkthrough of software logic
    - inspect code line-by-line
  - Dynamic Example: Test all the links and buttons on a web page

- Helicopter Example:
  - Test communication interface between laptop and helicopter to confirm all modes can be exercised.
  - Code inspection for Engine Controller to confirm appropriate engine controls for each helicopter mode

Levels of Real Time System Testing

- Unit Testing
- Software Integration Testing
- Software Validation and Verification Testing
- Software / Hardware Integration Testing
- System Testing

Unit Testing

- Focuses on smallest unit of software (function, module, procedure)
- Important control paths are tested
- Usually developed by the software engineer who wrote the unit
- Usually white-box oriented

- Helicopter Example : Communication Manager Testing
  - Verify the Comm Mgr can parse data from the user interface and send it to the RF modem
  - Exercise all paths within the Comm Mgr Software Unit(s)
  - Unit test could also include standalone tests of individual functions called by the Communication Manager

Software Integration Testing

- Testing that occurs when unit tested modules are integrated into the overall program structure
- Test focuses on the interfaces between software modules
- May be performed by developer or by independent test team
- Black box testing perspective
- Drivers and stubs will be required for external interfaces

- <u>Helicopter Example</u> : Fuel Monitor / Engine Control Testing
  - Use stubs for fuel level from A/D converter, mode selection from Comm Manager and data from other engine interfaces
  - Verify Fuel Monitor output given different mode inputs from Comm Manager.

Software Integration Testing

- What are drivers and stubs?
  - Simple software used to “plug” a complex interface during test
  - To the software under test, a drive or stub will behave identically to the real software
  - The “meat” behind the response is missing
  - <u>Example</u>
    - Interface to a real robotic arm takes commands and provides status response
    - Software stub would take a command, look up the right response in a table and wrap the response back to the software under test

Software Validation and Verification Testing

- Software Verification: Software tests designed to find errors in the system
- System Qualification: Demonstration of compliance with requirements to <u>sell</u> the system
- <u>Helicopter Example</u> :
  - Demonstration of user commands entered via the UI (SW)
  - Demonstration of user commands in control of the real helicopter (SYS)

Software / Hardware Integration Testing

- Software is incorporated with other system elements (stubs and drivers from SW integration testing are replaced with real systems)
- Groupings are tested to determine if they work together
- Process iterates until all hardware and software has been grouped into the full system
- Focus is on hardware/software testing

Software / Hardware Integration Testing (continued)

- Helicopter Example : Fuel Monitor / Engine Control Testing
  - Use stubs for mode selection from Comm Manager
  - Replace stubs with real fuel indicator and A/D converter, Replace stubs that replaced engine interfaces
  - Verify Fuel Monitor output given different mode inputs from Comm Manager stubs.
  - Eventually replace the stubs with the real Comm Manager and RF Modem interface to the laptop

System Testing

Confirming the system can endure real operation and meets its specified functional requirements. These tests include….

- Recovery Testing
  - Forcing the system to fail in a variety of ways and witnessing system recovery
- Security Testing
  - Stressing the protection mechanisms built into the system
- Stress testing
  - Confront the program with normal and abnormal situations
- Performance testing
  - Verifying that the system operates within its performance limits

System Testing (continued)

- Helicopter Example
  - Try a normal mission scenario (takeoff, hover, forward flight, right turn, left turn, hover, land)
  - Try illegal state transitions (forward flight --->land)
  - Cycle power on the laptop in the middle of a helicopter flight
  - Incorporate use of a simulation (just in case these last two don’t work right the first time)

Real Time System Testing - Summary

- Test is an important part of the realtime software life cycle
- A large portion of our effort must focus on avoiding failure
- Testing:
  - Uncovers errors
  - Fixes errors
  - Measures requirements conformance
  - Provides an indication of quality
- Testing can be black box (inputs/outputs) or white box (software paths) oriented
- Testing is performed at several levels (unit, software integration, sw/hw integration, system)
- Recommended Reading for this lecture: R 11.3
- Recommended Reading for next lecture: R 13