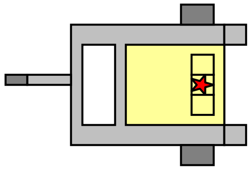
Aerospace Software Development & The Mars Rover Problem Lecture 2



Heidi Perry
Draper Laboratory
Unified Lecture
November 10, 2005

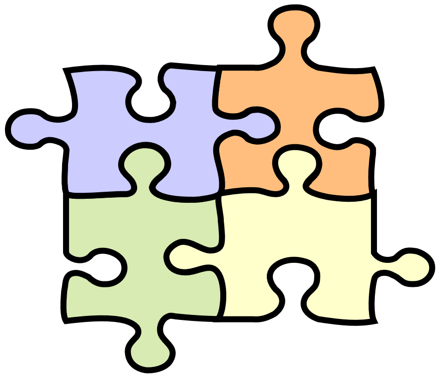
Today’s Lecture

- Software Reuse
- Coding Tips: The Data Log
- Software Lifecycle & Supporting Artifacts
- Software Test
- Unified Test Report
- System Problem Logistics

Software Reuse in the Real World

- Software Frameworks
- Open Source Development
- Internal code reuse between projects
- Libraries that come with development environments
- “Hey Joe, do you have any code that does this?”

Levels of Software Reuse



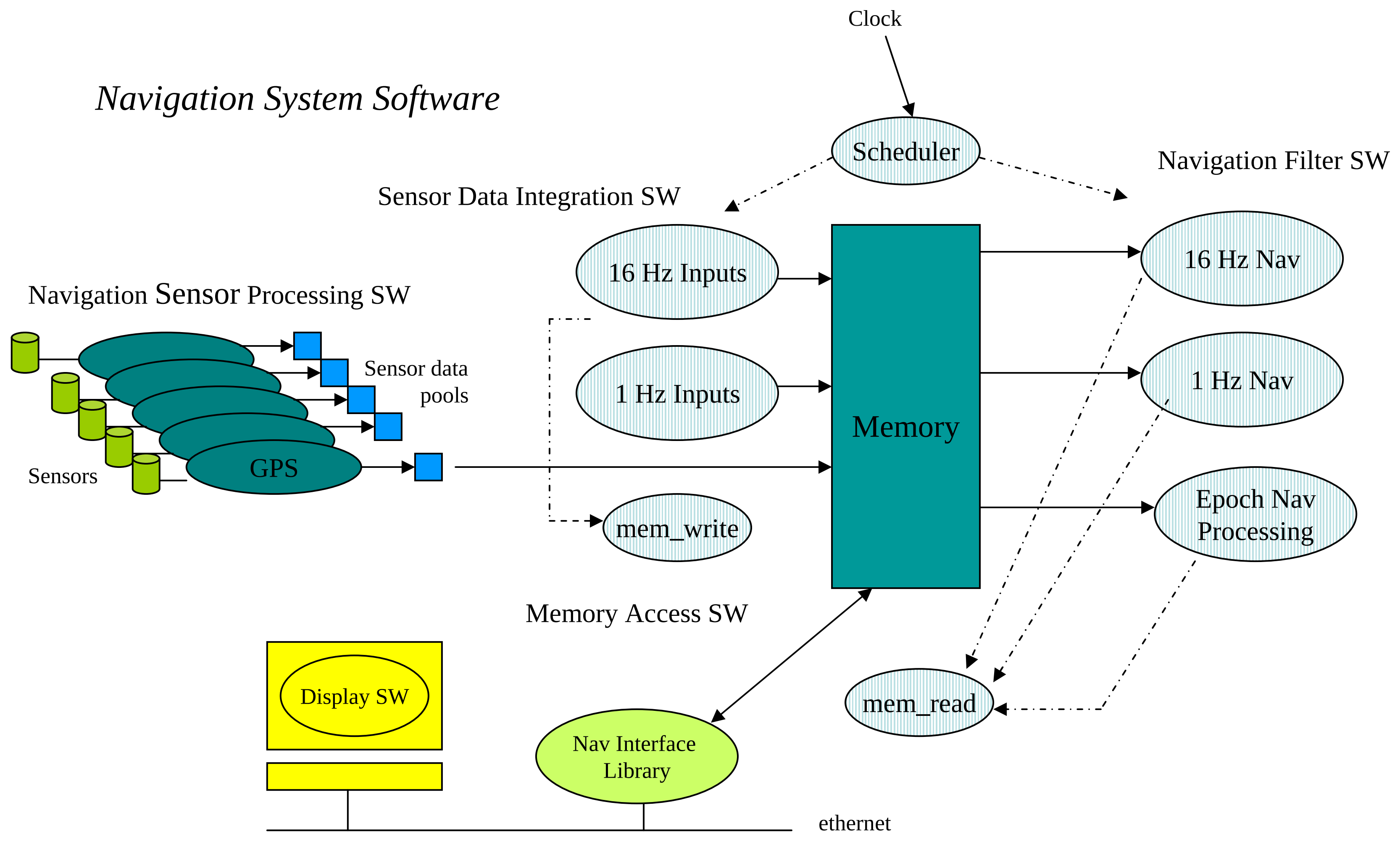
- Some fully reused
- Some calibrated, tweaked
- Some enhanced
- Some is just drawing on experience
- Some is small code snippets

Full Reuse Example: Frameworks

- Frameworks are a set of software procedures to support a particular application area
  - Includes main program to invoke the procedures
  - Additional software is written to tailor the framework for each new project
  - Often proprietary

- Examples
  - Navigation Framework
  - Autonomy Framework

Simple Navigation Software Framework



Unified Software Reuse

- How might you draw upon things that you've already done in the homework in order to solve the system problem?
  - Hint: Also look at the homework solutions when considering areas of reuse, particularly if your solution didn't work as well as you had hoped.
- The most obvious answer is the full reuse of the rover itself…
  - Problem C4: Build the Rover

Unified Software Reuse (2)

- The control algorithms from problem C8 will certainly be useful in driving around the grid.
  - Turn_Left_90 & Turn_Right_90
    - should be fully reusable (i.e. no change)
  - Go_Forward and Go_Backward
    - should be reusable by calibrating it: Find an appropriate calibration constant so that the rover can go forward and backward by *one square at a time*

Unified Software Reuse (3)

- The Simple Navigation problem from C10, though not written for the rover, could certainly be useful in using an array to store information about the grid.
  - It will be useful to turn the code from C10 into a procedure that aids in rover navigation. Think about what information the procedure can provide to the rest of your algorithm. You may want to consider using more then one output parameter.
  - The algorithm will obviously need to be modified to use a 5x5 grid instead of a 10x10.
  - The homework solution contained a particular concise way of solving the navigation problem. You will likely need to draw on this solution (or another concise solution) in order to make your code fit on the rover.

Unified Software Reuse (4)

- Finally, small parts of the code from the Discrete Rover Problem (C13) will be useful.
  - In particular, the code related to calibrating and checking the light sensors will be of use, as you will need to be able to find black squares as part of the system problem.

Coding Tips: Using the Datalog

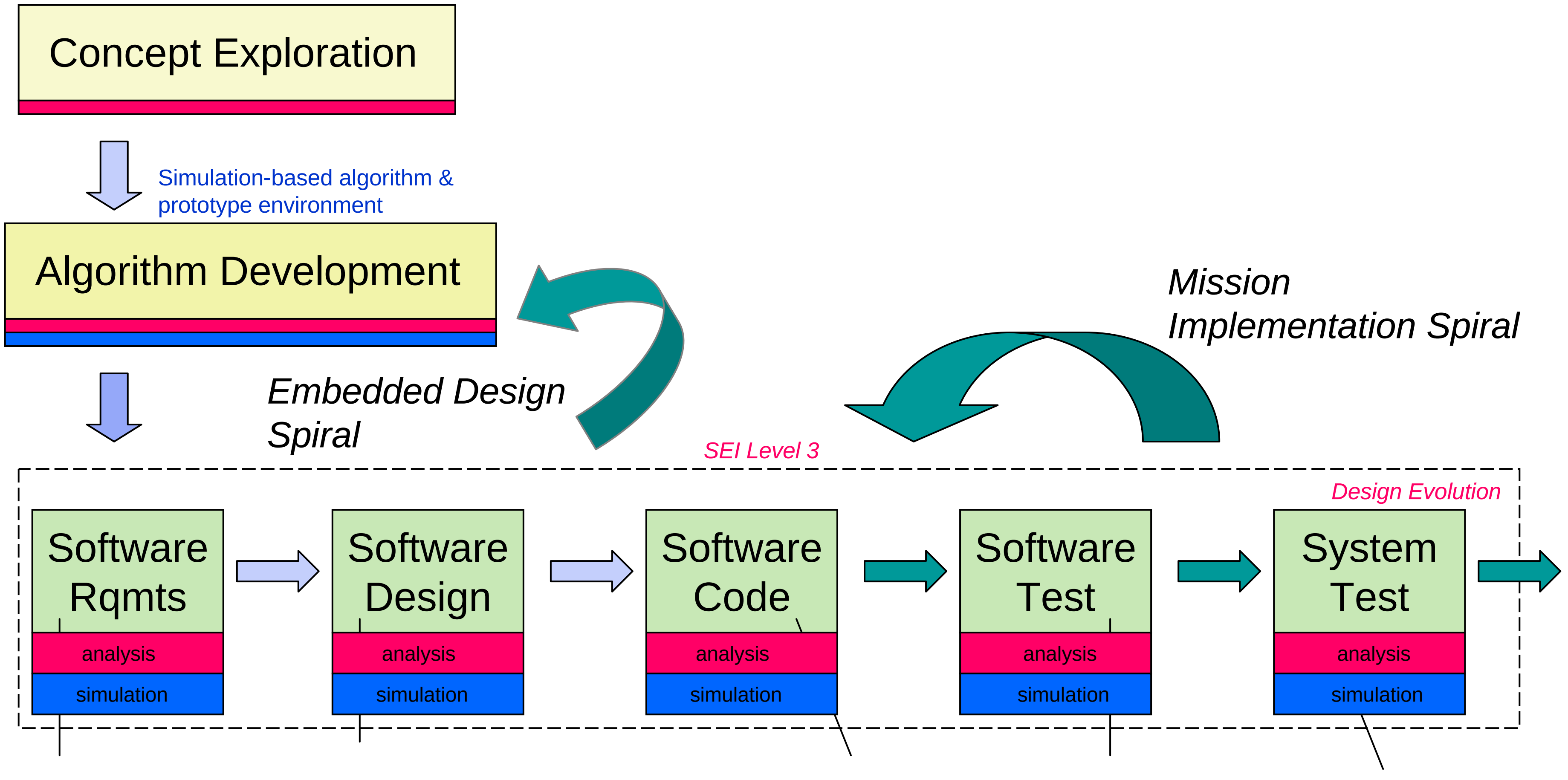
- There are two procedures defined in the Ada Mindstorms manual:

```
procedure Create_Datalog
   (Size : in Datalog_Range);

procedure  Add_To_Datalog
   (Value : in integer);
```

- These are equivalent to the procedures documented in the Not-Quite-C manual.

Embedded SW Lifecycle



Mathematics Models
Descriptive Models
Requirements DB
MATLAB

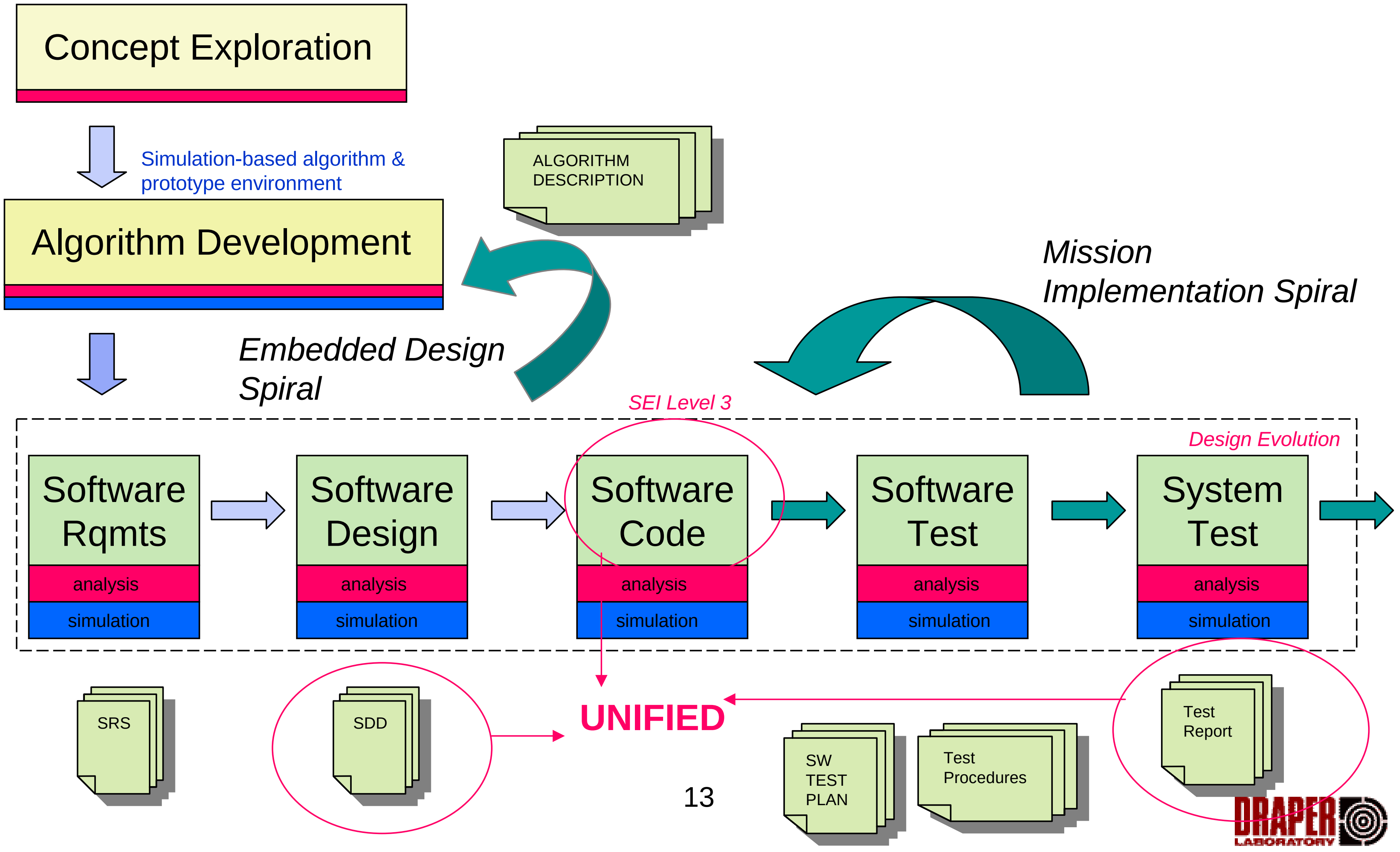
Model Driven Architecture / Specification Languages
Functional Decomposition
Object Oriented Design / UML
Draper SW Frameworks (Autonomy, GN&C, C&DH,
Fault Tolerant SW – Design & Code for reuse)

C/C++, Java, Ada
Assembly
MATLAB /
Simulink RTW Autocode

SW Unit Testing
SW Component Testing
Incremental Build Test
SW Verification
Model-Based Tests
Formal Methods

SW Validation

Supporting Artifacts (Documents)



Why Test? - Some Real Rover Examples

Mars Pathfinder - a multitasking priority problem

- Successful landing July 4, 1997
- A few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".
- Why? Priority inversion in multitasking Real Time OS. An upload of new software fixed the problem.

Why Test? - Some Real Rover Examples

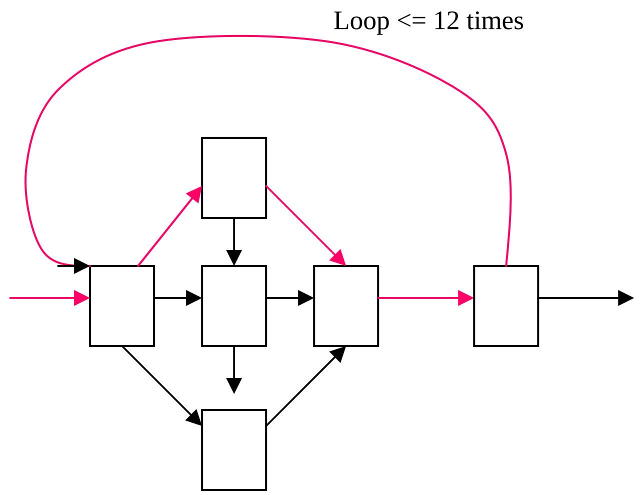
Mars Spirit – a memory allocation failure

- January 2004
- By design, data is collected by Spirit, files are created and stored in the flash file system until a communications window opens — an opportunity to transmit the data either directly to Earth or to one of the two orbiters circling the Red Planet. The data is still held in the flash system until retrieved and error-corrected on Earth
- Software command sent to Mars Rover to delete files was not correctly received
- As data collection continued, Spirit attempted to allocate more files than the RAM-based directory structure could accommodate. That caused an exception, which caused the task that had attempted the allocation to be suspended. That in turn led to a reboot, which attempted to mount the flash file system. But the utility software was unable to allocate enough memory for the directory structure in RAM, causing it to terminate, and so on.
- Spirit fell silent, alone on the emptiness of Mars, trying and trying to reboot
- Software engineers brought back the rover by uploading a series of low level file manipulation commands

SW Testing Techniques

- Black Box Testing
  - Only inputs and outputs of functions are considered
  - <u>How</u> 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

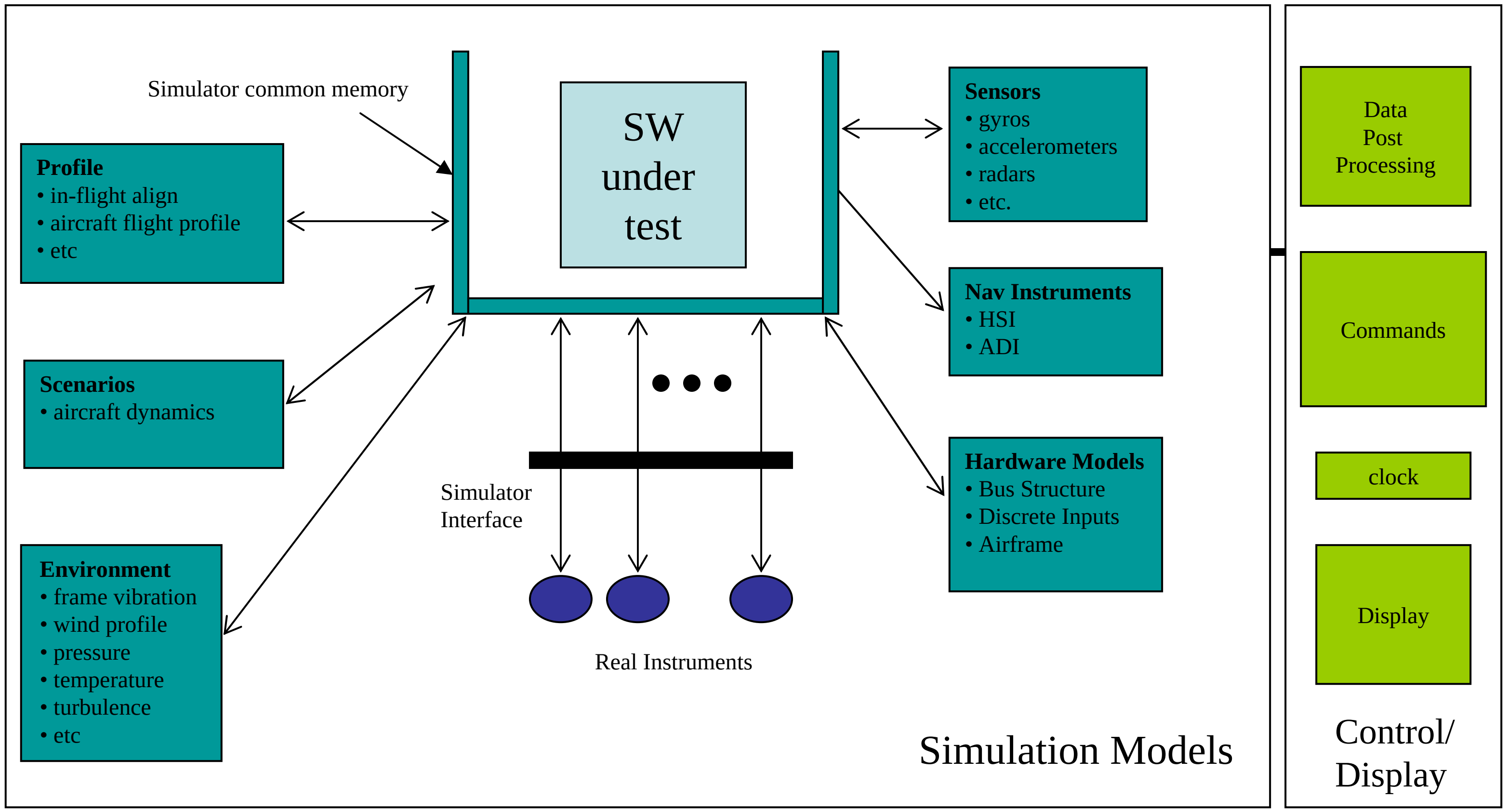
SW Testing Techniques



- White Box Testing
  - Exercises all paths in a module
  - Driven by logic
  - Static Example: Reviews (i.e. peer reviews)
    - Reading the code is a form of test. Code inspection can find a surprising number of problems.
  - Dynamic Example: Test all the links and buttons on a web page

SW Testing Techniques

A Simulation Test Bed Example



Levels of Real Time Software 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, method, procedure)
- Important control paths are tested
- Usually developed by the software engineer who wrote the unit

- Rover Example : Testing Read_Next_Square

```
Read_Next_Square()
          go forward 1 squares
          squares_visited = squares_visited + 1
          if squares_visited = 4
                       turn 90 degrees (right)
                       squares_visited = 1
                       corners_turned = corners_turned + 1
          return check sensor data
```

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 may be required for external interfaces

- Rover Example : Rover test on the grid

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
  - Real World Example:
    - 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

- Verification asks, "Is the product being built right?" It is the process of determining whether or not the products of a given phase of the software development cycle fulfill the established requirements.*

- Validation asks, "Is the right product being built?" It evaluates software at the end of the development lifecycle to ensure that the product not only complies with standard safety requirements and the specific criteria set forth by the customer, but performs exactly as expected. *

- Rover Example :
  - You will verify your rover system prior to demonstration
  - You TA will validate the rover software

** From NASA IV&V facility: http://www.ivv.nasa.gov/faq/index.shtml*

Testing for the System Problem

- Consider Using Inspections
  - Pen & Paper testing of your algorithm
    - Be the rover…think through your algorithm to try to break it on different grids without actually building the grids
- For Integration Testing, Gelb lounge has the Martian surface
  - Movable tiles
  - Try various configurations (start with a simple test, then try more complex configurations)

Assignment Details

- Part I: Due 10 November 2005 (75 pts)
  - Initial Software Design Document (team)
- Part II: Due 17 November 2005 (75 pts)
  - Listing of completed software (team) *(also submit code)*
  - Updated Software Design Document (to match completed software) (team)
- Part III: 28-30 November 2005 (50 pts)
  - Rover Demo (team)
  - Test Report (individual)
    - Identifies tests performed to develop completed rover
    - Includes copy of rover output file
    - Includes path reconstruction given rover output

Test Report

- Provides context for the item under test, as well as test results
- Stand-alone document
- Ordinarily really huge…for the system problem your report can be small (3-4 pages)
  - Reminder: Each **person** must turn in an **individual** test report.

Test Report Contents

1. Executive Summary
1.1 System Overview
1.2 System Architecture & Software Components
2. Test Environment
2.1 Software Items Under Test
2.2 Components in the Software Test Environment
3. Test Results
3.1 Overall Assessment of the Software Tested
3.2 Detailed Test Results
    3.2.1 Rover Maneuver Tests
    3.2.2 Sensor Data Tests
    3.2.3 Mission Data Transmit Tests
    3.2.4 System Tests

System Problem Test Report (Executive Summary)

1.1 System Overview

This paragraph shall briefly state the purpose of the system and the software to which this document applies. It shall describe the general nature of the system and software; summarize the history of system development, operation, and maintenance.

1.1 System Overview Suggestions

- Summarize the rover requirements (feel free to use SP6 system lab write-up material)
- Identify your team mates

System Problem Test Report (Executive Summary)

1.2 System Architecture & Software Components

This paragraph shall identify the parts of the system (i.e. hardware and software) that were either given or developed as part of the problem. Include how you created your software code (for example, how did you use solutions to the psets?)

1.2 System Architecture - Suggestions

- Identify the hardware and its purpose (i.e. lego RCX, describe the sensors, the batteries)
- Describe the software your team wrote, (most likely a navigation algorithm with reusable components from previous psets)
  - Identify the components that were reused
  - Identify any underlying design decisions (i.e. right-hand-rule, how your rover retained knowledge of the black squares, etc.)

System Problem Test Report (Test Environment)

2.1 Software Items Under Test

This paragraph shall identify any software by name, number and version, as applicable <u>that was tested</u> and summarized in this report.

In other words…provide the name of the file that your team submits. If it has a version number, provide that as well.

System Problem Test Report (Test Environment)

2.2 Components in the Software Test Environment

This paragraph shall identify by name, number, and version, as applicable, the support software items (e.g., operating systems, compilers, communications software, related applications software, etc.) necessary to <u>perform</u> the testing activities on the software identified in paragraph 2.1.

For example, you need AdaGide / AdaMindstorms, a PC running Windows, etc. Think about what you would need to give someone to test your system

System Problem Test Report (Test Results)

3.1 Overall Assessment of the Software Tested

This paragraph shall:

(a) Provide an overall assessment of the software as demonstrated by the test results in this report

(b) Identify any remaining deficiencies, limitations, or constraints that were detected by the testing performed.

Basically, describe how the rover worked or didn’t…

System Problem Test Report (3.2 Detailed Test Results)

3.2.1 Rover Maneuver Test(s)

- ***This paragraph shall describe the tests performed and the challenges faced when checking the rover's ability to move forward, backward and turn.***

3.2.2 Sensor Data Test(s)

- ***This paragraph shall describe tests performed to check the rover's reading of the light sensor data.***

3.2.3 Mission Data Transmit Test(s)

- ***This paragraph shall describe the tests used to check that the mission data could be transmitted to "earth.".***

System Problem Test Report (3.2 Detailed Test Results)

3.2.4 System Test(s)

- *Summarize what your rover did the first time you tried it on the 5x5 grid. Did you have to correct your algorithm after the first test? How did you learn from the rover's behavior?*
- *How did your rover perform during the final demonstration for credit? What was its path across the grid? Include a copy of your team's output from the rover during the demo.*

Here is a good place to indicate if your team performed any design or code inspections (i.e. paper-and-pen testing of your algorithm) prior to actual testing on the grid.

Demo Logistics

- Your team will sign up for a demonstration time slot
- C&P Staff will assemble a Martian surface for your lego rover
- Your rover will be loaded with the software turned in on 17 November
- The rover will be placed on square 1 facing east
- The rover will drive to square 25, avoiding black squares as required, then stop. Meanwhile, the C&P team will observe the rover and note the path the rover took across the Martian surface.
- The rover will then transmit its mission data to the PC
- The C&P team will retrieve this mission data file and provide it to the team
- From this data file, the team should be able to determine the path through the maze.
- Each person should get a copy of the mission data file for incorporation in his or her individual test report

Closing out the Design Phase

- We will grade the team rover design by tomorrow (Friday) afternoon
- You will have feedback before the weekend on your rover software design
- There are multiple ways to solve this rover problem.
  - We have implemented one solution that will be posted once we are completely done with the system problem.

Questions?