TAing Courses with Computer-Intensive Assignments

Dr. Sarah Heckman
Department of Computer Science
heckman@csc.ncsu.edu

Outline

• Computer-Intensive Assignments
  – Helping Students
• Grading Computer-Intensive Assignments
  – Test case generation strategies
  – Automated grading
  – Detection of inappropriate collaboration
Computer-Intensive Assignments

• What are some examples of computer-intensive assignments in your discipline?
  – Typical problems?
  – Medium of solution (program, Excel, Matlab, etc.)?
  – Length of solution?
  – Open endedness of solution (is there one solution per problem or is the solution a generic way to solve for a single solution with varying inputs?)
Helping Students

- Now that you’ve had the opportunity to hold some office hours and help some students,
  - What types of questions are you observing related to computer-intensive assignments?
  - How do students request help?
  - Are the problems related to the technology or the understanding of the problem (or both)?

Strategies for Helping Students

- Helping students with understanding and solving computer-intensive assignments is similar to helping students with other assignments
  - Encourage students to focus on a specific problem
    - Avoid “Is this right?” situations
  - Have students run a test case on their assignment that uncovers a flaw in their code
  - Guide students through solutions, with them doing the typing & verbalizing their thinking
    - Don’t touch the student’s keyboard!
    - Don’t just tell them the solution
Helping Students (Technology)

- Computer-intensive assignments do add another layer of complexity
  - The technology itself!!

- You may find
  - Students can’t install the tool correctly
  - Students don’t know how to use the tool
  - Students don’t know how to get started
    - Especially if the resources are scarce or the tool isn’t used in class

Strategies for Helping Students with Technology

- Where possible, and within your ability, try to troubleshoot installation problems
  - In the end, you’re not IT

- ITECS provides support for student-owned laptops for COE
  - ITECS can help with installing licensed software.
  - Email: eoshelp@ncsu.edu
  - Help Desk in 204 Daniels

- Recommend students use campus resources
  - VCL or EOS labs
Strategies for Helping Students with Technology

- When students don’t know how to use the tools for a class, it can be frustrating for everyone
  - If there’s an expectation that students should have the knowledge from previous classes, recommend resources that students could use as a reference
  - If the tool is new to the class, suggest creating tutorials or exercises that can introduce students to the tools in small chunks
    - Freshman/Sophomore Level: http://courses.ncsu.edu/csc216/common/tutorials/

Strategies for Helping Students with Technology

- Walk students through using the tool for one problem
  - Use the chain technique to have one student help another with the focus on using the tool
What about DE Students?

- Strategies for helping DE students with computer-intensive assignments?
  - Skype
  - Screen sharing solutions
    - There are VCL instances to facilitate that (or VCL instances can be created to use Virtual Network Computing (VNC) tools)
  - Screenshots
  - Executing student code with student tests that demonstrate the problems

Outline

- Computer-Intensive Assignments
  - Helping Students
- Grading Computer-Intensive Assignments
  - Test case generation strategies
  - Automated grading
  - Detection of inappropriate collaboration
Testing Introduction

- Testing is the process of finding software faults
  - Fault: “an incorrect step, process, or data definition in a program”
- Testing: “the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior”
- Test cases uncover failure by finding where the actual behavior of a program deviates from the expected behavior.
- Investigation of failures uncovering faults.

Black Box Testing

- Ignores the internals of the program – program treated as a black box
- Finds
  - Incorrect or missing function,
  - Interface errors,
  - Errors in data structures or external data base access,
  - Behavior or performance errors, and
  - Initialization and termination errors.
White Box Testing

• Internals of solution is known!
• Can use internal information to guide test
• Your tests should exercise
  – Independent paths within the source code
    (“important” paths)
  – Logical decisions as both true and false
  – Loops at their boundaries
  – Internal data structures

Test Case Information

• **Unique Identifier**
• **Input** into the program or program unit
  – Black box: how the user runs and interacts with the program
  – White box: inputs to methods that set up test
• **Expected output** from the program or program unit
  – What you expect to get based on input and requirements
• **Actual results** of running the test case
  – Black box: what the user gets from the program
  – White box: return values from functions or check on state via other functions
• **Key**: Repeatable and Specific
  – If you were to provide your tests to the students when you return the homework grades, would students see the same results?
Test Plan

- Formal document outlining the test cases for a project
- Description must be repeatable – have specific values!
- Expected results require specific values too!
- Write tests **before** posting the assignment if possible
- Adjust tests based on discussions that occur as part of completing the assignment
- If possible, provide (a subset) to students as part of the assignment
- If possible, provide all tests to students when returning assignment

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestName</td>
<td>Preconditions: Test Inputs</td>
<td>Test outputs</td>
<td>Actual outputs</td>
</tr>
</tbody>
</table>

StringAnalyzer Problem

*A client wants to collect information about the types of characters users use in a String input to a program. Specifically, the client wants to collect the number of times a digit between a minimum and maximum value, inclusive, is used in String input. The user specifies the string input, a minimum, and a maximum value. If the minimum and maximum values are not valid digits or if the maximum is less than the minimum, the error message, “Invalid bounds” is displayed to the user. Otherwise, the count of digits between the minimum and maximum values, inclusive, is displayed to the user in the following output: “The String contains X digits between minimum and maximum, inclusive.” The program repeats until the user enters the string “quit”.*
Test Requirements

- Ensure that all of the customer requirements are tested!

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>validInput</td>
<td>Preconditions: StringAnalyzer program started</td>
<td>The String contains 2 digits between 1 and 7, inclusive.</td>
<td></td>
</tr>
<tr>
<td>(Heckman)</td>
<td>String? A string with 3 numbers 4 9.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Minimum? 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maximum? 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>String? quit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>noInput</td>
<td>Preconditions: StringAnalyzer program started and note that the string input</td>
<td>The String contains 0 digits between 1 and 7, inclusive.</td>
<td></td>
</tr>
<tr>
<td>(Heckman)</td>
<td>for the first prompt is just pressing enter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>String?</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Minimum? 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maximum? 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>String? quit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Equivalence Classes

- Input space is broken into different classes
- Each equivalence class is tested
- Tests are written to include “middle” input values from each of the possible classes

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>noInput</td>
<td>Preconditions: StringAnalyzer program started and note that the string input</td>
<td>The String contains 0 digits between 1 and 7, inclusive.</td>
<td></td>
</tr>
<tr>
<td>(Heckman)</td>
<td>for the first prompt is just pressing enter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>String?</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Minimum? 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maximum? 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>String? quit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Boundary Value Analysis

- Programmers tend to make mistakes at boundaries
- Want to test program boundaries and values to either side of the boundary

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeroMinBound (Heckman)</td>
<td>Preconditions: StringAnalyzer program started</td>
<td>The String contains 3 digits between 0 and 7, inclusive.</td>
<td></td>
</tr>
<tr>
<td>Test Type: BVA</td>
<td>String? String 345 String. Minimum? 0 Maximum? 7 String? quit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diabolical or Dirty Test Cases

- Divide by zero?
- Wrong input type? String instead of a double
- Illogical path through functionality
- Don’t enter mandatory fields?
- Program aborted or device unplugged, dropped (do NOT do with my phones!), turned off?
Black Box Test Cases

- The input to a text field must be an integer between 5 and 10, inclusive. If the integer is outside the range, then an error message is displayed.

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Writing White Box Test Cases

- Focus on methods or small units of code – how can we test?
  - Automation!
- Exercise all paths or standard inputs/outputs from the small units
  - Create control flow diagram
  - Inputs for basis set of paths
  - Inputs for equivalence classes and boundary values
  - Inputs for diabolical tests
Automating Testing

- **Black Box Tests**
  - Automate with test scripts, headless runs, etc.
- **White Box Tests**
  - Automate with unit testing frameworks, test scripts, headless runs, etc.

Automating Black Box Testing

- Use scripts to run programs
  - Python, Perl, Bash, etc.
- One way to automate input and output is using redirection and diffs
  
  ```
  % ./program < input >! actual_output
  % diff expected_output actual_output
  ```
  - Example:
    http://courses.ncsu.edu/csc230/common/homeworks/2_homework/2_homework.html
- Run application in a headless manner (without GUI)
  - Command line arguments for the application (MATLAB, Eclipse, etc.)
  - Run scripting language behind the application (Excel using VB Scripts)
JUnit for White Box Testing

- JUnit is an API for automating white box testing
- Lots of other programming languages have xUnit testing frameworks
    - C
    - C++
    - MATLAB
    - Fortran
    - Many more...

Assert Methods

- Assert methods provide information about the expected and actual values of a test case
  - assertEquals(expected, actual);
    - For doubles, you will have a third argument, delta
  - assertTrue(actual);
  - assertFalse(actual);
  - assertNotNull(actual);
  - assertNull(actual);
  - assertNotSame(actual);
Brain Storming

• What automated testing techniques are you going to investigate?
  – Do a quick Google search for resources that may help you.
  – What would you need to learn to be successful?

Automating Feedback

• Excel spreadsheets and VB Scripts
  – Each grade item is a column
  – Each student is a row
  – The VB script will generate a text feedback file for each student
  – Return files to students via
    • Moodle – individually
    • Wolfware Classic – bulk
  – DEMO!!
  – Email me if you want a copy of the file
Detection of Inappropriate Collaboration

• Intuition should always be investigated
• Electronic submissions can have a more objective analysis
• MOSS
  – C, C++, Java, Python, FORTRAN, MATLAB, and more
  – Requires registration and an id to run
  – Linux is required to run
    • I find it easiest to run as a script on AFS when storing the student submission in the class’ locker in lec/X0X/submitted directory.
  – Returns a URL – send to instructor

Detection of Inappropriate Collaboration

• Look at electronic file properties
  – Creation time
  – Author information
  – Last Saved time
Brain Storming

• What strategies are you planning on using?
• What resources will you use?
• What information do you need to be successful?

Resources – University

• ITECS: http://www.itecs.ncsu.edu/
• VCL: http://vcl.ncsu.edu/
• EOS Labs: http://www.eos.ncsu.edu/labs
  – Need to have a server running and viewers connect to server
Resources - Testing

- Unit testing frameworks:
  http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks
- Remote Resources for testing:
  http://www.eos.ncsu.edu/remoteaccess/
- E115 Online Text Book:
  http://www.eos.ncsu.edu/e115/text.php
  – Chapters 2 and 3 for Linux command line
- Testing Overview:

Resources – Plagiarism Detection

- MOSS: http://theory.stanford.edu/~aiken/moss/
References

• L. Williams, CSC326 Slides