TAing Courses with Computer-Intensive Assignments

Workshop, September 11, 2015

Dr. Lina Battestilli
• Introductions

• Types of Computer-Intensive Assignments

• Strategies for Helping Students

• Grading –
  – Testing
  – Automation of Testing
  – Rubrics
  – Feedback

• Detection of Inappropriate collaboration
Dr. Lina Battestilli
Teaching Assistant Professor
Department of Computer Science
lbattestilli@ncsu.edu

• Graduate of NCSU, PhD in 2005
• Expertise in Networking and Cloud Computing
• Worked at IBM Research in the Next Generation Computing Group
• Faculty since 2012
Who are You?

Workshop Participants by Major

- CSC: 28
- MAE: 7
- ISE: 6
- CBE: 7
- ECE: 15
- CE: 3
Meet each other – groups of 2

- Name?
- Where are you from?
- Academic Level
- Major
- What class are you TAing?
  - How many students?
  - Do you work with other Tas?
Workshop Website

Slides, Links, Resources from this workshop can be found at:

http://go.ncsu.edu/ta_computer_courses
Outline

• Introductions

• Types of Computer-Intensive Assignments

• Strategies for Helping Students

• Grading –
  – Testing
  – Automation of Testing
  – Rubrics
  – Feedback

• Detection of Inappropriate collaboration
What are some examples of computer-intensive assignments in your discipline?

- Typical problems?
- Medium of solution (program, Excel, Matlab, etc.)?
- Length of solution?
- How Open ended is the solution?
  - is there one solution per problem or is the solution a generic way to solve for a single solution with varying inputs?
Types of Assignments

- JAVA Programs
- C Programs
- Matlab Programs
- Python Programs
- Assembly
What are the Assignments in the course you are a TA?

We will use a Google Form to gather your answers

http://www.google.com/forms

Fill out this Google Form:
http://goo.gl/forms/vwuivcSc6w
Outline

• Introductions
• Types of Computer-Intensive Assignments
• Strategies for Helping Students
• Grading –
  – Testing
  – Automation of Testing
  – Rubrics
  – Feedback
• Detection of Inappropriate collaboration
Think/Pair/Share

Helping 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
  - Answer specific questions, not generic “I don’t understand.” statements.
- 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!
• 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, create tutorials or exercises that can introduce students to the tools in small chunks
- Online video tutorials

Freshman/Sophomore Level:
http://courses.ncsu.edu/csc216/common/tutorials/

Junior Level:

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

• Peer Tutoring

• Pair Programming/Collaboration
What about DE Students?

- Video conferencing - Google Hangout
- Screen sharing solutions
  - There are VCL instances to facilitate that
  - Virtual Network Computing (VNC) tools
- Online Forums
- Emailing Screenshots
- Executing student code with student tests that demonstrate the problems
Outline

• Introductions
• Types of Computer-Intensive Assignments
• Strategies for Helping Students
• Grading –
  – Testing
    • Black box
    • White box
    – Automation of Testing
    – Feedback
    – Rubrics
• Detection of Inappropriate collaboration
A programmer makes a mistake.

The mistake manifests itself as a fault or defect in the program.

A failure is observed if the fault [or defect] is made visible. Other faults remain latent in the code until they are observed (if ever).
## Black Box and White Box Testing

<table>
<thead>
<tr>
<th>Type of Testing</th>
<th>White-box Testing</th>
<th>Black-box Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tester visibility</td>
<td>have visibility to the code and write test cases based upon the code</td>
<td>have no visibility to the code and write test cases based on possible inputs and outputs for functionality documented in specifications and/or requirements</td>
</tr>
<tr>
<td>A failed test case reveals Controlled?</td>
<td>a problem (fault)</td>
<td>a symptom of a problem (a failure)</td>
</tr>
<tr>
<td></td>
<td>Yes – the test case helps to identify the specific lines of code involved</td>
<td>No – it can be hard to find the cause of the failure</td>
</tr>
</tbody>
</table>

© Laurie Williams 2006
Test Cases

Comparing **expected** and **actual** output based on **specific input** to the program

Test case: the specific inputs under which a tester will be able to determine whether there are any faults with the program.
Writing Test Cases

Black Box Testing

- how the user runs and interacts with the program
- what the user gets from the program

White Box Testing

- inputs to methods that set up test
- 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?
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
There are four seasons that correspond to the given general dates for the northern hemisphere:

- **Winter:** December 21 to March 20
- **Spring:** March 21 to June 20
- **Summer:** June 21 to September 20
- **Fall:** September 21 to December 20

A customer wants a program that prompts the user a month and a day.

- **If the month and day are valid, the program will print the season for the month and day.**
- **If the month isn’t a match to full name of one of the calendar months, then the error message “Invalid month.” is printed and the user will be reprompted for the month until they enter a correct month.**
- **If the day is not valid for the given month (e.g., 30 in February), the error message “Invalid day.” is printed and the user will be reprompted for the day until they enter a correct day.** For this program, we will assume that we are not in a leap year, so February 29th would be considered invalid.

**Example Output:**

```
Enter a month: february
Enter a day: 14
Winter
```
```
Enter a month: 7
Invalid month.
Enter a month: July
Enter a day: 90
Invalid day.
Enter a day: 29
Summer
```
Outline

• Introductions
• Types of Computer-Intensive Assignments
• Strategies for Helping Students
• Grading –
  – Testing
    • Black box
    • White box
      – Automation of Testing
      – Feedback
      – Rubrics
• Detection of Inappropriate collaboration
Black Box Testing

- Ignores the internals of the program – program treated as black box
- Typically, black box tests are run by an unbiased third-party; not the programmer who developed the code.

Black Box Test Case

- Formal document outlining the black box test cases
- Tests must be repeatable, i.e. have specific input values
- Expected results require specific output
- Write the black box test plan before writing your program
Black Box Test Plan Template

- Formal document outlining the black box test cases
- Description must be **repeatable** – have **specific input values**
- Expected results require **specific output**
- Write the black box test plan **before** writing your program

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestName1</td>
<td>Preconditions: Test Inputs</td>
<td>Test outputs</td>
<td>Actual outputs</td>
</tr>
<tr>
<td>TestName2</td>
<td>Preconditions: Test Inputs</td>
<td>Test outputs</td>
<td>Actual outputs</td>
</tr>
<tr>
<td>TestName3</td>
<td>Preconditions: Test Inputs</td>
<td>Test outputs</td>
<td>Actual outputs</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Types of Black Box Tests

- **Test Requirements**
  - test the requirements

- **Equivalence Class Partitioning Tests**
  - limited amount of time to test and an infinite number of possible test cases.
  - Focus on test cases that will uncover new errors, do not test cases that are essentially equivalent to other test cases

- **Boundary Value Analysis Tests**
  - Programmers tend to make mistakes at the boundaries of the equivalence classes
Test Requirements

• Ensure that all of the customer requirements are tested!
• Example: Users should be given an error message if invalid month.

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalidMonth</td>
<td><strong>Preconditions:</strong> Seasons program started</td>
<td>Invalid month. Enter a month:</td>
<td>Program stops execution</td>
</tr>
<tr>
<td></td>
<td>Enter a month: <strong>Invalid</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Stop program</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Determine the Equivalent Classes

For the Seasons example:

<table>
<thead>
<tr>
<th>Equivalent Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Month</td>
</tr>
<tr>
<td>2. Day</td>
</tr>
<tr>
<td>3. Season</td>
</tr>
</tbody>
</table>

Each equivalence class has to be tested!!!

- Tests are written to include “middle” input values from each of the possible classes
- One test may consider multiple equivalence classes
  - One for each type of input/output
  - A test focuses on one equivalence class, but other values are needed for a full test. Those other values should be “middle” values.
- Helps further test requirements by considering groups of inputs/outputs
# Equivalent Classes – Month

<table>
<thead>
<tr>
<th>Month (Input)</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>January</td>
</tr>
<tr>
<td>Invalid</td>
<td>Not a month</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>winter</td>
<td><strong>Preconditions:</strong> Seasons program started</td>
<td>Winter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enter a month: <strong>January</strong></td>
<td>Program stops execution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enter a day: <strong>15</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>invalidMonth</td>
<td><strong>Preconditions:</strong> Seasons program started</td>
<td>Invalid month.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enter a month: <strong>Invalid</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Stop program</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Enter a month:</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Equivalent Classes – Day

<table>
<thead>
<tr>
<th>Day (Input)</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 1</td>
<td>-3</td>
</tr>
<tr>
<td>1 ≤ day ≤ max</td>
<td>15</td>
</tr>
<tr>
<td>&gt; Max</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
<th>Expected Results</th>
<th>Actual Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>winter</td>
<td><strong>Preconditions:</strong> Seasons program started</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enter a month: <strong>January</strong></td>
<td>Winter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enter a day: <strong>15</strong></td>
<td>Program stops execution</td>
<td></td>
</tr>
<tr>
<td>negativeDay</td>
<td><strong>Preconditions:</strong> Seasons program started</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enter a month: <strong>January</strong></td>
<td>Invalid day.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enter a day: -3</td>
<td>Enter a day:</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Stop program</strong></td>
<td>Program stops execution</td>
<td></td>
</tr>
</tbody>
</table>
### Boundary Value Analysis Test Cases

- 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>zeroDays</td>
<td><strong>Preconditions:</strong> Seasons program started Enter a month: January Enter a day: 0 <strong>Stop program</strong></td>
<td>Invalid day. Enter a day: Program stops execution</td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Introductions
• Types of Computer-Intensive Assignments
• Strategies for Helping Students
• Grading –
  – Testing
    • Black box
    • White box
  – Automation of Testing
  – Feedback
  – Rubrics
• Detection of Inappropriate collaboration
Writing White Box Test Cases

Code is known! Can use the code to guide writing of the test.

- The test class exercises all of your program’s methods (except main)
- You’ll want to move most of your functionality out of main method so you can test it
public class SeasonsTest {

    /** Months of the year. */
    private static final String[] MONTHS = {
            "August", "September", "October", "November", "December"};

    /** Max days in each month. */
    private static final int[] DAYS_IN_MONTHS = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    /**
     * Tests Seasons.isValidDayForMonth().
     */
    public static void testIsValidDayForMonth() {
        // Test middle day - equivalence class on middle day value
        String id = "Day Fifteen";
        String desc = "Seasons.isValidDayForMonth("January", 15)";
        String expected = "true";
        String actual = "" + Seasons.isValidDayForMonth("January", 15);
        testResult(id, desc, expected, actual);

        // Test negative day - equivalence class on day values less than 1
        id = "Negative Day";
        desc = "Seasons.isValidDayForMonth("January", -3)";
        expected = "false";
        actual = "" + Seasons.isValidDayForMonth("January", -3);
        testResult(id, desc, expected, actual);
    }

    // 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
Outline

- Introductions
- Types of Computer-Intensive Assignments
- Strategies for Helping Students
- Grading –
  - Testing
  - Automation of Testing
  - Feedback
  - Rubrics
- Detection of Inappropriate collaboration
Automating Black Box Testing

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 is linked on the workshop website

- 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)
Automating White Box Testing

White Box Testing

Automate with unit testing frameworks, test scripts, headless runs, etc.

Some Example frameworks:

- **JUnit** automates white box testing in JAVA
- Cody Coursework is for MATLAB

Lots of other programming languages have **xUnit** testing frameworks


- C
- C++
- MATLAB
- Fortran
- Many more...
The electricity accounts of residents in a very small town are calculated as follows, where units refers to units of electricity:

- If 500 units or fewer are used, the cost is 2 cents per unit.
- If more than 500, but less than 1000 units are used, the cost is $10 for the first 500 units and 5 cents for every unit in excess of 500.
- If 1000 units or more are used, the cost is $35 for the first 1000 units plus 10 cents for every unit in excess of 1000.
- A basic service fee of $5 is charged, no matter how much electricity is used.

Using if and elseif statements, for number of units \( n \), calculate the total cost. Store this answer in the variable \( cost \).

Solution

```matlab
if n==1
    cost = 0;
elseif
    cost = 0;
else
    cost = 0;
end
```
Automation

"I spend a lot of time on this task. I should write a program automating it!"

**Theory:**

- **Work:**
  - Writing code

- **Work on original task:**
  - Automation takes over

- **Free time:**

**Reality:**

- **Work:**
  - Writing code

- **Debugging:**
  - Ongoing development

- **Rethinking:**
  - No time for original task anymore

https://xkcd.com/1319/
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?
- Get with the TAs from your major
  - groups 3-4
Outline

• Introductions

• Types of Computer-Intensive Assignments

• Strategies for Helping Students

• Grading –
  – Testing
  – Automation of Testing
  – Rubrics
  – Feedback

• Detection of Inappropriate collaboration
<table>
<thead>
<tr>
<th>Value</th>
<th>Earned</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Olympics program</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+5 points</td>
<td>+5</td>
<td>Correct style.</td>
</tr>
<tr>
<td>+5 points</td>
<td>+5</td>
<td>Program is correctly named, compiles, and runs.</td>
</tr>
<tr>
<td>+5 points</td>
<td>+5</td>
<td>File input is handled correctly.</td>
</tr>
<tr>
<td>+5 points</td>
<td>+5</td>
<td>Program displays an appropriate menu.</td>
</tr>
<tr>
<td>+5 points</td>
<td>+5</td>
<td>User selection for the main menu works correctly.</td>
</tr>
<tr>
<td>+10 points</td>
<td>+10</td>
<td>List Total Medals by Country correctly implemented.</td>
</tr>
<tr>
<td>+10 points</td>
<td>+10</td>
<td>List Athletes by Age and Year correctly implemented.</td>
</tr>
<tr>
<td>+10 points</td>
<td>+10</td>
<td>Search by Athlete Name correctly implemented.</td>
</tr>
<tr>
<td>+10 points</td>
<td>+10</td>
<td>Search by Sport and Year correctly implemented.</td>
</tr>
<tr>
<td>MagicSquare class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+5 points</td>
<td>+5</td>
<td>MagicSquare is correctly named, compiles, and runs.</td>
</tr>
<tr>
<td>+15 points</td>
<td>+15</td>
<td>MagicSquare class passes teaching staff test cases.</td>
</tr>
<tr>
<td>+5 points</td>
<td>+5</td>
<td>isMagicSquare method correctly implemented.</td>
</tr>
<tr>
<td>+5 points</td>
<td>+5</td>
<td>MagicSquareTest is correctly named, compiles, and runs.</td>
</tr>
<tr>
<td>+5 points</td>
<td>+5</td>
<td>MagicSquareTest contains 7 test cases and all pass.</td>
</tr>
<tr>
<td>Javadoc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+2 points</td>
<td>+2</td>
<td>Correct syntax and placement.</td>
</tr>
<tr>
<td>+2 points</td>
<td>+2</td>
<td>Class comments fully describe the program’s functionality.</td>
</tr>
<tr>
<td>+2 points</td>
<td>+2</td>
<td>Method comments fully describe behavior.</td>
</tr>
<tr>
<td>+2 points</td>
<td>+2</td>
<td>Method comments fully describe inputs (parameters), outputs (return) information, and exception.</td>
</tr>
<tr>
<td>+1 points</td>
<td>+1</td>
<td>No spelling errors.</td>
</tr>
<tr>
<td>+1 points</td>
<td>+1</td>
<td>No grammatical errors.</td>
</tr>
</tbody>
</table>

Penalties:
-10 points | | Late electronic submission. |
-5 points | -5 | Hardcopy turned in late, not stapled, or gradesheet improperly filled out. |

Total Earned
+110 points | | |
### Olympics Program

**5 points - Correct style**
- Inspect the code
- 1 point for each magic number in the program.
- Note the line where you observe the infraction.
- Do NOT take off points for -1, 0, 1, 2.
- 1 point for each incorrect or inconsistent style problem (indentation, incorrect naming conventions, etc.).
- Note the lines where you observe the infraction.

**5 points - Program is correctly named, compiles, and runs**
- 5 points - program doesn't compile
  - If the program compiles:
    - 3 points - program not named correctly
    - 2 points - program doesn't run

**NOTE:** If you can easily fix their program, do so, and use the corrected program for grading.

---

<table>
<thead>
<tr>
<th>Value Earned</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+5 points</td>
<td>Correct style.</td>
</tr>
<tr>
<td>+5 points</td>
<td>Program is correctly named, compiles, and runs.</td>
</tr>
<tr>
<td>+5 points</td>
<td>File input is handled correctly.</td>
</tr>
<tr>
<td>+5 points</td>
<td>Program displays an appropriate menu.</td>
</tr>
<tr>
<td>+5 points</td>
<td>User selection for the main menu works correctly</td>
</tr>
<tr>
<td>+10 points</td>
<td>List Total Medals by Country correctly implemented.</td>
</tr>
<tr>
<td>+10 points</td>
<td>List Athletes by Age and Year correctly implemented.</td>
</tr>
<tr>
<td>+10 points</td>
<td>Search by Athlete Name correctly implemented.</td>
</tr>
<tr>
<td>+10 points</td>
<td>Search by Sport and Year correctly implemented.</td>
</tr>
</tbody>
</table>

The TA view is much more detailed!

---

created by Drs. Suzanne Balik and Jessica Schmidt, csc116
Detailed Grading Rubrics – TA View

TEST 2

%java Olympics

Olympics (2000-2012) - Please enter an option below.

C - List Total Medals by Country
A - List Athletes by Age and Year
N - Search by Athlete Name
S - Search by Sport and Year
Q - Quit the program

Option: C

Canada, 6 medal(s)
Great Britain, 1 medal(s)
Russia, 8 medal(s)
South Korea, 5 medal(s)
Ukraine, 1 medal(s)
United States, 8 medal(s)

Olympics (2000-2012) - Please enter an option below.

C - List Total Medals by Country
A - List Athletes by Age and Year
N - Search by Athlete Name
S - Search by Sport and Year
Q - Quit the program

Option: q

Goodbye!

Evaluate the acceptance of user input in Test 2 (above)
+1 points if the input of 'C' was accepted as valid
(no error message and printed out a list - doesn’t have to be correct list)

+1 point if the input of 'q' was accepted, goodbye printed and then quit
(if menu not redisplayed after option 'C' completed, run again to test the lower case 'q' input)

How many points to deduct if it test is not passed

Test to be done by the TA

created by Drs. Suzanne Balik and Jessica Schmidt, csc116
Outline

- Introductions
- Types of Computer-Intensive Assignments
- Strategies for Helping Students
- Grading –
  - Testing
  - Automation of Testing
  - Rubrics
  - Feedback
- Detection of Inappropriate collaboration
Automating Grade Feedback

Excel Spreadsheet

- Each grade item is a column
- Each student is a row

Java Program

Generates a formatted PDF with feedback and grade for each student

Return files to students via
- Moodle – individually
- Wolfware Classic – bulk

DEMO!!
Download Java Program from this Workshop’s Website
Excel Spreadsheet

JAVA Program in Eclipse

PDF Generated

DEMO
Automating Grade Feedback

Excel Spreadsheet

- Each grade item is a column
- Each student is a row

The VB script will generate a text feedback file for each student

Generates a TEXT document with feedback and grade for each student

Return TEXT files to students via
- Moodle – individually
- Wolfware Classic – bulk

Download Excel File with VB Script from this Workshop’s Website
Outline

• Introductions
• Types of Computer-Intensive Assignments
• Strategies for Helping Students
• Grading –
  – Testing
  – Automation of Testing
  – Rubrics
  – Feedback

• Detection of Inappropriate collaboration
Can the students collaborate (individual work or in teams?)

Besides the textbook, course material and what about online reference, what sites are OK?

Examples of Cheating:

• It is cheating to give any student access to any of your work which you have completed for individual class assignments.
• It is cheating AND plagiarism to use another person’s work and claim it as your own. You are expected to complete all assignments on your own, unless otherwise specified in the assignment.
• It is cheating to interfere with another student’s use of computing resources or to circumvent system security.
• It is cheating to email, ftp, post on the Internet, bulletin boards, etc. your work for others to obtain. Do not use sites that allow you to “anonymously” post code. Those sites are searchable, and others may find your code.
• It is cheating to ask or pay another person or persons to complete an assignment for you.
• It is cheating AND plagiarism to use code that you find online.
• It is cheating to give another student access to your account (NC State account or others that you use for university work) or to give them your account password.
• It is cheating for you and another student to work collaboratively on an assignment, unless otherwise specified by the assignment.
Examples of NOT Cheating:
• Using code from the class website (with citations in the comments).
• Using code from other programs YOU wrote.
• Using code from other programs that YOU and a partner wrote as part of assigned exercises.
• Help from the TAs, or Instructor (with citations in the comments).

Protecting Yourself:
• Do not leave papers lying around your workstation
• Do not dispose of important papers in the lab recycling bins and trash cans until after the assignment is graded.
• Do not give out your password.
• Do not leave your workstation unattended or forget to log yourself out.
• Do not give other students access to any of your workspace or email them any code.
• Do not give other students access to your course materials on your personal computer.
• Do not email, ftp, or post your code on the Internet, bulletin boards, etc.
• Keep all copies of final and intermediate work until after assignment is graded.
• Keep graded assignments until after you receive the final grade for the course.

These guidelines could be added to the course syllabus!
Detection of Inappropriate Collaboration

- Intuition should always be investigated
- Look at electronic file properties
  - Creation time
  - Author information
  - Last Saved time
- Electronic submissions can have a more objective analysis
MOSS – system to detect Plagiarism

https://theory.stanford.edu/~aiken/moss/

- C, C++, Java, Python, FORTRAN, MATLAB, etc.
- Requires registration and an id to run
- Linux is required to run
  - easiest to run as a script on AFS when storing the student submission in the class’ locker in lec/X0X/submitted directory.
- An HTML report is sent to instructor
Brain Storming

• What strategies are you planning on using?

• What resources will you use?

• What do you plan to research and learn more about?
Resources – University

- ITECS: [http://www.itecs.ncsu.edu/](http://www.itecs.ncsu.edu/)
- VCL: [http://vcl.ncsu.edu/](http://vcl.ncsu.edu/)
- EOS Labs: [http://www.eos.ncsu.edu/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
Resources – Plagiarism Detection

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

• Special thanks to Sarah Heckman who shared her materials from leading the “TAing Courses with Computer-Intensive Assignments Workshop” in 2014

• Suzanne Balik, Jessica Schmidt, Grading Rubrics for csc116


• L. Williams, CSC326 Slides