Teaching a University-wide Programming Laboratory

Managing a C Programming Laboratory for a Large Class with Diverse Interests

Vishv Malhotra and Ashish Anand
Indian Institute of Technology, Guwahati
Assam, India
Introduction – continues

• There are now 23 IITs.
• Five of them are called original – Pre 1963
• Guwahati is 6\textsuperscript{th} – Mid 1994
• Rest are called New! – Post 2008
• Some have been converted from existing universities – c. 1847, 1919, 1926)
Motivation for the work

• Given the population of India (~1.3 Billion) that provides students for these IITs (~12000 places) competition is very strong. (Applicants ≈ 130,000)
• Yet at IIT Guwahati in 2017 we had over 130 failing students in a class of ~700 in Programming Laboratory course.
• This large failure rate does not sit well with the admission processes!
• We suggest that it is due to undue haste in teaching these courses.

• Can we train students to construct programs in a way that accommodates those who need time as well as those who are quick programmers?
Agenda for Today’s Presentation

• What causes this disappointing pass-rate?
• What we did to address this cause?
• How we did it?
• What were the outcomes?
• What new concerns this caused?
• Seek your comments and questions.
What may be the Cause of Low Progression-rate?

• The reason for disengagement among the students is (an analogy)
  • We train the students to ride push bike for some time.
    • Some learn to ride; others may need more time.
    • We ignore their need for more time.
  • We take all of them to Motorised bike training!
    • Broken bones are our fault!
• Students who do not match the pace drop out!
  • Solution: Let each student progress at their own pace.
Solution: Break Contents into Useful Parts – Modules

• Create useful modules – Each useful module is
  • Made of easily-identified contents and programming skills
  • Examinable unit with meaningful target skills
  • A sequential skills-progression over the previous module

• Four Modules
  1. **Module 1**: Be able to write programs for computations mimicking a session with a calculator. Basic types – `int`, `float`.
  2. **Module 2**: A single function program – `main()` with flow controls
  3. **Module 3**: Programming abstractions – functions, parameters – by value and by reference (pointers), C defined types
  4. **Module 4**: Advanced topics – Pointers, user data-structures, files
Modules, Stages, Drills, Assessments, Examinations

- Module is an examinable unit giving a set final course grade
- Each module has a number of stages to support learning and training.
- Stages in a module are sequential units supporting training/learning
- Each stage has a drill manual and a set of assessment problems
**Solution – Modules are trained through stages**

<table>
<thead>
<tr>
<th>Module</th>
<th>Stage</th>
<th>Drill topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module 1</td>
<td>Stage 1</td>
<td>Basic UNIX commands</td>
</tr>
<tr>
<td></td>
<td>Stage 2</td>
<td>Imperative statements</td>
</tr>
<tr>
<td>Module 2</td>
<td>Stage 1</td>
<td>Numerical values and their input-output</td>
</tr>
<tr>
<td></td>
<td>Stage 2</td>
<td>Conditional control flow, <code>assert()</code></td>
</tr>
<tr>
<td></td>
<td>Stage 3</td>
<td>Loops, operators with side-effects</td>
</tr>
<tr>
<td>Module 3</td>
<td>Stage 1</td>
<td>Non-recursive functions</td>
</tr>
<tr>
<td></td>
<td>Stage 2</td>
<td>Arrays, <code>structs</code>, strings</td>
</tr>
<tr>
<td></td>
<td>Stage 3</td>
<td>Recursive functions, Call by reference</td>
</tr>
<tr>
<td>Module 4</td>
<td>Stage 1</td>
<td>Data structures (linked list, stack), object orientation, header (.h) files</td>
</tr>
<tr>
<td></td>
<td>Stage 2</td>
<td>Files and long-term data storage</td>
</tr>
<tr>
<td></td>
<td>Stage 3</td>
<td></td>
</tr>
</tbody>
</table>
Stage: Training Routine

- Stage: *Maximum* amount of work for a laboratory session
- Students complete a drill lesson for a stage with support from the tutor(s)
- Students demonstrate (stage) learning by completing a randomly chosen assessment problem.
- If a student cannot demonstrate learning, student stays at the stage till stage is learned.
- Module Examinations scheduled at the set dates – Module examination administered if at least 2 stages completed.
- Success at a module examination delivers the module specific grade
Training and Progression over Modules & Stages
Training Routine: Keller Plans

• Failure at a Module examination (Snake bite)
  • Repeat the module training

• Classical approach to teaching: *same pace, different learning*

• Keller plans: *different pace, same learning*

• Our approach: *different pace, different learning, different grades*
  • No more than one stage can be completed in a weekly laboratory session
Outcomes: Benefits of the Changed Arrangements

- Big reduction in the failed student count
- Shift from the lower grades to the middle grades
- Students have clear view of their options
Effect of the Changed Practice: Wrong Training Avoided

- Students at grades below CC have moved up. Because,
  - Better support and more time to learn basic topics.
  - Examinations/assessments appropriate to the preparedness.
- Students at the top grades see little change in their ways and achievements

<table>
<thead>
<tr>
<th>Grades</th>
<th>Percentage of class</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017</td>
<td>2018</td>
</tr>
<tr>
<td>F</td>
<td>0.0 19.7 37.1 49.2 60.1 70.4 81.6 89.8 99.6</td>
</tr>
<tr>
<td>2018</td>
<td>0.0 1.9 8.6 12.7 40.7 50.9 78.0 88.4 97.1</td>
</tr>
</tbody>
</table>
When the Students Completed each Module?

<table>
<thead>
<tr>
<th>Module</th>
<th>Mid-Sem Exams</th>
<th>End-Sem Exams</th>
<th>Total (%age)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Part 1</td>
<td>Part 2</td>
<td>Week 1</td>
</tr>
<tr>
<td>Module 1</td>
<td>682 (93.0%)</td>
<td>7 (0.9%)</td>
<td>28 (3.8%)</td>
</tr>
<tr>
<td>Module 2</td>
<td>474 (64.7%)</td>
<td>108 (14.7%)</td>
<td>62 (8.5%)</td>
</tr>
<tr>
<td>Module 3</td>
<td></td>
<td>176 (24.0%)</td>
<td>164 (22.4%)</td>
</tr>
<tr>
<td>Module 4</td>
<td></td>
<td></td>
<td>85 (11.6%)</td>
</tr>
</tbody>
</table>
Students’ Say

• It is a nice method for conducting lab classes. Drills and module concept is fare.

• The Grading system is very confusing to understand in the first go. Evaluation (in both labs sessions and exams) by the TAs was really biased for some lab groups.

• From a learning point of perspective, the course was simply flawless. Practicing 10-15 questions each week and some being very challenging at the first glance was really amazing and I really learned a lot. So, I guess pre-disclosure of the assessments for the lab should be continued.
New Concerns: Plagiarism and Cheating?
Cheating Risk and its Epicenter
Or Did They Over-practice?

Cheating Risk and Epicenter

<table>
<thead>
<tr>
<th></th>
<th>AS/AH</th>
<th>AB</th>
<th>BB</th>
<th>BC</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>7%</td>
<td>13%</td>
<td>55%</td>
<td>25%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Risk Spread

- AS/AH: 13%
- AB: 7%
- BB: 25%
- BC: 55%
- CC: 0%
Summary: What has been Achieved

- Clear identification of the modules and module contents sets clear plan to define training needs.
- Assessment and training processes are better understood and managed.
- Students had clear idea of where they stand and had good control over their time utilisation across their courses.
- Unexpected good benefits to the success-rate
  - Included challenging skills (Backtracking) during the semester.

Motivated students were not affected by those who seek more time. And, the faster students not hurry those needing time and support.
Thanks for your presence and attention

I welcome your comments and

would like to answer questions


A Drill lesson and a related set of assessment exercises are here: Training Lessons for Minimum Pass Standards in a University-wide Undergraduate C Programming Laboratory. https://doi.org/10.13140/rg.2.2.22673.89441
Typical Methods Used to Cheat – Sorry Skipping 😊

• Smuggled solutions
  • Assessment exercises were available for the students to practice.

• Unclear boundary to define cheating!
  • Bringing paper or electronic copy is wrong
  • Is memorising a solution cheating or not?
    • It is definitely wrong if the solution was created by a different person.
  • If the student prepared the solution and memorised?
    • Do not have a firm opinion.
Another Way Used to Cheat

- `printf()` correct answer
- Kept a compiled version ready to demonstrate to the tutor at the end when there is time pressure.
- Tutors failed to verify *Does the program run correctly* check.

<table>
<thead>
<tr>
<th></th>
<th>Student</th>
<th>Tutor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is the program appropriately commented? And, do the comments help in understanding the program code?</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Is the amount of comments in the program appropriate? That is, the amount of comments is neither too little nor too much.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Is the name of the programmer and date of creation included in the demonstrated program?</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Are all constants included in the program code from the exercise statement? The program output should only be computed from the program inputs and constants listed in the problem description.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Are the identifiers used as variables helpful and describe the variable use correctly? All pointer variables must have an easily understandable part in the variable name to distinguish it from non-pointer variables.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Are the variable type declarations appropriate?</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Is the program correctly indented and it is easy to read and understand?</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Student's code should have at least two useful <code>assert()</code> declarations.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Does the program run correctly?</td>
<td></td>
</tr>
</tbody>
</table>
Solutions to cheating Issue?

• Reduce the number of examinations to 3
  • Each of 2 hours + 1 hour for checking status
• Collect programs in a per-problem repository and use similarity checks to catch cheating cliques.
  • Automatic tools may give too many false positives for novice, short programs.
• Drill completion interviews before stage-level assessments
  • Time consuming and labour intensive.