CSE 1325: Object-Oriented Programming

Lecture 26

Patterns (and Anti-Patterns) of Code and Life

Mr. George F. Rice

george.rice@uta.edu

Office Hours:
Prof Rice 12:30 Tuesday and Thursday in ERB 336
For TAs see this web page

Old hackers never die. They just exit to a higher shell.

Donut Day!

You made it! Have a donut!

- Please take only what you ordered until everyone has had a fair shot.
- After 10 minutes, have more!
- Each section has its own donuts – you need not leave any behind.
- Do NOT leave any behind!

Donut sketches by aleksandra85foto per the Pixabay license

Overview: The Rest of the Story

- Exam #3 Review
- Final Thoughts
  - Anti-Patterns
  - Unsolicited Advice
  - Class Survey

Paul Harvey, famous for his “The Rest of the Story” series, receives the Presidential Medal of Freedom on Nov 9, 2005
Photo by Shealah Craighead, in the public domain
https://georgewbush-whitehouse.archives.gov/news/releases/2005/11/images/20051109-2_la5h5992jpg-515h.html

Exams Are Graded and Posted

with, of course, a review and suggested solutions!

- No errata
- Your grade and scan of your graded exam was posted to Canvas Monday evening (after the makeup exam)
  - Appeal via email or Canvas Inbox ONLY to preserve a permanent record
  - Appeals MUST be filed by May 7 (decision may take longer)
- The Exam #3 review document with suggested solutions is on Canvas at Modules > Exam #3
  - Complete buildable code for the Free Response questions is on GitHub

Statistics and Such

- 74 of 83 students took the exam (including 1 makeup exam)
- 3 errata (first 2 were listed on the board and verbally announced)
  - The instructions said FR 2b would code a guard – it was actually 1a
  - FR 1j said to read the name “to the n” – should be “to the newline”
  - Several questions had incorrect points, and only 97 points (excluding bonus) were available – scans were corrected and +3 added
- Scores have ranged from 30.5 to 106 (out of 106)
  - An average of 76.5% (typical average in Java era of 78%) with a 77% median (grades trended down slightly)
  - No questions tossed, but 5 points added due to specific issues common across exams
- The exam timing closely matched Exam #2

Exam Markings

- **Vocabulary** – Red “X” marks errors. +2 for each correct definition, 30 points total. Points earned at bottom of page.
- **Multiple Choice** – Red “X” indicates incorrect answer (sorry, you’ll need to consult the Exam Review for correct answers). +2 for each correct choice, 10 points per page, 30 points total. Points earned are shown at bottom of each page.
- **Free Response** – Corrections *often* marked in detail – this took a LOT of time, but hopefully you’ll READ and CAREFULLY CONSIDER each!
  - Points awarded *per question step* indicated beside the question on the page on which the answer was *expected* (if you used external sheets). On a few exams, these were duplicated beside the question itself, although you don’t get the points twice!
- **Final Score** – The sum of all points on every page has been posted **on Canvas *only*** (NOT on the exam)
  - No scaling of exam grades was expected or employed

Continuing Access to My GitHub

- The cse1325-prof repo will be renamed cse1325-prof-202501 before next semester
  - The *next* class will get a *fresh* cse1325-prof
- You may continue to access it for as long as you like and it exists
  - I have no plans to delete it, but bits rot
  - If you value the material, you may fork it
  - https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/fork-a-repo

Releasing Your GitHub

- As of May 12, 2025, your CSE1325 code is your own
  - You may make your repository public
  - You may republish code you wrote under any license
  - You may republish any code you wrote that is mixed with my code as long as:
    - My portion of the code is clearly marked to avoid confusion
    - You follow the Gnu GPL 3.0 for mixed code products
- I encourage you to continue to grow free software (both share-alike and permissive) and free culture (such as Creative Commons creative works)
  - And share the wealth with your artistic contributors!

Anti-Patterns

- Anti-Patterns are common errors reflected in software systems
  - Common enough to be given a name
  - One or more strategies are known to exist
- This is a small subset of the most common / irksome (IMHO) anti-patterns
  - Wikipedia actually has an extensive list
    - https://en.wikipedia.org/wiki/Anti-pattern
  - But only these *may* be bonus questions on the exam!

Anti-Patterns: What Not To Do

Anti-Pattern: Comment Inversion and Documentation Inversion

- **Comment Inversion**: Adding obvious code comments (“adds two variables”) and omitting useful comments such as why those variables should be added in the first place!
- **Documentation Inversion**: Explaining the system in terms of itself (“select 'New Foo' to create a new Foo”) rather than documenting what a Foo is and why the user may want one
- In both cases, empathy is power. Consider the reader and what they are likely to want to know, rather than just writing down what you happen to know
  - User-testing documentation is helpful to learn empathy

Anti-Pattern: Error Hiding

- Catching an error and either doing nothing with it or displaying a meaningless message
  - May also refer to destroying evidence – overwriting the stack trace or disposing of problematic input
  - May also refer to addressing the error inappropriately, e.g., printing an error message in a library routine
- Catch exceptions only for specific reasons for which reasonable remediation is well-understood

AntiPattern: Cargo Cult Programming

- Inclusion of code, language features, data structures, or patterns without understanding why
  - Often occurs when an inexperienced programmer copies code without understanding it
  - Lack of understanding results in redundant code, subtle errors, & unnecessary or misleading comments
- Complete due diligence and thoroughly understand EVERYTHING you include in your designs, code, and documentation
  - Ask questions! That what Sr. Programmers are for!

AntiPattern: Not Invented Here (NIH) Syndrome

- NIH is reimplementing an existing solution, e.g., a standard class library, due to the mistaken belief that code “not invented here” is inferior
  - If the longevity of the existing solution is in question, delegate to the Buyer to escrow or otherwise ensure access
  - If licensing concerns exist, discuss with Legal
  - If the code doesn't fit the problem precisely, engineer an adapter or façade

Anti-Pattern: The “God Class”

- Functionality migrates upward in the class hierarchy until most methods are at the root
- This is a symptom that inheritance *may be* less appropriate for your design than composition
  - Use UML to understand the current class hierarchy
  - Test multiple redesigns with UML to more properly partition and encapsulate your data and allocate responsibilities to managable units of code.

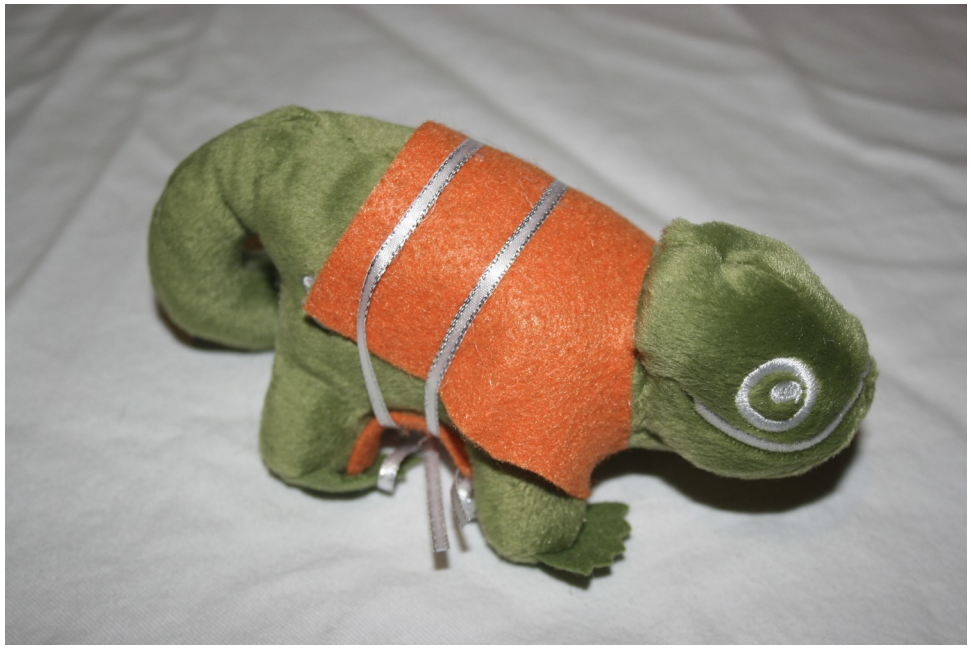
AntiPattern: The Big Ball of Mud

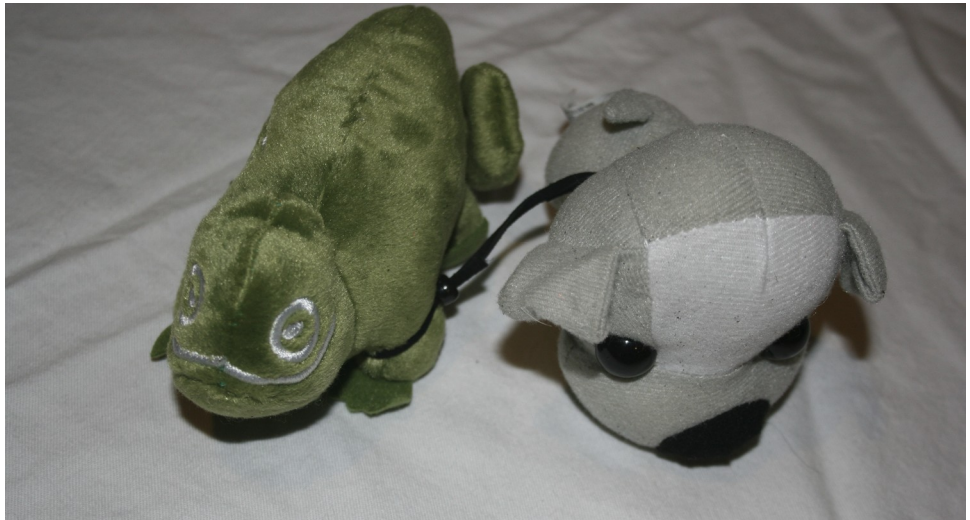
- A software system lacking any discernible architecture, and is thus a “haphazardly structured, sprawling, sloppy, duct-tape-and-baling-wire, spaghetti-code jungle” (Wikipedia).
  - Often the result of a project with high programmer turnover and a long life of unstable requirements, bad managers, and tight budgets
  - Sometimes called “legacy code”
- *Usually* avoid temptation to rewrite – instead, try to *refactor*
  - Start by writing good test code that captures correct behavior
  - Next, define a clear and appropriate architecture for the system
  - Replace portions of the system one-by-one with well-designed code, using your test code to verify equivalent functionality
  - Continue until the Big Ball of Mud becomes maintainable enough

Unsolicited Career Advice

- **Integrity matters most.** Never lie, mislead, or claim undue credit *for any reason*.
- **Make your team and your boss look good.** Credit others for success, but accept responsibility for your own mistakes quickly. **Focus on the solution**, not the blame.
- **Be a teacher** and help others grow and succeed, then cheer their success.
- **Network broadly.** Remember peoples' names. Cherish and practice loyalty.
- **Be the person who knows *who knows* the answer.** That's often much more valuable than actually knowing the answer. You can't know everything.
- **Prefer "Yes".** Consider no task beneath you, and do what is necessary for the team to win. **But say "no" when necessary**, e.g., to optimize your work load.
- **Code early and often.** A software manager or architect who doesn't code is clueless.
- **Learn new languages and tools often.** Automate everything.
- **Keep a personal task backlog.** Keep it updated. Work it by priority.
- **Keep an engineering log.** Write down everything, especially solutions *and reasons*.
- **Start stuff. Show initiative.** Motivate your team, even if you're the junior member. Make good things happen. That's what is *really* contagious – and it keeps work fun!

The “Geekos” Awards





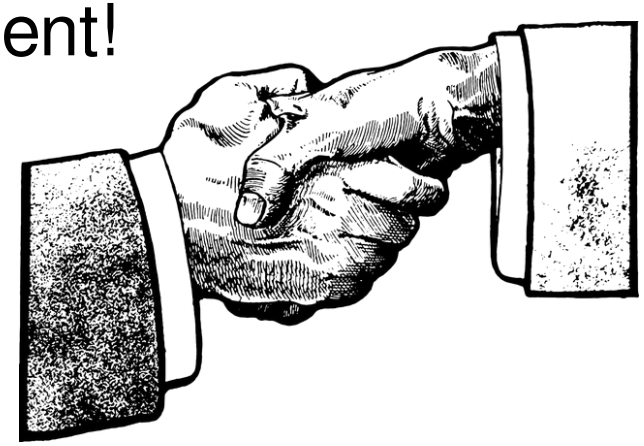
The “Geekos” Awards

The “Geekos” Awards

The “Geekos” Awards

Class Survey

- The class survey is now open!
  - I see *no* feedback until *after* your final grades are posted
  - I read and consider *every* comment!
  - **Completely anonymous**
- WARNING: Survey closes *today*!



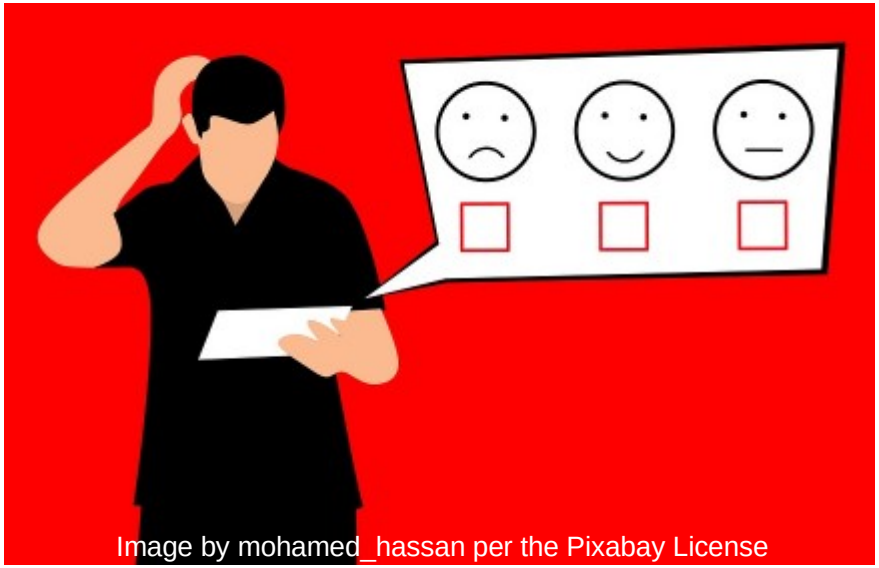


Image by mohamed_hassan per the Pixabay License
https://pixabay.com/en/experience-feedback-survey-customer-3239623/

I'll accept and **appreciate** additional feedback via email or in person, including **updates** on your academic and professional progress, at any time in the future!