Final Review

JD Kilgallin

CPSC:480

11/30/22

Final

- Wednesday, Dec 7, 5:15-7:15 PM, CAS 134 (regular classroom)
- One page of notes **single sided** allowed. You will turn this in with the exam.
- Comparable to the practice final exam in length and composition (~75% content since midterm). All lecture and assignment content is examinable except as previously marked.
- Scenario for later sections to be posted ahead of time
- Extra review/study session Monday, Dec 5 @5:15, CAS 230 (my office)
- Top recommended materials to review:
  - Midterm review lecture (12b)
  - Final review lecture (28b, i.e. this one)
  - Practice final & solution
  - Exercise 2a solution (Pokedex requirements)
  - Exercise 6 solution (test cases)
  - Project 3&4 handouts

Topics Pre-Midterm

- Software engineering fundamentals
- History of software
- Software engineering terminology
- Software products and projects
- Software development lifecycle
- Software engineering methodologies
- Software development organizations
- Software engineering roles
- Tools for software engineering
- Version control/Git/GitHub
- Collaborative development
- Requirements engineering
- Software project planning
- Software product design
- Software systems modeling
- Formal methods

Topics Post-Midterm

- Cross-cutting concerns
- Code quality concepts
- Code style guidelines & standards
- Code reviews
- Software architecture patterns
- Measuring & managing complexity
- Refactoring
- Concepts of software bugs
- Bug lifecycle & triage
- Quality assurance concepts
- Software testing methodologies
- Release engineering
- Software configuration management
- Software automation concepts
- Risk management principles
- Sales engineering concepts
- Other expectations of SW Engineers
- REST APIs
- Data science
- Applications of Python

Not covered

- Public Key Infrastructure beyond code signing concepts
- Keyfactor coding standards (review exercise 5 standards instead)
- Guest lecture content
- Python syntax and language concepts
- Topics excluded from midterm (modern history, interview tips, markdown, GitHub, Temporal Logic of Actions model-checker)

Learning objectives lectures 13-17

- Cross-cutting concerns
- Code quality concepts
- Elements of coding standards and style guidelines
- Reasoning behind coding standards
- Code review concepts and strategies
- Software patterns and frameworks
- Measuring complexity
- Managing complexity
- Types of software refactoring
- When to refactor

Learning objectives lectures 18-19

- Software defect concepts
- Categorizing bugs
- Software defect lifecycle
- Tracking bugs
- Root Cause Analysis
- Quality assurance concepts
- Software testing concepts
- Types of testing
- Writing and running tests

Learning objectives lectures 21b-22

- Software build processes
- Software versioning
- Software release engineering
- Software deployment and installation processes
- Software Configuration Management (SCM) concepts
- Purpose of automation
- Types of automation
- Automatic execution of software development tasks
- Automating other processes

Learning objectives lectures 23-26

- Risks associated with software engineering
- Risk factors
- Risk management concepts
- Preparation for other tasks required of practicing Software Engineers
- ~~Fundamental concepts of Python development~~
- Applications of Python for software engineers

Cross-cutting concerns – system, integrity

- Configurability – Parameters controlling operation of the software
- Logging – Recording actions taken by the software
- Observability – Allowing insight into the operation of the software
- Performance – Latency and throughput of the software
- Memory Management – Effectively maintaining data in memory
- Persistence – Permanent storage of data
- Security – Protection against malicious actors and threats
- Privacy – Confidentiality of sensitive data and prevention of disclosure
- Data Validation – Ensuring data conforms to expectations
- Error detection – Identification of data inaccuracies or inconsistencies

Cross-cutting concerns – human factors, arch.

- Aesthetics – Appearance of user interface
- Localization – Translation of I/O to user language and conventions
- Accessibility – Ability for all users to operate the software
- Help – Ability to provide guidance on usage of the software in context
- Licensing – Adherence and enforcement of code terms and conditions
- Compliance – Conformance to applicable standards and regulations
- Extensibility – Ability to easily enhance/extend program functionality
- Scalability – Ability to meet high usage demand
- Testability – Ability to write and run automated test cases on program

Code Style, Standards, & Reviews

- Code quality encompasses many aspects of the source code, like comments and variable names
- A consistent code style increases readability and maintainability of the source, as well as reliability of the product and user perceptions.
- It is important to be able to comprehend and recall program functionality on a quick read and have confidence in code changes; this requires more time up front but saves time in the long run.
- Following separation of concerns and limiting technical debt are two key parts of code quality.
- Many teams adhere to code standards, and may use a linter to enforce compliance.
- Code reviews are a common way to improve code quality (& skills)

Avoid "else"

Avoid excessive indentation by handling special conditions in an “if” statement and keeping the “expected” path at an outer level.

Bad:

```
if (valid) {
        doAllTheUsualStuff();
}
else {
        throw new Exception(…);
}
```

Good:

```
if(!valid) {
        throw new Exception(…);
}
// Don't put an else here!
doAllTheUsualStuff();
```

Return early

- Avoid allowing execution past the point that the return value is known – if a return value is set but the method continues running, it isn't clear whether it can still be modified or re-assigned.
- Avoid assigning a value that will always be re-assigned and will never actually be returned.

Bad:

```
int result = -1;
for(...){
      if (condition) {
            result = i;
            break;
      }
}
return result;
```

Good:

```
for(...){
      if (condition){
            return i;
      }
}
return -1;
```

Architecture Pattern

- Most problems have many solutions. Some are easier, some are better for certain situations, etc. But most solutions don't need to reinvent the wheel.
  - For example, Reddit, Keyfactor, and retail inventory systems all implement Create-Read-Update-Delete (CRUD) operations for different resources (i.e. posts, certificates, goods) and have similar architectures as a result.
- Recall: a pattern is a recognizable, repeatable template for a solution to one class of problems that has been proven to be effective.
  - We discussed the Model-View-Controller (MVC) pattern and briefly mentioned a few others.
- May dictate modules' internal construction or only describe interface.
- A product may use multiple architecture patterns.
- There are also patterns for classes, interfaces, and other entities.

Cyclomatic Complexity

- Number of distinct code paths through a program
  - Two paths are distinct ("linearly independent") if you can list the instructions in order, and there's at least one instruction that's in one but not the other.
  - Every conditional ("if" statement, ternary "? :", etc) increases CC by number of Boolean terms ("if (a ^ b) {…}" = "if (a) { if (b) {…}}"; CC += 2).
  - Every loop increases CC by 1; at the end of the loop, you can go on or go back.
- If you drew a state diagram with **every** line of code as its own state, the cyclomatic complexity is E - N + P, where E is the number of transitions (edges), N is the number of states (nodes), and P is the number of entry points or disjoint components.
- Can be computed for any level: an individual function, class, module, application, or even a larger system of multiple applications.

Refactoring

- Changing the structure of a software system or component without changing its functionality.
- Common reasons include eliminating technical debt, improving product performance or reliability, making code more readable/ testable/maintainable, swapping libraries or technologies used for an operation, or because you don't understand someone else's code.
- Inevitable in large projects; may comprise 20-30% of developers' time!
- Frequently done in conjunction with new development; halting new feature development is unappealing to stakeholders. This is a factor in why it takes more work to modify a large product than a small one.
- Can provide a major productivity boost to the rest of the team.

Concepts of Software Bugs

- Bugs are software defects caused by errors in programming.
- Bugs can be serious issues causing significant harm.
- Bugs can be found at any point in the product lifecycle after the code has been written, but the sooner they're caught, the better.
- Bugs may impact product functionality, performance, usability, or security.
- There are many possible causes of bugs.
- Severity measures user impact, priority measures vendor impact.

Bug Lifecycle

- Bug fixes are software development and follow a lifecycle similar to development for new requirements.
- Bugs are reported, triaged, assigned, investigated, reproduced, & fixed
- Fixes must be verified and released, and test cases should be added to prevent regression.
- Bugs may not make it all the way through the lifecycle and end in another terminal state (possibly to be reopened in the future). Common reasons to close a bug report include:
  - "not a bug" or "working as intended"
  - "not reproduceable"
  - "can't fix" or "won't fix"
  - "need more information"
  - "duplicate"

Bugs Summary

- Bug lifecycle includes report, triage, investigation, fix, and verification.
- Bugs are usually reported through software, but multiple systems may be needed to take reports from all types of stakeholder.
- Triage meetings decide on handling of a bug report.
- Fixes must be verified and released, and test cases should be added to prevent regression.
- Root cause analysis involves identifying the underlying code issue causing the bug and identifying the desired solution.
- Good design, code reviews, and automated tests help prevent bugs.

SCM

- Mature software organizations don't take the raw program bytes and put them through the release process for the first time at the end.
- Changes to *any* input or process that affects release artifacts between versions need to be carefully documented ("change control" or "change management").
- Software Configuration Management is the umbrella activity that occurs throughout the development cycle to manage these changes, along with the combination of people, processes, and tools that participate.
- Project management is responsible for enforcing change control process, and it's a major function of project management and version control software (frequently *called* SCM software).
- The source code repository and commit history are important SCM software components developers interact with regularly, as are CI/CD tools and Infrastructure-as-Code platforms (e.g. Chef, Puppet, Ansible).

Release Engineering & SCM Summary

- Software builds are used to convert source code into executables.
- Software releases start by completing code, creating a release candidate build, and conducting acceptance testing.
- Code signing certificates are a critical tool for ensuring build integrity.
- When a release is approved, it goes through a distribution process.
- Released software is more than just a compiled binary.
- Tracking released software versions and their input is essential.
- Release engineering is the discipline of ensuring quality in this process.
- Deployments can be very complex, especially with multiple instances.
- SCM is the activity of managing inputs to the release process.

Software automation

- The application of software technologies to minimize human input required for a task.
- Three main goals:
  - Efficiency – Reduce workload by eliminating extraneous steps so developers can focus on other tasks.
  - Reliability – Reduce human error by allowing computers to execute defined procedures.
  - Productivity – Provide self-documenting, distributable solutions that can be used across a team and beyond.
- Widely used by many IT disciplines to automate a broad range of business processes and tasks across the whole organization.
- Software engineers are uniquely positioned in an organization with the skillset to implement advanced, high-impact automation.

Types of Risk in Software Engineering

- Project Risks – Threats to the project plan. That is, factors that may cause the software to not be deliverable on time and on budget.
- Technical Risks – Threats to the software design. Implementations that turn out to be more challenging than anticipated can make it more difficult or even impossible to realize software requirements.
- Business Risks – Threats to the development team. This includes:
  - Market Risk – Building a product nobody wants. Leads to project termination.
  - Strategic Risk – Building a product that no longer fits into the vendor's business strategy (reorganizations/M&A, business pivots, new opportunities)
  - Sales Risk – Building a product that is hard to sell due to complexity.
  - Management Risk – Losing support from a senior executive or key stakeholder
  - Budget Risk – Vendor becomes unable to allocate resources to the project.

Risk Management Summary

- There are many types of risk that occur in software engineering.
- Good preparation helps to avoid risks turning into incidents and to limit the impact of the incident.
- Not all risks can be anticipated and can only be reacted to.
- Most risks don't lead to absolute catastrophe but it is still worthwhile to try to prevent them and develop contingency plans for them.
- The risks that are mainly the responsibility of the development team are to the project budget and timeline, or implementation in the face of technical limitations.

Other Activities

- Software engineers are expected to participate in a wide range of activities outside of writing code, even outside of the software development project lifecycle.
- Software engineers are leading experts on some aspects of the programs they write, and this makes them a great resource for other areas of the business like sales, marketing, and support.
- Software engineers bring a lot of valuable experience and skillsets that can be applied to provide value to the company in other ways.
- Some aspects of the workforce affect software engineers differently, such as company security training.
- Many software companies are rich workplaces that enable their engineers to succeed at a wide range of tasks. Find a good fit for you.
- Software engineering offers exciting opportunities beyond your employer.

REST APIs

- Many applications define an interface that can be accessed over HTTP web requests.
- REpresentational State Transfer (REST) is a paradigm for standardizing request and response formats to access resources via web API.
- URLs are grouped based on resource types, and use nouns to identify resources. HTTP verbs "POST", "GET", "PUT", and "DELETE" define Create/Read/Update/Delete operations, respectively.
  GET https://keyfactor.local/Users/1/Roles could be a REST endpoint for retrieving the roles for user with id "1".
- The OpenAPI Initiative defines JSON and YAML formats for representing a RESTful API, called its "OpenAPI specification" (formerly "Swagger"). An OpenAPI specification defines the API with enough detail that it can be consumed to programmatically access the API – client code can be generated in any language to make HTTP requests with correct input/output data.

Data Science

- Data science is, broadly, the transformation of raw data into useful, actionable information.
- Data science requires the simultaneous application of computer science, math/statistics, and domain knowledge to identify patterns.
- Data science is valuable for businesses to gain actionable insights from data they've collected. For example, Netflix can use data science to recommend shows based on what else you've liked, and what other shows are correlated with the ones you've liked based on other users' preferences. This helps Netflix retain customers.
- The problems and techniques in data science can be very broad, but they help to provide insights to problems that don't have an exact algorithmic solution (like predicting Netflix preferences).

Python Applications

- Python is an expressive, interpreted language that is well-suited for short programs, especially ones involving data structure manipulation
- The functional programming constructs such as lambda functions, comprehensions, and map/filter/reduce/groupby functions reinforce a very clean coding style that engineers should follow in many other cases with other languages.
- Python is much less suitable for complex software engineering projects as the slow interpreter and dynamic typing make it difficult to assess impacts of code changes.
- Major applications of Python include software engineering interview questions, automation scripts, and data science analytics.

Don't forget

- Review exercises, quizzes, team projects, practice final, and solutions as well as lecture contents.
- Review the scenario when it's posted and practice the questions you might anticipate on it.
- One **side** of a sheet of paper for notes.
- Final review Monday classtime in my office.