CPSC:480 Software Engineering Final Exam Notes

Peyton Gardner

12/5/2024

Types of testing

1. **Black Box**
   - **Viewpoint:** The user
   - **Focus:** Functionality of the software
   - **What it tests:** The tester is concerned with whether the software behaves as expected, focusing on input-output behavior without considering the internal code or logic. It checks the software's overall functionality, usability, and adherence to specifications. The user views the system as a "black box."
2. **White Box**
   - **Viewpoint:** Developer (internal perspective)
   - **Focus:** Internal logic and structure of the application
   - **What it tests:** The tester has access to the source code and tests the software at the code level. This involves examining the program's flow, logic, and internal workings to identify bugs in specific components (e.g., code coverage, branch testing, path testing).
3. **Grey Box**
   - **Viewpoint:** Combination of tester and developer
   - **Focus:** Both the functionality and the internal workings of the application
   - **What it tests:** The test has partial knowledge of the internal structures (such as the system architecture or database design) but is primarily focused on how the system functions from the users perspective. Grey box testing often uses system architecture and documentation to focus on areas like security vulnerabilities, API calls, or authentication mechanisms

Test-Driven Development (TDD)

A software development methodology where tests are written before the actual code to ensure the system behaves as expected. It helps improve the design of the code, enhances maintainability, and reduces bugs.

Process:

1. Write a test (Red): Write a test the describes a small unit of behavior you want to implement. The test should initially fail. This confirms the test is valid and the feature has not yet been implemented.
2. Write the code to pass the test (Green): Write just enough code to make the test pass. It doesn't have to be pretty.
3. Refactor (clean up): Refactor the code to improve its structure, readability, or efficiency, while keeping the test passing. Clean up the code.
4. Continue these steps to add features to the software.

Principles: You refactor and "clean up" your code during each test. There is not a big "clean up" stage at the end. You do it as you develop. TDD gives you confidence with code changes. It ensures the functionality doesn't change.

Process Models

Waterfall Model:

The waterfall model is a linear and sequential approach where each phase of the development process is completed before moving onto the next. Overall, the Waterfall process is less successful than agile. It is an

easy, clear approach that is easy to track. However, it is difficult to go back and make changes once a phase is completed. It also works poorly when requirements are unclear or may change.

Phases:
1. **Requirements gathering:** Define all project requirements
2. **System design:** Design the system based on the gathered requirements
3. **Implementation:** Code the system
4. **Testing:** Test the system for bugs and verify that it meets the requirements
5. **Deployment:** Release the product for use
6. **Maintenance:** Address issues that arise post-deployment

Agile Model:

Agile is a more flexible and iterative approach to software development. It focuses on delivering small, functional increments of software in short cycles, often called sprints or iterations. It is lightweight, doing (vs. planning), multiple passes, evolutionary approach. It is strongly iterative and evolutionary. Change is inevitable. Agile will catch those changes. Strong focus on collaboration with customers and stakeholders.

Requirements Volatility:

Refers to the degree to which a project's requirements change during its development. It is an important factor to consider when choosing a process model. Waterfall assumes that all requirements are known upfront and that they won't change significantly during development. However, this usually isn't practical. Agile models are designed to handle high requirements volatility. This allows the team to respond quickly.

Overall, the waterfall methodology is generally worse than the agile methodology.

Software Evolution

Software Evolution = Software Development + Software Maintenance
It focuses on supporting maintenance, not development for processes, tools, etc. It does not distinguish between development and maintenance.

Greenfield Development

Greenfield Development is starting with nothing and create a finished product with no existing code, design, or previous system. There is little pure greenfield development in the real world.

Brownfield Development

Brownfield Development is starting with an existing project/environment and create a finished product with existing code, design, or previous system. Most development in the real world is brownfield development.

Lehman's Laws Summary

- Software must change over time
- Changes cause complexity, which increases the cost of change. Resources allocated to reduce complexity deplete resources for additional features
- Software projects in an organization tend towards the same quality, bug rate, etc.
- Over time, the rate of development for each iteration doesn't vary much. Tend to meet the same number of goals with each iteration
- Must implement features that were previously not considered necessary
- Without changes, the software appears to decrease in quality
- What affects software evolution is a multi-level, multi-iteration process

Refactoring

Refactoring- Process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. It does not change the behavior of the program. Changes to a small context (e.g., individual method) or the entire program. Its primary goal is to improve the code's readability, maintainability, and extensibility while ensuring it remains functional.

"Replace Nested Conditional with Guard Clause" Refactoring

This refactoring technique simplifies nested conditional statements by introducing guard clauses, which handle specific cases early in the code, allowing the main logic to be more straightforward. It improves readability, maintainability, and separates edge cases from core logic. Practice a few of these problems!

Git Workflow

Understand the main git commands, understand branches, GitHub issues, pull requests, and commit messages.

Software Lifespan Models

Stages that software go through from conception to death. Software in different stages is very different to work on and requires different types of management. Software is a product; therefore, stages are similar to the stages in the lifespan of other products.

Stages:
1. **Initial Development:** Similar to Waterfall, but of limited duration. This stage includes requirements, design, and implementation. Fundamental decisions: technologies, architecture, and program domain knowledge.
2. **Evolution:** Adapt the application to the ever-changing user and operating environment. Add new features. Correct mistakes and misunderstandings. Respond to knowledge gained (learning) by developers and users. Program typically grows in size and functionality. Evolution is possible due to software architecture and knowledge of the software team.
3. **Servicing:** Program no longer evolves- it decays, stabilizes, or management decision not to support evolution. Changes are limited to patches and wrappers- low cost but cause further deterioration. Very different process than evolution. There is no need for senior engineers. The process is stable and is easily measured and managed.
4. **Phaseout:** No more servicing. The system may still be in production. Users must work around deficiencies.
5. **Close Down:** Software is disconnected from users. Users are directed to a replacement. "Exit Strategy" is needed. Changing to another system requires training. What do you do with all the data?

Code Decay:

The loss of software coherence and knowledge. Less coherent software requires more extensive knowledge. If knowledge is lost, changes lead to faster deterioration. Loss of key personnel = loss of knowledge. Try to eliminate or slow code decay.

Reengineering:

Reversal from the *3. Servicing Stage* to the *2. Evolution Stage*. This is expensive and rare. Not simply a technical problem, as it must also address the historical knowledge of the software team.

Software Change

Software Change (SC) is the process of changing existing code.

Maintenance Change Categories:

- **Perfective:** improve documentation, rewrite code for computational efficiency, improve software design
- **Adaptive:** accommodate changes to data inputs, files, and hardware and system software
- **Corrective:** emergency fixes, routine debugging
- **Other**

Concept Location:

Concepts extracted from the change request. Extracted concepts are located in the code and used as a starting point for a software change.
Examples:

- Add Discover card for possible credit cards -> classes, methods, and data structures that involve other credit cards.
- Fix spelling of "password" (from "passwd") -> find out where "passwd" is in the system
- Change API method length() to size() -> rename method refactoring in API

Impact Analysis:

Determine the impact of change on an existing system. Change has **not** occurred yet. Typically done at a class/file level. This impacts set classes identified in concept location. Class dependencies are analyzed, and impacted classes are added to the impact set.

In short, software changes need to be identified, classified, planned, and executed. **Concept Location** determines where a change needs to occur in the code. Can use *regular expressions* (i.e., regex) or the structure of the code. **Impact analysis** used to determine the change's effect on the entire system. This decision might be to not perform the software change.

Scrum

Iterative agile software development framework. Holistic: one team performs multiple functions across overlapping phases. Common team responsibilities do not mean that developers don't specialize in technology, e.g., database, UI.

Roles:

- Product Owner: Business owner, the voice of the customer.
- Development Team: Self-organized team of 3-9 members.
- Scrum Master: Ensures that the scrum process is followed, but is not the team leader.

Sprint: Scrum unit of development. It is a fixed duration of 1 week to 1 month. It always ends on time, but which features the team completes may have changed.

Backlog: Set of requirements in user story format.

Product backlog: List of requirements ordered by business values and needs.

Sprint backlog: List of work the development team attempts in the next sprint.
The sprint begins with selecting sprint backlog from the product backlog.

Sprint Planning Meeting: At the start of a sprint. Select what work the team does. Put together the sprint backlog. The entire team works on prioritizing the product backlog (4 hours). The development team creates a plan for the sprint (4 hours).

Story Time: Estimation for backlog using planning poker or other methods. This should not be longer than an hour. Does not include breaking stories into tasks. The team decides the number of meetings per week.

Daily Scrum Meeting: Everyone on the development team has an update. Limited to 15 minutes at the same location and time every day. The meeting always starts on time, even if some are missing. Convenient location so people can get started right away on work.

Daily Scrum Meeting Questions:

- What have you done since yesterday?
- What are you planning to do today?
- What obstacles are in your way?

Burn Down Chart: Remaining work in the sprint backlog. Typically readable by all members (not just the development team).

Sprint Review Meeting: Review user stories completed and uncompleted during the sprint. Demo the completed user stories. 4-hour limit.

Sprint Retrospective: What went well? What could be improved? The entire team reflects on the past sprint. Time for making continuous process improvements. 3-hour time limit.

Documents/Artifacts

- **Product Backlog:** ordered list of requirements, typically in a user story format: title, description, estimate, priority
- **Sprint Backlog:** requirements to be implemented in a sprint
- **Increment:** all the product backlog completed during a sprint.
- **Burn-Down Chart:** the status of work on the sprint backlog.

SE Code of Ethics (in order)

1. Public
2. Client and Employer
3. Product
4. Judgement
5. Management
6. Profession
7. Colleagues
8. Self