**Final Exam**

**Multiple Choice Questions:**

* Software engineering definitions

An area of computer science which relates to techniques, methods, practices and tools for the application of a systematic, discipline, quantifiable approach to the development , operation, and maintenance o software;, that is, the application of engineering to software

* Software application domains

System Software • Application Software • Engineering/Scientific Software • Embedded Software • Product-line Software • Web/Mobile Applications • Artificial Intelligence Software

* Software engineering layers

A quality focus - process – methods - tools

* Generic process framework

A collection of activities, actions, and tasks that are performed when some work product is to be created.

* Framework and umbrella activities

A process framework establishes the foundation for a complete process by identifying a small number of framework activities that are applicable to all software projects.

* Process flows

• Communication •Planning • Modeling • Construction • Deployment

* Stakeholders

A stakeholder is a party that has an interest in a company and can either affect or be affected by the business. The primary stakeholders in a typical corporation are its investors, employees, customers, and suppliers.

* Prescriptive process models
* Agile process models

A process for one project might be significantly different than a

process adopted for another problem, project, team, or

organization.

* Recommended process model

Waterfall/V-mode/Prototyping/Spiral/Unified

* Actors (primary/secondary, active/passive)

Actor specifies a role played by a user or any other external system that interacts with our system.

* Use cases & User stories

Use case represents a set of actions performed by a system for a specific goal.

* Functional& non-functional requirement

Defines a function of a system or its component.

A requirement that specifies criteria that can be used to judge the operation of a system,

rather than specific behaviours.

* Requirements engineering tasks

• Inception

• Elicitation

• Elaboration

• Negotiation

• Specification

• Validation

• Requirements management

* Conflict resolution technique

• Agreement: stakeholders work together to negotiate a solution to the

conflict.

• Compromise: use alternative parts of various solutions to try and come

up with a solution that could be a compromise for all stakeholders.

• Voting: ask all stakeholders involved with the requirements and/or the

conflict itself to vote on a set of alternative options.

• Overruling: more senior stakeholder’s requirements or proposed

solution is the one that will be taken forward as the resolution.

* Grammatical parses
* UML diagrams (e.g.what one to use for a specific case).
* Requirements models

1. To describe what the customer requires.

2. To establish a basis for the creation of a software design.

3. To define a set of requirements that can be validated once the software is built.

* Design models

Class-based

Behavior-based

* Abstraction

Conceptual process wherein general rules and concepts are derived from the

usage and classification of specific examples, literal signifiers, first principles, or

other methods.

* Coupling & cohesion:

• Cohesion: is an indication of the relative functional strength of a module.

• Coupling: is an indication of the relative interdependence among modules.

* Refactoring
* Separation of concerns

Separation of concerns suggests that a complex problem can be more easily handled if it is subdivided into pieces that can each be solved independently.

* Information Hiding

Information hiding implies that effective modularity can be achieved by defining a set of independent modules that communicate with one another only when necessary

* Software components
* Architectural styles
* Patterns & anti-patterns

Design pattern can be thought of as a three-part rule which expresses a relation between a certain context, a problem, and a solution.

Anti-patterns describe commonly used solutions to design problems that usually have negative effects on software quality.

* Architectural Context Diagram
* Archetypes
* Basic design principles (e.g.Interface Segregation Principle, Dependency Inversion Principle, etc.)
* Mandel’s three golden rules for User Interface Design
* UX and UI

UX = User Experience

UI = User Interface

* UI metaphors
* Customer journey maps and user personas

1. Gather stakeholders.

2. Conduct research. Collect all information you can

about the things users may experience using the

software and define your customer phases.

3. Build the model. Create a visualization of the

touchpoints.

4. Refine the design. Make the deliverable visually

appealing and ensure touchpoints are identified.

5. Identify gaps. Note any gaps in the customer

experience or points of friction or pain (poor transition

between phases).

6. Implement your findings. Assign responsible parties

to bridge the gaps and resolve pain points found.

* Task analysis
* Google’s 5-Day UX design sprint

Day 1: Understand

Day 2: Sketch

Day 3: Decide

Day 4: Prototype

Day 5: Validate

* Useability guidelines

• Anticipation

• Communication

• Consistency

• Controlled Autonomy

• Efficiency

• Flexibility

• Focus

• Human Interface Objects

• Latency Reduction

• Learnability

• Metaphors

• Readability

• Track State

• Visible Navigation

* Views of quality

Pragmatic Views

• Transcendental View: argues (like Persig) that quality is something that you immediately recognize but cannot explicitly define.

• User View: sees product quality in terms of meeting the end-user’s specific goals.

• Manufacturer’s View: defines quality in terms of making sure a product conforms to its original specification.

• Product View: suggests that quality can be tied to inherent characteristics (for

example: functions and features) of a product.

• Value-Based View: measures quality based on how much a customer is willing to

pay for a product.

Software Views

For software, three kinds of quality may be encountered:

• Quality of Design: encompasses requirements, specifications,

and the design of the system.

• Quality of Conformance: is an issue focused primarily on

implementation.

• User Satisfaction = compliant product + good quality + delivery

within budget and schedule.

* ISO 25010 standard

Quality in Use Model

• Effectiveness: Accuracy and completeness with which users achieve

goals.

• Efficiency: Resources expended to achieve user goals completely with

desired accuracy.

• Satisfaction: Usefulness, trust, pleasure, comfort

• Freedom from Risk: Mitigation of economic, health, safety, and

environmental risks.

• Context Coverage: Completeness, flexibility.

Product Quality Model

• Functional Suitability: Complete, correct, appropriate.

• Performance Efficiency: Timing, resource use, capacity.

• Compatibility: Coexistence, interoperability.

• Usability: Appropriateness, learnability, operability, error protection, aesthetics,

accessibility.

• Reliability: Maturity, availability, fault tolerance, recoverability.

• Security: Confidentiality, integrity, authenticity, accountability.

• Maintainability: Reusability, modifiability, testability, modularity.

• Portability: Adaptability, installability, replaceability.

* Case studies of quality (e.g.Animusic)
* Quality costs

• Prevention Costs: quality planning, formal technical reviews, test equipment, training.

• Appraisal Costs: conducting technical reviews, data collection and metrics evaluation, testing and debugging.

• Internal Failure Costs: rework, repair, failure mode analysis.

• External Failure Costs: complaint resolution, product return and replacement, help line support, warranty work

* Review metrics

• Preparation effort ( Ep ): the effort (in person-hours) required to review a work product prior to the actual review meeting.

• Assessment effort ( Ea ): the effort (in person-hours) that is expending during the actual review.

• Rework effort ( Er ): the effort (in person-hours) that is dedicated to the correction of those errors uncovered during the review.

• Work product size ( WPS ): a measure of the size of the work product that has been

reviewed (for example: the number of UML models, or the number of document pages, or the number of lines of code).

• Minor errors found ( Errminor ): the number of errors found that can be categorized as

minor (requiring less than some pre-specified effort to correct).

• Major errors found ( Errmajor ): the number of errors found that can be categorized as

major (requiring more than some pre-specified effort to correct)

* Reviews and postmortem evaluations

A postmortem evaluation (PME) is a mechanism to determine what went right and what went wrong with the software engineering process and practices applied to a specific project.

* Errors and defects

• Errors: a quality problem found before the software is released to end users.

• Defects: a quality problem found only after the software has been released to end-users.

* Loop testing

Loop Testing: is a white-box testing technique that focuses exclusively on the validity of loop constructs.

* Basis path testing

Designing Test Cases

1. Using the design or code as a foundation, draw a corresponding flow graph.

2. Determine the cyclomatic complexity of the resultant flow graph.

3. Determine a basis set of linearly independent paths.

4. Prepare test cases that will force execution of each path in the basis set.

* Unit testing

Individual units of source code (methods, classes, modules, components, etc.) are tested to determine whether they are fit for use.

* Integration testing

Individual units of source code (methods, classes, modules, components, etc.) are tested to determine whether they are fit for use.

* Validation testing

Validates the requirements established as part of requirements modeling against the software that has been constructed.

* System testing

Software and other system elements are tested as a whole all together

* Equivalence partitioning

Method that divides the input domain of a program into classes of data from which test

cases can be derived.

Uncovers a class of errors that might otherwise require many test cases to be executed to correct.

* Cyclomatic complexity

Three ways to compute cyclomatic complexity:

1. The number of regions of the flow graph corresponds to the cyclomatic complexity.

2. Cyclomatic complexity V(G) for a flow graph G is defined as:

V(G) = E − N + 2

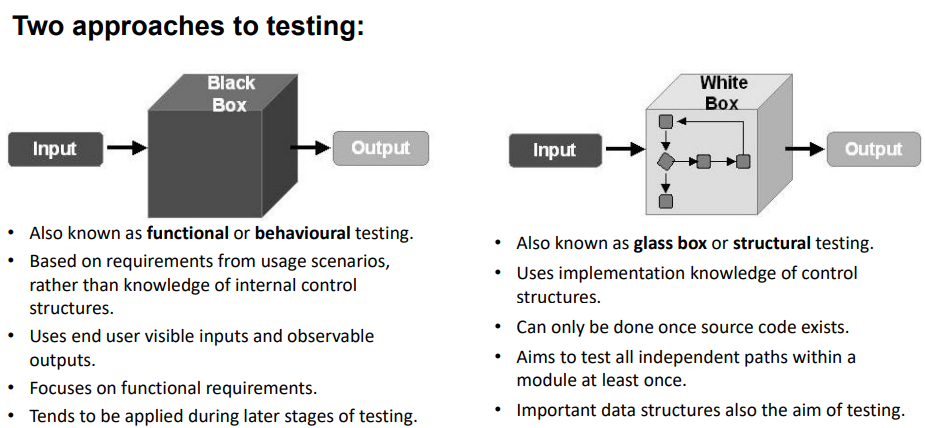
E is the number of flow graph edges

N is the number of nodes.

3. Cyclomatic complexity V(G) for a flow graph G is also defined as:

V(G) = P + 1

P is number of predicate nodes contained in the flow graph G.

* White/Black box
* Verification & validation
* Smoke testing

1. Software components that have been translated into code are integrated into a build that includes all data files, libraries, reusable modules, and components required to implement one or more

product functions.

2. A series of tests is designed to expose “show-stopper” errors that will keep the build from properly performing its function, causing the project to fall behind schedule.

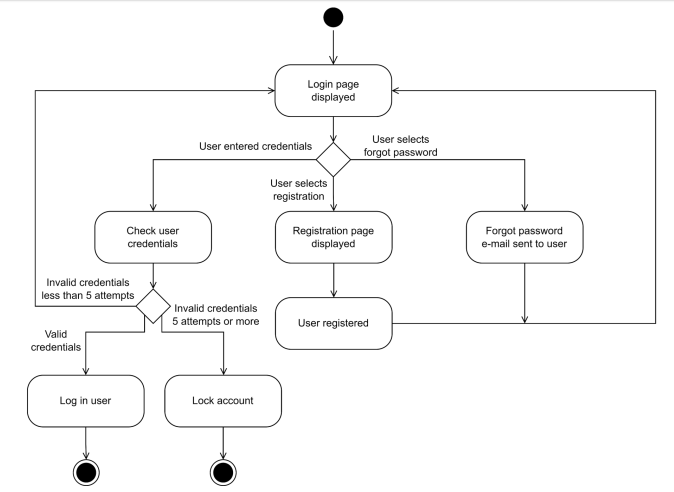
3. The build is integrated (either top-down or bottom-up) with other builds, and the entire product (in its current form) is smoke tested daily.

* Regression testing

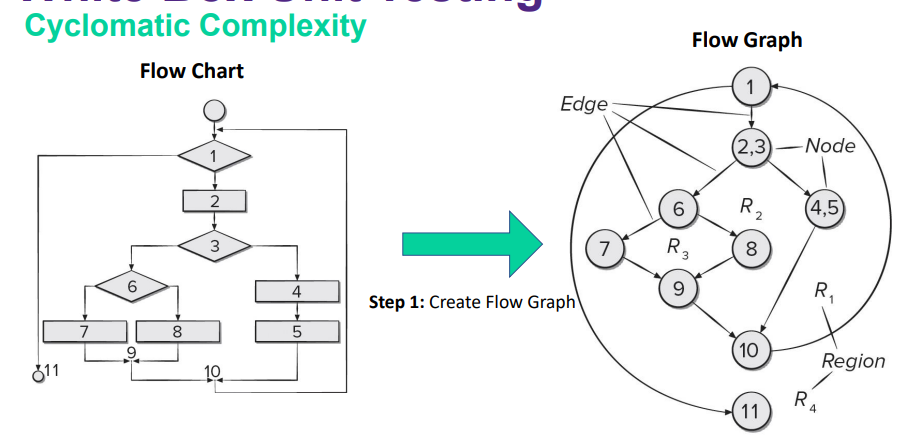
Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

**Short Answer Questions:**

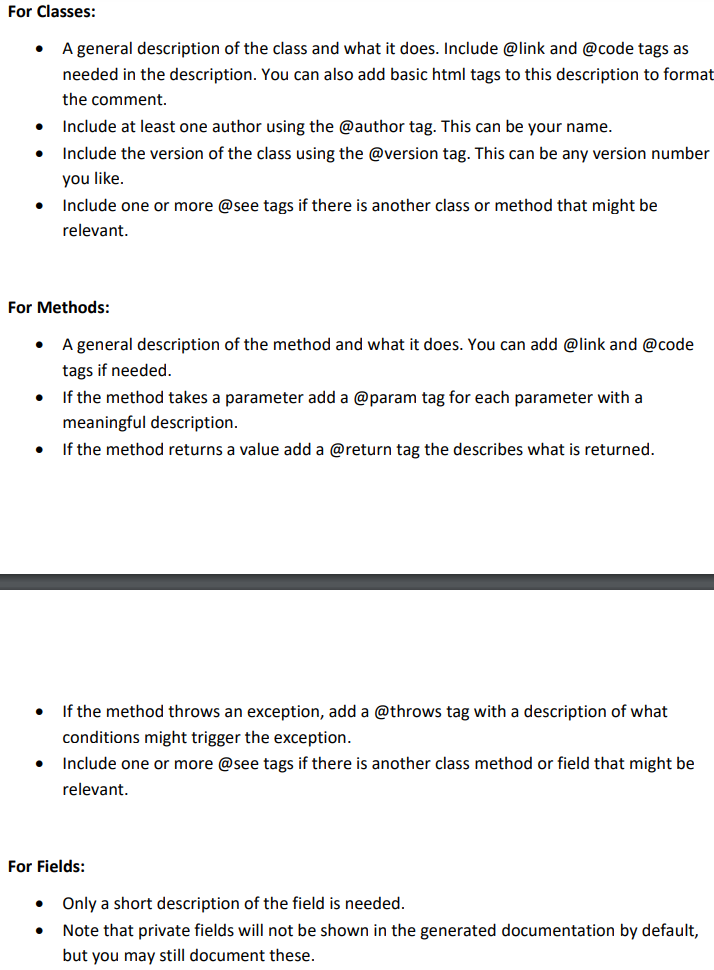
* Some important topics: UML activity diagrams (used for both representing algorithms and use cases).



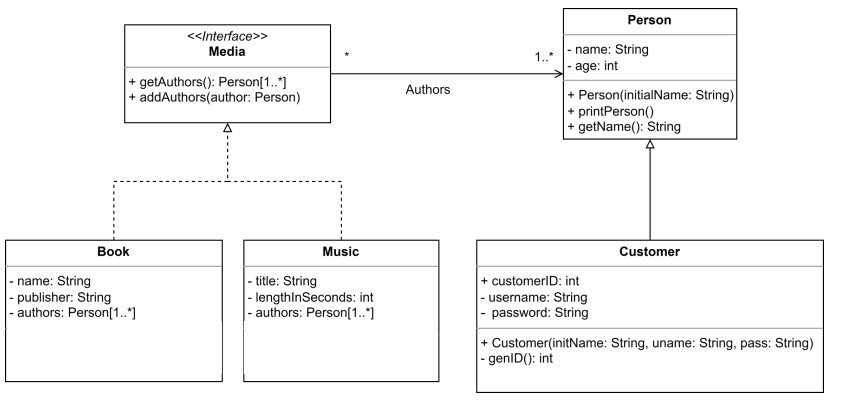
* Flow graphs and Cyclomatic Complexity



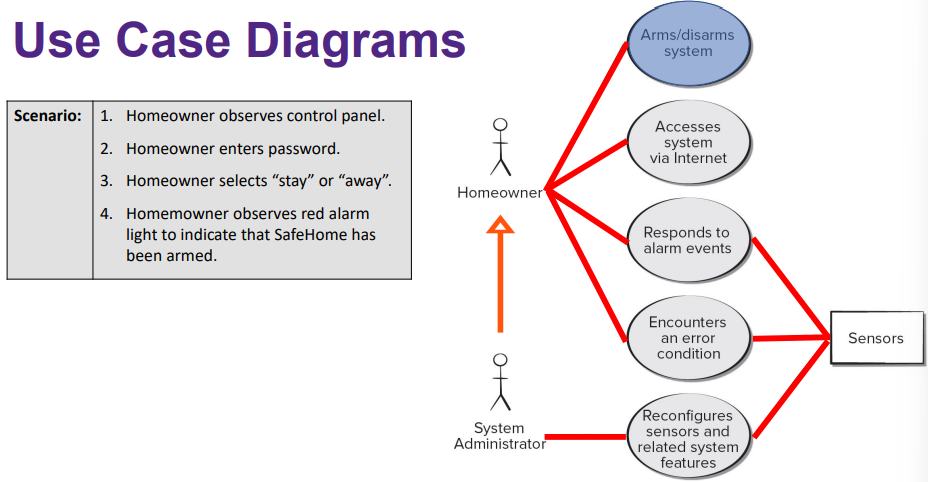
* Basis path sets
* JavaDoccomment (should know common tags: @return, @param, @throws, @see, @author, etc.)



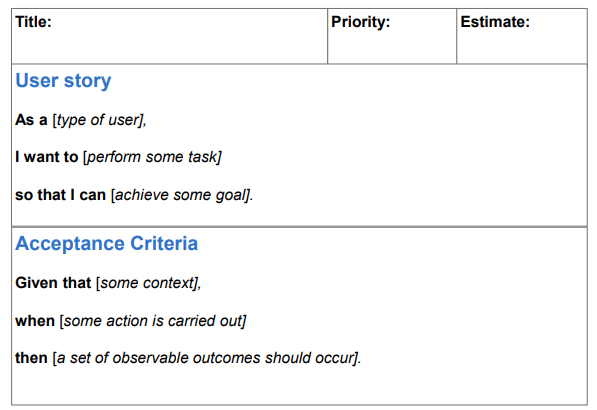
* JUnit (should know how common asserts work: assertEquals, assertTrue, assertThrows, assertArrayEquals, etc.)
* UML class diagrams (including relationships between classes, data types, method parameters, public/private, multiplicity, etc.).



* UML use case diagrams (including generalizations between actors).



* Use cases (template will be given)
* Actor descriptions (template will be given)
* User stories and acceptance criteria



* Anything from your project’s requirements, design, and testing documents