276 Course Content Summary

This summary highlights what material is and is not testable. Many questions may rely on you understanding and applying this knowledge; it is not sufficient to memorize this list, you must be able to use the material. Some items may have been specifically mentioned in class as being testable and may not be mentioned here.

- Most topics listed refer to the slide headings.
- "Know" means have memorized.
  - "Know the days of the week" means have memorized Monday, Tuesday, ...; and "understand" each of them.
  - You do not need to memorize explanations word-for-word; you should know in your own words.
- "Understand" means: once told the topic (or items), be able to talk about it/them.
  - "Understand all flavours of ice-cream" means given a flavour (say chocolate mint chip), be able to describe it; compare (similarities) and contrast (differences) it to other flavours. But, don't memorize all the flavour names (not asked "List all 31 flavours").
- The test will focus on material from lecture, but there will be some on Android development and tools.
  - This guide summarizes content from lecture and the Android assignments.

1. Lecture content

0 - Admin

- Review expectations and consequences for academic honesty.
  Be warned: I am **passionate** about it!
- Understand what are the three components to the course.
- Nothing testable.

1 - Intro to Software Engineering (SE)

- Know what is software engineering, and the importance of software.
  - Know how it is a discipline, and what it means to be concerned with all aspects of software production.
- Know the four fundamental software process activities.
- Understand each of the four essential attributes of good software.
- Understand the diversity of the types of software engineering.
  - Understand each type of application.
  - Able to discuss how each may need a different process.
  - Able to explain the need to be professional managed.

2 - Revision Control (Git)

- Purpose and need for revision control.
- Git:
  - Know how Git operates in terms of the working directory, local repository, remote repository.
  - Know what each of the following means: commit, head, clone.
  - Know what the following git commands do: clone, status, add, commit, push, log, pull, merge.
    - Able to understand Git commands (which appear in the command line)

    - Know what it means to be a distributed revision control system. Know how GitLab fits in.
    - Know how to do the following inside Android Studio: add, commit, pull, view which files have been changed and not yet committed.
    - Know basic git sequence for editing code (without a branch)
    - Know how to merge conflicts in a .java file with Android Studio
    - Know what `.gitignore` is for
    - Know what a tag is
    - Know what makes a commit message good.
    - Know how to revert changes in a file with Git.
- Know lock-edit-unlock and/vs checkout-edit-merge.
- Understand atomic operations, tag.
- Know the minimum requirement to being a team member
- Know how often commits and pushes are expected.
- Know the three points about how coding with source control is different, and why.

3 - Testing

- Know white box vs black box testing; able to discuss benefits of each.
- Know acceptance testing:
    - what it is, how to generate the tests, and how tests can be performed.
    - Know alpha and beta testing.
- Know usual content of a bug report, able to analyze or write a bug report.
    - Understand the life-cycle of a bug and what each of the resolutions mean.
    - Able to discuss value of an example file/input that demonstrates a bug.
- Unit Testing
    - Know what is a unit test, and its purpose.
    - Understand purpose of the test runner, and how it executes tests.
        - Know green-bar vs red-bar.
    - Able to write a JUnit 5 test:
        - `@Test`, `@Disable`
        - `assertEquals()`, `asserTrue()`, `assertFalse()`, `assertThrows()`
        - Should be able to write tests similar to testing `Puppy.java`
    - Understand steps in Android Studio to add tests to a class and run tests.
    - Need not know the Java `import` statements for JUnit.
- Testing Theory
    - Know FIRST properties of good tests
    - Understand an input vector
    - Know equivalence class partition testing and boundary value analysis and CORRECT boundary conditions
    - Know guideline based testing and its guidelines.
    - Able to apply equivalence class partition testing, boundary value analysis, and guideline based testing to an example.
    - Know code coverage; understand how to check it and view result in Android Studio.
    - Know why it is important that test code be held to same quality standard as production code.
    - Able to discuss what to do if a routine is unusually buggy.

4 - Software Processes

- Process Activities
  - Know what a software process is.
  - Know the four software process activities (same as "Intro to SE slides).
  - Understand each of the steps in the "Software Specification" overview.
  - Know the basic process to get from System Specification to an Executable System. Understand the different design activities.
  - Understand three testing stages.
- Software Processes
  - Know the two software development (planning) paradigms
  - Know the two software delivery (timing) options: single delivery vs incremental
    - Able to explain how each software development paradigm relates to each software delivery option.
  - Know the waterfall model and its phases (don't need exact names).
    - Understand waterfall model's ability to cope with change.
  - Know incremental development, and how it can be used by either paradigm.
    - Know 2 benefits of incremental development
    - Know a drawback to incremental development.
  - Understand refactoring.

5 - Git Process

- Know (feature) branches:
  - Know what is a GitLab issue, and why to use it.
  - Know how to create an issue, assign it to yourself, create a branch.
  - Able to explain why a feature branch is better than committing directly to master.
  - Know what a merge request is, how to create one.
  - Know the workflow for working with an issue and branch.
  - Know what the rest of the team can do with your merge request once created.

6 - Change Risk

- Know why change is inevitable, and the cost of change.
- Know change avoidance vs change tolerance, and one technique suited to each.
- Throwaway Prototyping:
  - Know when and how it is used.
  - Know what can be ignored.
  - Know why it is a throwaway.
  - Know how it avoids change.
  - Understand the ways it improves a system
- Incremental delivery:
  - Know how it tolerates change.
  - Know use and benefits of incremental delivery.
  - Know how it reduces risk of project failure.
  - Understand problems with incremental delivery.

7 - Scrum

- Know the 4 changes listed between Agile and BDUF
- Know backlog, user story, iteration.
- For each roles (product owner, scrum master, team member, repo manager), know:
  - their function as part of the scrum team
  - their responsibility
- Sprint Ceremonies
  - Know what a sprint is.
  - Know the 4 ceremonies discussed and their purpose
  - Know velocity.
  - Know the 3 questions to answer in a daily stand-up, and benefit of the stand-up.
- Estimation
  - Know story points, why better than estimates in hours.
  - Know the estimation game, as played in class.

8 - Agile

- Agile
  - Know the "inspiration" for agile methods
  - Know the four "values" of the Agile Manifesto. Able to explain what each one is a choice between, and why agile methods make the choice it does.
  - Able to explain how each Agile value fits with “Inspect and Adapt”.
  - Able to explain how Scrum achieves each Agile value.
  - Know how Scrum applies “Inspect and Adapt”.
  - Understand the five principles of agile methods.
- Plan driven vs Agile
  - Able to explain applicability of agile methods and their limitations.
  - Able to discuss how agile methods apply to software maintenance.
  - Able to discuss the choice between plan-driven and agile development and the factors involved.
- XP
  - Know what XP is.
  - Understand each of the XP practices discussed
  - Know pair programming, not designing for change, and refactoring.
    - Know three refactorings.
- Test Driven Development (TDD)
  - Know test driven development, automated test, and its advantages.
  - Know XP testing: test driven development, customer involvement, and test automation.
  - Know the TDD and Pair Programming game.

9 - Requirements Engineering (RE)

- Know what is RE.
- Know user requirements and system requirements; quantitative vs qualitative.
- Know functional vs non-functional requirements
  - Which applies to the whole system vs specific portion?
  - Ambiguity, completeness, consistency.
  - Know how non-functional requirements may lead to functional requirements.
  - Able to apply metrics for quantifying non-functional requirements.
- Understand the domain requirements problems.

10 - Implementation Issues

- Complexity
  - Understand that developers work across many orders of magnitude.
  - Know the "primary technical imperative"; know why it is so important; know how it relates to levels of abstraction.
- Code Reviews
  - Know what a code review is.
  - Know the code review tips
  - Understand how a checklist of common defects could be used as part of a code review.
  - Know benefits of code reviews
    - Understand the relative effectiveness of code reviews, and unit testing.
- Style Guide
  - Know the reason for having a style guide.
- Code Reuse
  - Know the advantages of software reuse.
  - Understand costs of reuse, and danger of reuse.

10.2 – Code Sense

- Know what "code sense" is (3 points)
- Know and be able to apply:
  - Use intention revealing names
  - Principle of least surprise
  - Extract method
  - DRY: and methods to apply it
- Know the limits of adding comments to bad code
- Understand 2 and N-layer architectures
- Know temporal coupling, encapsulation, single-source of truth

11 - Requirements Document

- Know what a requirements document is.
  - Understand and discuss its uses.
- Know what requirements specification is.
  - Understand ways of writing requirements.
- Able to discuss advantages and disadvantages of stating requirements in natural language.
- Understand guidelines for writing requirements
  - Able to bake cookies as specified according to MIL-C-44072C (just kidding).
- Understand structured and tabular format for expressing requirements.

- Understand cases presented in lecture of problems arising from code reuse.

12 - RE Process

- Know the four activities common to all RE processes
- Elicitation and Analysis
    - Understand stakeholder involvement, and methods to elicit requirements from them.
    - Know (at least 3 of) the problems with requirements elicitation
    - Know interviewing: open vs closed, strength, weakness.
    - Know ethnography, its benefit and drawback.
- Know recording requirements in User Stories
    - Know the template.
    - Know what an epic is and what to do with them.
- Understand requirements management.

13 - Teamwork

- Know team vs group
- Understand benefits of a team
- Know stages of a team
- Understand basic team rules, and team decision making.
- Understand the habits of highly effective people covered in lecture.

14 - Gender Diversity

- Know what is meant by “CS’s diversity problem”
- Know about unconscious biases and how they correlate to perceiving oneself as unbiased.
- Know what the CS culture problem is, associated with gender diversity.
- Know what a microaggression is, able to give an example and recognize them. Know their affect on people.
- Understand the issues presented around diversity at Riot Games, Google’s “echo chamber”, and instructors observations about teamwork.
- Know the four “call to action” points.

15 - System Modeling

- What is system modeling
    - Understand 4 perspectives, able to associate UML diagrams with perspectives.
- Know context model diagram and its use.
- Know use-case diagrams
- Know class diagrams.
    - Indicating class name
    - Relationships (has-a, uses, is-a), know arrow for each and how to decide which one to use.
- Know state diagram: able to draw one given a description of states and transitions.
    - Know how to show start, states, what happens in a state, transitions via events, a transition from a group of states, and an end state.
- Understand model driven engineering.

16 - Legal & Ethics

- Know what open source development is.
  - Know the information about the 3 open source licenses discussed in class
    - Know the implications for using a component under any of those three licenses within a project.
    - Recognize GPL vs BSD licenses when you see them.
- NDA & Non-Compete
  - Know what an NDA is, and why an employer would require one.
    - Know what IP is.
    - Know what an obligation of non-disclosure means (see sample NDA).
    - Know the questionable aspects of the NDA discussed in class.
  - Know what a non-compete agreement is.
    - Know why a non-compete can be a problematic document to sign if not done carefully.
- Ethics
  - Understand the issues of professional responsibility
  - Understand code of ethics and each item in the ACM/IEEE code of ethics
  - Understand how acting in a moral way could cost a software developer.

17 - Other Topics

Assignment & Project Related Tasks

- Know how to work with model/view (or UI) separation. Able to argue the benefit of this.
- Know the singleton pattern: able to describe it, able to make a simple class into a Singleton, and able to explain its benefits.

Scrum Retrospectives

- Know the purpose of a retrospective and what it produces.
- Know and able to explain the 5 steps to a Scrum team retrospective.

Dysfunctional Teams (activity)

- Able to analyze action of team member and assign blame for project difficulties. Able to justify these ratings.

2. Android and Tools

- Git topics are mentioned in the above sections.
- Know the purpose of each file in `src/` and `res/` directories of the project created in the assignments, such as:
  - Layouts, activity classes (`.java`), android manifest, `string.xml`
  - Where to put images
- Know how to code the following:
  - Given the name (id) and class type of a layout element (such as a `TextView` with id `menu_title`), write the Java code to a create a variable reference to the object (using `findViewById()`)
  - Given the resource ID (such as `R.id.btn_test`) of a button, register an `OnClickListener` which prints a `Log` message when the button is clicked.
  - Given the name of a string defined in string.xml (such as `menu_help`), access the string via Java code.
  - Display a `Toast` message
- Know what each of the following are: activities, intents.
  - Understand the process of creating a second activity.
  - Able to write the call to switch to a new activity (using `startActivity()`, and with results).
  - Know how to encapsulate knowledge about extras to within a single activity.
  - How to separate the model (data-layer) from the UI/Controller (Android activities). For example, holding a reference to a lens collection, and working with that.
  - Know how to create and use a singleton across multiple activities; why a singleton is needed/useful.
- Understand
  - How a `ListView`, its adapters, and an array fit together
  - Purpose of a `TextWatcher`
- Understand the `Activity` stack
  - Know `finish()`
  - Know how to launch an activity for result, or launch with an intent for extra data.
  - Know how an activity can access information it was handed in an intent.
  - Need not memorize the exact arguments to Activity's methods
  - How to have an Activity produce the intent for other Activities to use to launch it.

- Nothing on:
  - Animations.
  - Generating UI components from Java code (such as dynamically generating buttons).
  - Project topics which were not covered in lecture (as these may have only been done by some fraction of students in the class).