Answers to Guide 3

1. Key features of AGILE development over traditional practices
   1. Individuals and interactions over tools and processes
   2. Working software over comprehensive documentation
   3. Customer collaboration over contract negotiation
   4. Responding to change over following a plan
   5. (emphasis on the left side over the right side)
2. Adaptive rather than predictive
   1. Rolling wave approach to schedule planning wherein project milestones are flexible and also the path by which to reach them.
   2. Focuses on adapting to the current reality of the project rather than analyzing and planning for the future.
3. Favor code to documentation
   1. AGILE development favors code because extensive documentation will be out of sync with the current code base due to the nature of iterative development.
   2. AGILE proponents believe there are better methods to achieve relevant project goals than by maintaining extensive static documentation
   3. There should be a balance of just enough documentation to assist in maintenance, communication, learning, and knowledge sharing.
4. Extreme Programming (XP)
   1. Practices: Pair programming and Unit testing
      1. Pair programming: agile software development technique where two programmers work together at one workstation – the driver writes the code while the observer/navigator reviews the code.
      2. Unit testing: software testing method where individual units of source code and/or sets of one or more program modules along with associated control data, usage procedures, and operating procedures, are tested to determined fitness for use.
      3. Test-driven development. (TDD)
      4. Customer on-site is desirable.
         1. Personal communication with customer over documentation.
   2. What is so extreme about XP?
      1. XP takes the best practices of traditional software engineering to “extreme” levels.
      2. It is a methodology that aims to improve software quality and responsiveness to changing customer requirements.
      3. Extremely short cycles and low ceremony.
   3. Divided into 4 basic activities within the software development process
      1. Coding
      2. Testing – unit tests, acceptances tests, automated testing, integration testing. (test driven development)
      3. Listening – client/customer input and business logic.
      4. Designing
5. Scrum
   1. Sprint: the basic unit of development in Scrum. Time-boxed effort (limited duration).1-4 weeks.
   2. Daily scrum: each day during a sprint where a team holds a stand-up meeting is held with specific guidelines.
   3. Review: Held at the end of each sprint. Review of work completed and planned work uncompleted. Determine what should be done in the next sprint. Present to stakeholders completed work.
   4. Retrospective: Held at the end of each sprint. Reflection on previous sprint. Identification and agreement on continuous process improvement actions.
   5. Backlog: the list of work the development team needs to address during the next sprint. Derived from the product backlog in priority order and used to fill tasks for the next sprint.
   6. Burn-down chart:
      1. Sprint burn-down chart: chart showing remaining work in the sprint backlog. Visualization of the sprint’s progress.
      2. Horizontal axis shows days in sprint and vertical axis shows work remaining each day.
6. KanBan:
   1. Scheduling system for lean manufacturing and just-in-time manufacturing.
   2. Manages and improves work by balancing demands with available capacity and improving the handling of system level bottlenecks.
   3. Kanban board visualizes a view of progress and process.
   4. Works is pulled as capacity permits, rather than work being pushed into the process when requested.
   5. It is a visual process management system which assists decision-making about what, when, and how much to produce.
7. Project Management Tools – Github
   1. Git – Tutorial
      1. Version control system: Tracks the history of a collection of files. Each version captures a snapshot of the files at a point in time and allows switching between the versions. Each version is stored in a repository
      2. Distributed version control: Each user has a complete local copy of a repository, can clone this repository, and each clone contains the full history of the collection of files.
         1. No single point of failure as each clone of the repository can be used as the original repository.
         2. Each clone can exchange versions of different files with other repositories.
         3. Also uses a central server repository but for pure convention.
      3. Centralized version control: Provides a server software component which stores and manages the different versions of the files.
         1. Has a single point of failure being the server.
      4. Clone: The process of copying an existing Git repository via the Git tooling, which contains the complete file history of the repository.
      5. Commit: Creates a new persistent snapshot of the staging area in the Git repository. Creates a new commit object in the Git repository and is immutable.
      6. Push: Sends new versions of a user’s local repository to remote repositories.
      7. Pull: Integrate changes from other repositories into a user’s local repository.
   2. Github
      1. Repo: Organizes a single project and contains files, folders, images, videos, spreadsheets, data sets, etc.
      2. Branch: Method to work on different versions of a repository at one time. Makes a copy or snapshot of the master branch. Can re-merge with the master branch via push.
      3. Pull request: Create and open a request wherein proposed changes that are to be reviewed and then merged into the branch.
      4. Understanding the GitHub Flow: A lightweight, branch-based workflow that supports teams and projects where deployments are made regularly
         1. Create a branch. Anything in the master branch is deployable.
         2. Add commits. Creates transparent history that allows others to follow and understand your changes.
         3. Open a pull request. Initiates discussion about commits and allow others to see what changes are merged if request is accepted.
         4. Discuss and review your code. Can continue to push to your branch in the process.
         5. Deploy. Can deploy from a branch for final testing in production before merging to master.
         6. Merge. Pull requests preserve a record of the historical changes to the code even after merging.
8. Android
   1. Understanding Activities and Intents
      1. Activities: Represents a single screen in the app with an interface the user can interact with. An app is a collection of activities that are independent of each other.
      2. Intents: Message objects that make a request to the Android runtime to start an activity or other app component in your app or in some other app.
      3. Intent data: contains a reference to the data you want the receiving activity to operate on.
         1. Explicit intent: specify the receiving activity using the activity’s fully qualified class name.
         2. Implicit intent: declare a general action to perform the intent and the Android system matches request to activity or other component that can handle requested action.
         3. Can only send one piece of information via a specific URI.
      4. Intent extras: key-value pairs that carry information the receiving activity requires to accomplish the requested action.
         1. Stored in a Bundle object (map).
         2. Keys are strings and values can be any primitive or object type.
         3. Use when passing more than one piece of info or info cannot be expressed by URI.
      5. Back navigation: Allows users to return to the previous activity by tapping the device back button.
         1. Uses the back stack – set of activities that the user has visited and that can be returned to by the user with the back button.
         2. Temporal navigation – back button navigates the history of recently viewed screen in reverse chronological order.
         3. Once popped from the stack, the activity is destroyed.
      6. Up navigation: a.k.a. ancestral or logical navigation. Used to navigate within an app based on explicit hierarchical relationships between screens.
         1. Uses a parent and child relationship between activities.
         2. Must specify the parent activity of each child activity in android manifest.xml
   2. The activity lifecycle and managing state
      1. @Override decoration: A Java annotation that tells the compiler the following method overrides a method of its superclass.
      2. Activity created (onCreate() method)
         1. Started for the first time and initializes the activity.
         2. Only required callback in each activity.
         3. Perform basic application startup logic that should only occur once.
      3. Activity started (onStart() method)
         1. Called after started state and also if stopped activity returns to foreground.
         2. Counterpart to onStop() method.
         3. User cannot interact with activity till onResume() is called, activity is running, and activity is in the foreground.
      4. Activity resumed/running (onResume() method)
         1. User is able to interact with the app in this state.
         2. Activity is initialized, visible on screen, and ready to use.
         3. Counterpart to onPause() method.
         4. Not a transient state – activity remains in state as long as in foreground and user is interacting.
      5. Activity paused (onPause() method)
         1. Occurs when activity is going into background, activity is only partially visible due to another activity overlaid on top, or in multi-window/split screen mode when another activity has focus.
         2. Don’t use with heavy-load operations as it delays transition to another activity.
         3. Can transition to stopped or resumed state.
         4. Used to stop animation/video playback, release hardware-intensive resources, or commit unsaved activity changes.
      6. Activity stopped (onStop() method)
         1. Used when activity is no longer visible on screen.
         2. Stored in the back stack unless killed by Android system when resources are low.
         3. Use for heavyweight operations, saving persistent data, or releasing resources.
      7. Activity destroyed (onDestroy() method)
         1. Activity instance is reclaimed by system and is shut down completely.
         2. Called when using finish(), user back or up navigation, low memory in device, or device configuration change occurs.
         3. Fully cleans up after activity.
         4. Use onPause() or onStop() to save state and data, not onDestroy().
      8. Activity restarted (onRestart() method)
         1. Used only if stopped activity is restarted again – called in between onStop() and onStart().
         2. Not usually used to stop or start resources – use onStop and onStart.