|  |  |
| --- | --- |
| **SDLC (Software Development LifeCycle)**  -Waterfall  -Agile (Pyramid metaphor)  -Iterative Delivery (Project in Chunks)  -Each iterations around 4 weeks  -Develop Flexibly  Example Agile Processes  SCRUM (Project Managment)  -Timeboxed Iterations that are 4 weeks  -Uses a Product Backlog and Sprint Backlog  -Daily Meeting (Pigs and Chickens)  -*Each Sprint has a deliverable*  Extreme Programming  EVOf  RUP Dx  **Android**  Activity - Visual representation of an Android Application  View - Interface Widget (button, text, spinner etc)  Intent - Use intent to switch activities  Service - Background Task    **UML Diagrams**  **Modifier**   * **Public** * **Private**   **# Protected**  **~ Package**  **N/A Package Visibility**    **The line between classes should have a 1-2 word description**  **Of their relationship.**    **Functions Format: Modifier FuncName(returnVar : Type)**  Example: +method(returnVariable, String)  Abstract Classes are shown as {Class} or *Class*  Class implements interface (points towards interface)    Class Inheritance from Class (higher level class pointed to)  Dependency (used for dependencies on outside packages)    **Aggregation - Wheel is part of Car, but wheel can outlive car**    **Composition - If the company object dies, the department object dies** | INVEST acronym -- Mike Cohn  Independent - Stories should be able to be developed independent of each other.  Negotiable - Do not put implementation language in story. Keep it general enough so you have design flexibility. Don’t try to put all the details in the title.  Valuable - Has to provide business value to the customer  Estimable - Can’t be too big, too complex, or too unfamiliar to developers  Small - Cannot span an iteration. If too big break into smaller stories and use the bigger story as an epic  Testable - Have to be able to specify acceptance tests for the story  **User Stories** : As a <>, I want to <> so that<>.  **Acceptance Scenarios:** GIVEN <actor doing something>, WHEN <actor does something> THEN <program does something>  This style of language is known as a Behavior Driven Development (BDD) format.  **Done Done:**  Test Cases Written  Acceptance Criteria Met  Code reviewed by another team member  **Objects**  -Know Information  -Maintain Connections  -Perform Services  -Make Decisions  **How to Model**  -Brainstorm - Domain Analysis / Noun Analysis  -Filter - Narrow Nouns down to things specifically relevant to customer  -Draw Model  -Domains are about ideas and associations, NO CODE!!!  **Responsibility Driven Design (RDD)**  -Single Responsibility Principle (SRP) - Each class should have a single role/responsibility  **Stereotypes**  -Information Holder - knows things  -Service Provider - Does things for others  -Structurer - Organizes data  -Interfacer - adapt one system to another  -Coordinator - Decide things - Non Adaptive Logic  -Controller - Decide things - Adaptive Logic  **Zenhub**  Issues - Features/Stories/Products  Milestones - Sprints/Iterations  Labels - filtering and organization  Estimation - Fibonacci Sequence  **Robustness Diagrams**  **Boundary Class -** UI or API class/External System  **Entity Class -** Class from Domain Model  **Controller Class -** Class for business logic  **Rules**  -External Actors only talk to Boundary Classes  -Entities only talk to Controllers  -Controllers can talk to Boundary Classes and Entity **Classes**  **Software Architectures**  Client - Server  P2P (Peer to Peer)  Blackboard  Pipe & Filter  MVC (Model View Controller)  N-tier  Layered  **Control Styles**  -Centralized  -Delegated  -Dispersed  **Trust Boundaries**  Pass copies of data instead of sharing  Check on timliness, relavance , and correctly formed data  Make objects read only to prevent modification |
|  |  |