**Software Quality**

Topics:

* Verification and validation
  + Verify against a specification, is it built correctly
  + Validation, verification might be right but is this what the user wanted, holistic, is this the right thing
* Types of testing
  + Unit testing
    - Quality assurance, improve developers lives, design flexible and changeable, allows for iterative development with working code
    - Programmer writes their own unit test, make sure bits of code do what you expect
    - Don’t move on until all tests pass
  + integration testing
    - putting pieces of working code together
  + functional and system testing
    - stress, performance, usability testing
  + acceptance testing
    - meet the req’s
  + beta testing
    - real world testing
  + regression testing- “smoke tests”
    - try to uncover bugs after making updates, changes didn’t break anything
* Unit testing, test cases
  + Test cases should have: ID, Short Description, States & Inputs, Expected Results
  + Reveal missing, inconsistent, or ambiguous specification
* Test-driven development (tdd)
  + Write unit tests before the code is written
  + Red green refactor?
* Code coverage and testing
  + Do the tests cover all of the code, edge cases?
* Inspections, including the fagan-style approach
  + next most used practice after testing for quality management
  + can save time on testing, discover defects
  + Fagan style- process for inspection that’s more formal
    - Inspect: detailed design, clean compiled code, complete unit test, requirement specs, design docs/models, test plans, system documentation
    - Group of people, moderator, recorder, author, reader, inspector
    - Don’t fix anything at the meeting, just identify what needs work
      * Planning, overview, preparation, meeting, rework, follow-up
    - Emphasize producing quality product not placing blame
    - Process control, like check points in the development process
    - Different from walkthroughs- collection and analysis of data to determine the new project schedule (based on amount of rework), next group of developers know the status of the project, quality assurance can study the effectiveness of the inspections, do more inspections need to be scheduled? At what stage is it most cost effective to schedule the inspection?

**Software Process Topics**

Topics:

* What do we mean by software process? Why does it matter?
  + Framework for defining tasks at each step of building software, how to develop, maintain, replace and alter the software
  + Increases the quality of the product, make process efficient
  + Planning, analysis, design, implementation, maintenance
* Software process models
  + Waterfall, spiral, agile, define a strategy for software development
* Examples: waterfall, spiral (big ideas, not details)
  + Waterfall- linear, no returning to previous phases, emphasizes documentation
    - Disadvantages: need to finish phase before moving on, requirements get “locked in,” emphasis is on deadline vs on user requirements, doesn’t emphasize risk, prototypes, user interaction, quality
    - Best for- projects with clear requirements that will not change
  + Spiral- has risk analysis at each stage, prototypes, iterative development, repetition
    - Cycles have stages: planning, evaluate/ risk analysis, develop and verify next stage, review and plan for next stage/ feedback
      * Best for- projects that may change with reqs that are not as well defined
  + Agile- extreme programming and SCRUM
    - Helps respond to unpredictability by using **sprints**
    - Cycles, ongoing releases, testing at each iteration
    - Emphasizes interaction with customer, developer, and tester, and less documentation
    - Working software to show to customer
    - Fast delivery, specific deliverable features at each iteration
* Software development methodologies
* Plan-driven vs agile development methodologies
  + Plan driven would be waterfall, more documentation
* Scrum – roles, ceremonies, artifacts, sprints
  + SCRUM- product owner, team, scrum master
    - Meetings: backlog refinement, sprint planning, daily scrum, sprint review, sprint retrospective
* Common issues: role of things like iteration, risk management, prototyping, quality assurance
  + Agile does iteration and risk emphasis during each cycle, make sure the requirements are what you want all the way along, make dynamic changes
* Risk management
  + Strategies
    - Identify, analyze, prioritize, develop mitigation plan
    - Try prototyping, simulation, look for bottlenecks
  + Anticipated unfavorable event that may occur during project development
    - Security leak, short on time, need to pay for some resource, can’t find bugs, lose team member, don’t have the skills required to do something, wrong requirements, incorrect design, ambiguous docs
* Software tools to assist in software development

**Configuration Management and Git**

Topics:

* General terms and concepts for VCS and DVCS
  + Enable multiple people to work simultaneously on same project
  + Integrates work done by team members
  + Access old versions of project
  + Distributed: modern, faster, fewer errors, more repositories
    - Make edits on working copy, can commit those changes to your local repository(add to staging area first), must push to central repository for all users to see your changes, then they have to pull to their own repository to work on it
    - Fetched branch is origin/master can merge with your local master
    - Pull fetches and merges in one op
  + Centralized: working copies committed directly to the central repository
* Creating repos with init or clone (general ideas)
  + Clone downloads a whole project and its history
  + Creates new local repository
* Git's commands: add, commit (Git's staging area, index)
  + Must do before committing to local repository, use descriptive messages
  + Don’t commit generated files
* Local and remote repos, pull, push, fetch
* Git branches, merge, resolving conflicts
  + Resolve conflicts manually, must add on the file you want to use
* Typical workflow working with Git or other DVCS
  + Fetch code from remote repo
  + Merge into local branch
  + Code/test file in local branch
  + Commit to local repo
  + Push to remote repo

**Requirements**

Topics:

* What is elicitation (requirement gathering) and how to do it
  + Figuring out what the client needs/wants their software to do
    - Interviews- questionnaire, automate existing manual process, simulate what people do in real life without the software
    - Prototyping- not a working system
    - scenario-type methods
      * stories/user stories, use cases- story board
* Functional, non-functional, constraints
  + Functional- what the system does, an action eg. Send an email when an order is made
  + Non-functional-constraints (how not what) and restrictions/characteristics (must send email within 12 hours of purchase)
    - Performance, dependability (also security), implementable, maintainability, testability
* Concepts including: domain, system context (related to system boundary, project scope)
  + Domain- problem area for the system
  + System boundary- what is our system responsible for/ what’s within the system/ interacts with other systems like user, other hardware
  + Project scope- what we need to do, what we don’t need to do
* Quality attributes for software requirements statements
  + Correct, unambiguous, complete, consistent, verifiable, design independent, traceable, traced, readable by customer, concise, organized
* Specification documents in plan-driven methodologies (general ideas)
  + User stories
  + Scenarios
  + Use cases
* Documentation techniques: scenarios, stories

**Requirements Engineering (part 2)**

* Requirement analysis
  + Techniques: interviews, scenarios, rapid prototyping, analyze existing system/documentation related to domain, info from marketing, formal meetings with clients
  + Trying to ‘capture’ what the customer expects the system to do
  + User oriented
* Use cases: details, examples, how different than stories
  + Use cases – identify actors and sequence of events that lead to some result, provide a high level view of the system, identify internal and external factors
  + Both tied into scenarios, both document requirements
  + **Use case has pre and post description**
  + **Distinguishes between exceptions and acceptable alternatives**
  + Use cases are families of scenarios, grouping similar behaviors
    - Good, bad, alternative things can happen
    - Stories are smaller
    - Use cases are more explicit, can define what is outside the system more clearly
  + Will sometimes accidentally lead to premature development
  + **In scrum- stories also have conditions of satisfaction for confirmation- become the basis for acceptance tests**
  + **Use cases -Document system boundary, good for designing test cases, good for designing user interface**
* Requirements specification: general principles
  + Restate the requirements in technical terms so developer can start design
  + Developer oriented, more rigorous modeling and documentation
  + Product backlog -> leads to stories, assign stories, sprints
  + Requirements in stories drive your work
* Formal vs. semi-formal vs. non-formal forms of specification
  + Notation slide: natural language, semi formal (uml and diagram), mathematical semantics (formal)
    - Non-formal- understood by everyone, but can be ambiguous
    - Semi-formal- unified modeling languages, structuring natural language, can semi be validated/error checked
      * **UML’s – standard way for documenting class structures** are good to use but not the ultimate solution, not a method but is *used* by methods
        + Analysis: Use cases, class diagrams,

Class diagram: documents classes, their subtypes and associations

* + - * + Design interaction diagram, package, state and activity diagrams
    - Formal- accurate, unambiguous, mathematical semantics
  + Problems with using natural language

**Security in Software Design**

* Confidentiality, integrity, availability (CIA)
  + Goals/objectives of information security
  + Confidentiality- data made viewable only for authorized users
    - Data at rest/ data in transit
  + Integrity- ensure accuracy + consistency of data over its entire life-cycle, while still modifiable and editable by authorized users
  + Availability- data is usable by and available to authorized users, avoid single points of failure, can we recover data if something breaks?
* Nonrepudiation and Auditability
  + Nonrepudiation – system can confirm that a sender cannot convincingly deny having sent a message (?)
  + Auditability- system can trace all actions related to a given asset
* Authentication and Authorization.  Principle of least privilege.
  + Authentication- confirm identity: know(eg password), have (RSA keychain), are (biometrics)
  + Authorization- specify who/what has access to system functions, data
  + Concept of least privilege- give user only the privileges essential to their work, this will reduce risk
* Hash functions use to support integrity
  + Hash function hashes data and gets some result, can use to identify the contents of something, if you send it and hash it again, the hash values should match
* Security and requirements, including anti-requirements and abuse cases
  + Fall under non-functional requirements
  + Anti-requirement: things you don’t want the system to do, what happens when security features fails or are breached
  + Abuse cases- interaction between the system and an actor, with harm resulting to the system, actor or other stakeholders, actor could be attacker or bad user

**Python**

* Modules, packages, package management, pip
  + Pip- package manager, finds install files for a package on the web and downloard them, will also download packages the original package was dependent upon
  + Package- software component that works with our programming environment
* Testing: assertions, what nose does, etc.
  + Nose runs tests on your code, sniffs out functions that start with test, doesn’t have to be a unittest
* Classes: constructors, common "magic" methods for len(), in, str() etc.
  + \_\_len\_\_() and \_\_str\_\_() are methods called by the function len()
* Lists and dictionaries (basic use, iterating over)
  + For e in list ?
* Nothing on files, pycrypto library