Midterm 1 Review

- We covered Three Modules

- What to expect:
  - 26 questions in total
  - True or False (10 * 2pts)
  - *Multiple Choice/Select/Fill in the Blank (10 * 5pts)*
  - *Short answer (6 * 5pts)*
  - ***Grades are not Finalized right after the exam.***

- *Often times, software requirements are derived from higher level user and system requirements.*
- All system requirements end up getting "flowed down" to the software subsystem.
- Functional requirements are statements of service that the system should provide. They describe what the system should do.
- Non-functional requirements are reserved to describe the things the system should not do.
- Non-functional requirements could affect the overall architecture of a system.
- It is very unusual for software requirements to be refined over time.
- It can be hard to differentiate a software requirement from a design feature/decision at times(Maintainabiliy, Efficiency,….).

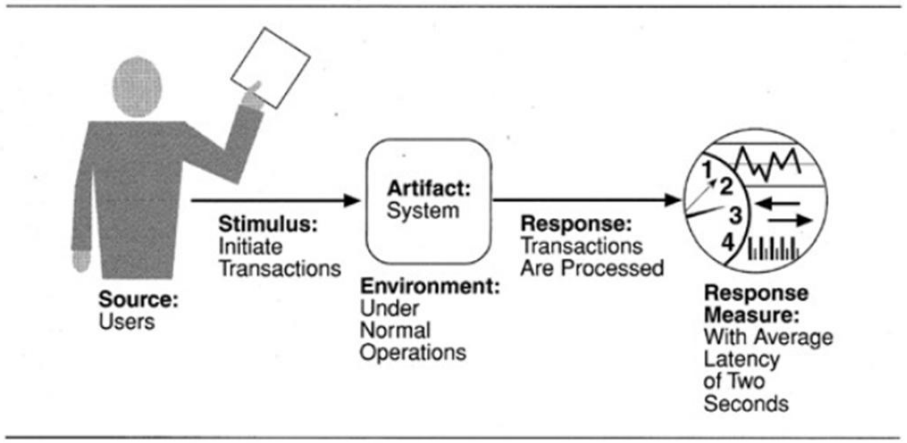
- *Often times, it is not helpful to observe what stakeholders need the system to do when trying to develop software requirements.*
- *A software requirements specification is informally controlled during the development of the specification, until the requirements have been reviewed and are "baselined".*
- *Fixing a software requirement error after product delivery could cost significantly more than fixing an implementation error.*
- *Rework can consume 30%-50% of a software product's total development costs.*
- *Of all of the software process models that we learned about, Agile is by far one of the oldest.*
- *Waterfall and "V" software development models are very similar to one another in many regards because they both traditionally result in a document / artifact of some type that is signed off and approved before proceeding to the next phase of development.*

- *There are multiple forms of Agile that can be adopted by different software development teams, depending on the nature of the product under development, team size, and discipline formality.*
- *There is a constant demand for developing software cheaper, better and faster.*
- *"Fast to failure" is one of the features of the Waterfall software development model.*
- *System modeling is accomplished through many different "views" of the system.*
- *Software quality attributes are qualitative, but not quantitative, measures of the "goodness" of software.*
- *The "Usability" quality attribute evaluates how easy or how hard it is to configure / reconfigure the software system if changes are made to it.*

- *Many software quality attributes can result in direct conflicts with one another.*
- *Quality attributes do not affect software requirements.*
- *Architectural patterns are a means of representing, sharing, and reusing knowledge.*
- *Using object oriented design features are useful predominately for small, simple, and compact software products that are limited in functionality.*
- *In the Unified Modeling Language (UML), structural design models describe the "static" structure of the system, while dynamic models show the static interactions between objects.*
- *There are multiple levels of potential reuse in a software product, including at the abstraction level and the component level.*

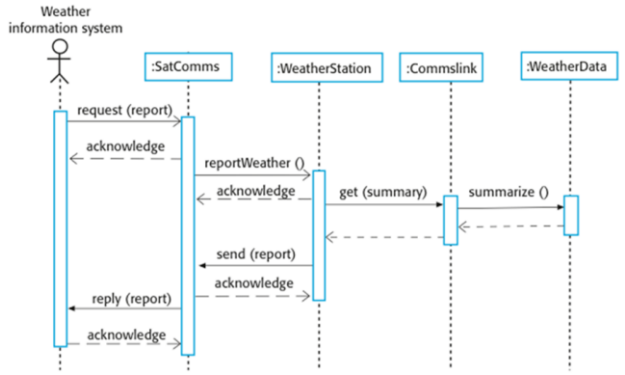
- *In software architecture modeling, we often consider the following perspectives:*
  - Context or environment the software operates in
  - Interactions between software and its components or environment
  - Structure of the data processed by the software
- For SW Quality Attributes:
  - Reliability: A product performs specific functions under specific conditions for a given amount of time.
  - Interoperability: Considers how a system interacts with other systems.
  - Performance: Evaluates how well the system executes specified / required functions over time.
  - Availability: Measures the amount of time a system is able to perform its intended functions.
  - Modifiability: Considers how the system can accommodate anticipated and unanticipated changes.

- *The following diagram describes the software quality attribute: Performance*

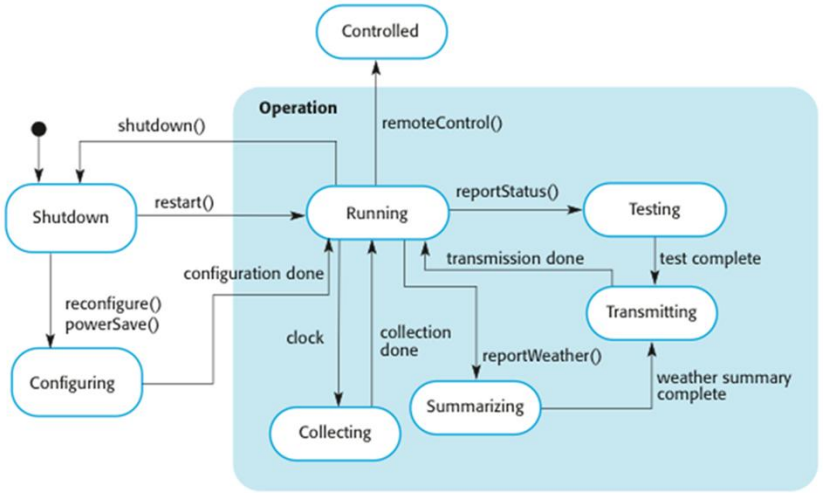


- *For the architecture patterns:*
  - Layered: Organizes the system into layers with related functionality at each layer.
  - Model-View-Controller: Separates presentation and interactions from the system data. Structured into logical components that interact together.
  - Client-Server: Organizes the system into services, each service delivered from a separate server. Clients are users of these services and access servers as requested.

- *Following is an example of sequence diagram*



- *Following is an example of status transition diagram*

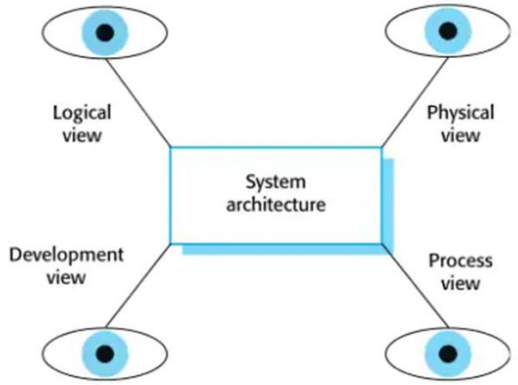


- *“Non-functional requirements“ includes:*
  - Security requirements
  - Efficiency requirements
- Primary activities performed in requirements engineering include:
  - Discovering requirements by working with various product stakeholders at multiple levels.
  - Maintaining the integrity of the requirements baseline by controlling and tracking changes to the requirements over time.
  - Ensuring the requirements describe / define what the customers and users need the software to do.
- Software requirements should adhere to the following quality standards as they are being developed:
  - It can stand on its own.
  - It can be verified.
  - Is traceable to higher level system requirements.

- For requirement validation techniques:
  - Requirement reviews: Verifies that the requirements are captured accurately.
  - Prototyping: Can be used to find gaps in communication / specification early and demonstrates requirement understanding.
  - Test case development: Used to determine if a requirement is testable and reduces ambiguity.
- Changes in software requirements can ultimately affect:
  - Design
  - Type of data collected
  - Cost and schedule
  - Configuration baseline
- *Scope creep:* Allowing customers to change requirements without revisiting development plans and costs.
- During subsystem integration, we need to ensure integrated component logic is correct.

- For SDLC:
  - Software Requirements Analysis: Decompose system level requirements into specific software requirements.
  - Architecture / Design: Develop models to represent the structure of the software.
  - Implementation: Develop a code base.
  - Test: Perform integration and validation of the software against requirements.
  - Production Support: Maintain rigorous configuration control of all software artifacts applicable to a given software release
- For software process models:
  - Waterfall: Cascades from one phase to another, traditionally resulting in an artifact that is signed off and approved before proceeding to the next phase.
  - V: Each phase closely aligns to a complementary testing phase.
  - Iterative: Requirements are broken into multiple standalone software builds/releases wherein each build will repeatedly undergo different phases of the SDLC before moving on to the next capability release/build.
  - Agile: Breaks the software development activity up into short 2 to 3 weeks sprints in which 1 or more phases of the SDLC may be employed to achieve the goals of the sprint.

- In a few sentences, *explain* how spending time to develop a robust and sound architecture can limit the amount of time required to refactor a system's design in the future.
  - Improves stakeholder communication
  - Supports more thorough system analysis
  - Supports large scale system reuse
- *Explain "how"* each of the "views" in the diagram below captures the software system architecture / design of a software system.
  - LOGICAL: Shows key abstractions in the system
  - DEVELOPMENT: Shows how the software is decomposed for development
  - PROCESS VIEW: Shows how at “run time”, the system is composed of interacting processes
  - PHYSICAL VIEW: Shows the system hardware and software components are distributed across processing elements



- Explain the primary design stages used in developing a software design?
  - Define the context and the modes the system will be used in
  - Design the system architecture
  - Identify the principal software objects
  - Develop design models
  - Specify object interfaces

- In Object-Oriented (OO) design, what is a *'class'*? What is an *'object'*? How do the two differ from one another? How do the two complement each other?
  - Class: Code template for creating objects, Defines the variables and methods, Determines how an object behaves
  - Objects: Self contained component which consists of properties to make a particular type of data useful, Model real world objects you find in everyday life
- Explain why software reuse is important:
  - Designs were very unstructured and difficult to maintain in early days
  - Costs and schedule pressure
  - Common practice now

- Explain what "open-source' software is, how it can help software engineers develop software more quickly. Give at least 4 examples of commonly used open-source software.
  - Open-source software (OSS) is software that is released with a license that allows anyone to use, study, change, and distribute the software and its source code. The copyright holder grants these rights to users.
- Explain what the term "spaghetti code" means. Describe at least 3 strategies used to avoid developing spaghetti code.
  - The phrase spaghetti code dates to the 1970s as a metaphor for sloppy, unmaintainable code.
  - Spaghetti code derives its meaning from the tangled, disconnected mess of intertwined and /or contiguous code segments symbolized by a bowl of spaghetti.
  - planning and preparation, modularity, Naming conventions and comments, Less is more

- Explain what *"requirements traceability"* means. Why is it important?
  - Requirements traceability is the ability to connect requirements to other artifacts, such as software tests or bugs.
  - It's used to track requirements and prove that they have been fulfilled.
- Explain the difference between a *"functional requirement"* and a *"non-functional requirement".* Give an example of both types of software requirements.
  - Functional requirements define how the system must work and non functional requirements detail how it should perform.
- Describe at least 3 checks that can be used to evaluate the "goodness" of a software requirement.
  - Validity checks
  - Consistency checks
  - Completeness checks
  - Realism checks
  - Verifiability checks

- Define / describe 3 different principles from the ACM's Software Code of Ethics. Give an example of each of the 3 principles that you chose to describe.
  - Public
  - Client and Employer
  - Product
  - Judgement
  - Management
  - Profession
  - Colleagues
  - Self
- Discuss the pros / cons of both *Agile* software development and *Waterfall* software development processes.
  - Waterfall is a better method when a project must meet strict regulations as it requires deliverables for each phase before proceeding to the next one.
  - Agile is better suited for teams that plan on moving fast, experimenting with direction and don't know how the final project will look before they start.