Specifications

* Specification - made by customer for design team
* Specification is designed to be a comprehensive set of documents that allow different kinds of stakeholders to communicate with one another. It's intended to be versatile and include the clients to ensure there are no misunderstandings.
* **Requirement process**
* Requirements define what the system does, not how it does it.
  + eg sorting algorithm, we specify that we want a list to be sorted, we don’t specify it has to be sorted with quick-sort
  + 4 Desirable high level properties:
    - 1. Complete
      2. Consistent
      3. Precise
      4. Concise
  + Two high level categories:
    - Functional:
      * Capture what it is that the system should do
      * eg. user should be able to log out with a single click
    - Non-Functional:
      * Captures properties that the system should have
      * eg. System should be useable, performant or testable
      * Can vary in who they are actually important to, eg customer will care more about usability, performance etc. but development team will care more about reliability, complexity and testability i.e different stakeholders will care to different degrees about the varying non-functional requirements within a system.
      * These are often in conflict in one another. eg adding a lot of cashes will make a system faster but will increase complexity similarly usability and security conflict a lot. Hence these must be discussed early in the specification process
  + Additional requirements
    - Design Constraints: constraints within an organisation that influence the product. eg regulatory framework, internal processes, budgetary constraints
    - Environmental Constraints: relates to external factors eg. external authentication provider, cloud service etc
    - Preferences: Encode the priority of requirements and features
* RequirementElicitation**:**
  + 1. Elicitation: we gather the requirements for the system and figure out what will actually be built.
  + 2. Analysis -> points back to elicitation. we look at the requirements and check them for consistency. Check if there are any concerns that the development team might form.
  + 3. Reification: formally writing down our requirements so that they exist in a durable format. user stories are often used
    - Reification refers to making abstract concepts real or concrete. So once a tentative model is elucidated from the requirements process, it can be mocked up for further verification: we can then examine the model to see if it's viable.
  + 4. Validation: final list of requirements and make sure that they match what the customer wants
* **Validating Requirements:**
* Advantages of validating:
  + Reduced need to go back to elicitation and analysis phase
* Better ways to validate requirements:
  + use measurable objectives, they’re easier to achieve
* A way to:
  + concretely evaluate the features added to our system
  + make sure they perform as expected
  + are correct
* **User Stories:**
  + way of capturing requirements in a format that's useful for engineers and customers
  + These are short concrete documents that capture:
    - who a feature is for,
    - what its value is,
    - what’s involved in creating that feature,
    - what its costs are,
    - how long it will take to build.
  + These documents provide the customer a really concrete understanding of what it is that the development team is going to do when they’re building out a feature
  + They also provide data back to the customer so they can understand the overall costs and trade offs of adding a feature to their system
  + They increase cohesion between the customer/product owner and the development team
  + Widely used in agile methodologies
  + Have 5 main parts:
    - Role-Goal benefit: Forces the customer to really think about:
      * Who is going to benefit from a feature (role)
      * What they’re trying to achieve (goal)
      * Why they want to achieve that (benefit)
      * eg: a statement that captures the Role-Goal benefit format is:
        + As an online learner, I want to be able to read reviews about online courses, so that I can decide which course I want to enroll in.
        + Role: learner, Goal: to be able to read and write reviews, Benefit: decide which course to enrol in
      * The role can also be a non-human entity eg backend systems, third party servers etc
    - Limitations: scope down the role goal benefit to apply it only in the subset of situations that matter for this feature
    - The definition of done:
      * Giving the dev team a concrete understanding of how the customer will be validating whether or not the feature being delivered is actually done, complete and correct.
      * It should be easily verifiable so the client knows what the expect, and the person fulfilling the user story can be validated that they've completed the task. Therefore, we want this to be as clear as possible, without ambiguities. Must not refer back to a previous part of the user story.
    - Engineering tasks: Keep track of how this feature interacts with other features within the system
      * Engineering notes should discuss at a technical level any notes to bear in mind or steps to take; they should not describe desirable traits or make estimates on time.
    - Effort estimate:
      * Overall cost of a feature
      * Compare effort/cost vs value
    - Cost needs to be implementable in a single iteration
* **Invest guidlines:**
  + Independence: All user stories should be independent from each other as much as possible.
  + Negotiable
  + Value: User story must add a meaningful value for the product
  + Estimable: We must be able to provide concrete effort/cost for creating a feature
  + Small: User stories should be small
  + Testability:
* **Decomposing user stories:**