**The product vision**: The starting point for software product development is a ‘product vision’. Product visions are simple statements that define the essence of the product to be developed. **The product vision should answer three fundamental questions:** What is the product to be developed?Who are the target customers and users?Why should customers buy this product?

**Intro to requirement engineering** One of the most tedious jobs at all (for Software Engineers)**,** It is one big document “Requirement Specification”, Senior Design Report/Document**,** In an Agile context **not needed,** But for specific projects, **no choice!,** Mistakes mean Vendor will love you (Change Request = money)

**Requirements Specification:** A contract between the client and a developer(s)**,** Describes **what** the system has to do**,** Does **not** describe **how** the software is to be constructed**,**

If some requirements are **missing**, specification is **incomplete**

If some requirements are **conflicting**, specification is **ambiguous** or **inconsistent**

Often divided into: **High-level** requirements **->** More **readable** overview**,** Outlines **why** the system is being built, **benefits** to be achieved**. Low-level** (or detailed) requirements **->** Highly **detailed** descriptions of work the system **will perform** Often evolve as the **requirements are analyzed**, or as the system is **being developed.**

Maintaining requirements specification over time is **critical and difficult** (Some kind or archive is needed => Confluence?!)

Nonfunctional requirement categories; **Quality attributes** (…bilities)-> Reliability and availability, Performance (response, throughput, storage use), Security, Maintainability, Portability **Constraints** -> Not enough time, Not enough money, No security experience **External interfaces ->** Hardware**,** Other software**,** External agents **User interfaces ->** Specific guidelines to follow **Error handling ->** What to do for: (Warnings, Errors, Exceptions)**,** Detecting errors**,** Gathering errors

**Requirements Elicitation:** Heavy involvement with **client,** Strongly focused on **business needs** and issues

* **What** not **how**
* People (Business Analyst) who do this often, have a business background
  + Need great **communication** skills
  + Need to understand the **business domain** and how the system **will be used**
  + Need to understand **what’s possible** and **lead** the client discussion

**High-Level Requirements**

* Define **sponsors**
* **Business case** to justify system
* Scope definition
* Major **constraints** on development and deployment (Cost, sites, schedule…)
* Major **functionality**
* Success definition

**Low-Level Requirements**

* Usually deal with **technical** concerns
* Tradeoffs among issues
* Best practice is to write tests for every requirement
* Each requirement needs to be **traceable**
  + Uniquely identifiable (testable)
  + Initial documents to requirement
  + Elements of system to requirement
* Requirements are often in the context of existing systems
* Functionality in the context of business flow
* Interfaces
  + Error reporting
  + With existing systems
    - Control and data transfer to and from, retry on error
* Other constraints are usually non-functional

**Prioritizing Requirements**

Factors that affect prioritization:

* + Resources
* Time
* Customer demands
* Future customer needs
* Competition and market conditions
* Problems in an existing product
* Many schemes, but for this class we’ll use three categories
* Absolutely necessary/critical
* Highly desirable/recommended
* Possible/suggested

**Requirements Definition, Prototyping, Review**

* Client is involved in all aspects
* Often done in conjunction with requirements analysis
* **Definition** involves formally writing out the requirements
  + Organizations typically have some standard format
* **Prototyping** is trying simple versions of some aspects of the system to gain clarification
  + Typically, user interfaces
  + Low-fidelity: paper or cardboard markups, sketches on whiteboard, etc.
  + High-fidelity: actual screens created using tools like visual basic, html, etc.
* **Review** is done with client to validate the requirements

**Requirements Specification**

* Client is not involved
* The specification is what client will sign off on
  + “If you build what this describes, I will buy it”
* Some researchers distinguish between
  + Requirements document (high-level, **client-speak**)
  + Requirements specification (low-level, **design-speak**)
* Usually uses a standard format and language
* IEEE/EIA standard contents
  + Introduction
  + High-level description
    - General description, major functions, user characteristics, , major constraints, dependencies
  + Detailed requirements
    - Each functional requirement by input, process, and output
    - Interfaces
    - Performance requirements
    - Standards compliance and system constraints
    - ities (reliability, availability, recoverability, …
    - Unique requirements
* IEEE Standard 830-1998 provides templates of several different ways to categorize requirements
  + External interfaces
  + Functional requirements
  + Performance requirements
  + Logical database requirements
  + Design constraints
  + Software system attributes (nonfunctional)
* In practice, categorization often depends on application area and system architecture or implementation

**Analysis Models**

Can also group modeling approaches as

* + Scenario-based models: Unified modeling language (UML):
  + Use case diagrams
  + Activity diagrams
  + Swim lane diagrams
  + Etc.
* Data-based models
  + Entity-relationship diagrams

**Text Use Cases**

* More like a business task
  + Description of a task
  + Precondition/postcondition
  + Workflow
  + Priority
  + User Stories (Senior Design/Agile)
* A screenshot of a computer program

  Description automatically generated

**User Stories Format**

* As a **[PERSONA]**, I want to **[ACTION]** so that I can **[MOTIVATION]**
* As a **teacher**,   
  I want to be able to **create a group** of students and teachers,   
  so that I can **share information with that group**.

**Common mistakes in user stories**

* A story doesn’t tell developers **what** code to write.
* It is a description of the **problem** we are trying to **solve**.
* Should be able to complete within one sprint, preferably 1-2 days.
* User stories will be translated into the **Product Backlog**

**UML Use Case Diagram**

**Text Use cases describe how an “actor” interacts with the system**

* + **What things that actor can do**
  + **Steps required to accomplish each action**
  + **Typically employ use case diagrams, narrative text, pre- and post-conditions, exceptions**

A diagram of a person's diagram

Description automatically generated

A diagram of a restaurant

Description automatically generated

**Activity Diagrams**

* Show dynamic behavior of   
  (part of a) system
* Like flowcharts except can have parallel actions
* A diagram of a group of ideas

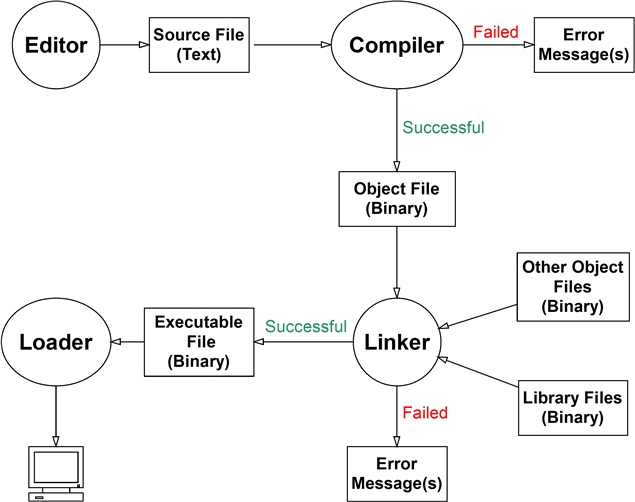
  Description automatically generated

**Swim Lane Diagram**

* Activity diagram showing actions by different categories of actors
* Not a Sequence Diagram!
* A diagram of a process

  Description automatically generated

**Data Flow Diagrams**

* Indicate inputs, transformations on those inputs, and outputs
* 

**State Transition Diagrams**

* Good for describing behavioral models
* A diagram of a broken bottle

  Description automatically generated

**Quality Attributes:**

**Quality Metrics in Requirements**

* Quality in a requirements specification is based on how well the specification expresses client wants and needs
* Collect a number of values to determine quality
* Called metrics
* Most valuable when target values (indicating when adequate quality is reached) are determined in advance
* QA organization is often included in requirements analysis

**Attributes that Promote Quality**

* Requirements are **accessible**
* Specification is **comprehensive**
* Each requirement is **understandable**
* Each requirement is **unambiguous**
* Specification is **consistent**
* Requirements are **effectively prioritized**
* Requirements are **secure**
* Specification is **self-complete**
* Each requirement is **testable**
* Each requirement is **traceable**

**Accessibility**

Need numbering scheme to identify requirements

* + Allow determining if requirement has been implemented
* Allow tracing to code

Considerations

* + Know where high-level and detailed requirements are listed
* Detailed requirements are grouped by relevant high-level requirement
* Can requirements be searched by relevant factors
  + Keyword, use case, interface type, user type,…

Good metric is average time to access a detailed requirement

* Choose random sample of >= 150 requirements
* Time people trying to find sample requirements
* Score based on performance compared to organization norms
  + 0 score for “extremely long” access times
  + 10 score for “extremely fast”

**Comprehensiveness**

* % of total requirements specified
* Could have client evaluate this but
  + Time commitment is a problem
  + Contradictory stakeholder desires a problem
* If system is implemented iteratively, detailed requirements for later iterations   
  may not yet be done
  + Can summarize requirements that haven’t been detailed yet
* IEEE defines completeness using formula with 18 different quantities and 10 weights

**Understandability**

* Measure by taking random sample of relevant people (clients and designers) and asking opinions
* Considerations to improve understandability
  + Written to level of typical reader understanding?
  + Use vocabulary of the client problem domain?
  + Describe only external behavior?
  + Don’t say how to solve problem or how to design system

**Unambiguity**

* Sample set of readers, have each score each requirement
  + 2 if exactly one clear meaning
  + 0 if multiple possible interpretations
  + 1 otherwise
  + Sum ambiguity measure over all detailed requirements, divide by 2\*number of requirements

**Consistency**

* One approach
  + Sample >= 150 detailed requirements
  + Evaluate each against ALL other requirements to see if another requirement contradicts it
  + Score 0, 1, 2
* Measures only pairwise inconsistency
  + Also possible to have inconsistency among higher order groups

**Prioritization**

* Prioritize after first release of system, since clients **often change their mind**  
  (Prototype is helpful)
* Rank requirements as high, medium, or low priority
  + Good prioritization would have each group about the same size
  + Calculate variability measure

**Security**

* Some security requirements are different – they say what the system **should NOT DO**
  + An automated user of the application enters a known user ID and more than 10 passwords per second
  + A user accesses more than 30 customer records in a single session and transmits those to another address within 10 seconds of access
* As opposed to
  + All passwords shall be unavailable except to the system administrator

Some things to check:

* + Evaluate the system for places where intrusion might be possible,   
    and verify that concrete security requirements are identified
  + Has user identity and password security been specified
  + Has ownership of files and data access been specified
  + Has encryption been specified where appropriate
  + Has protection from known security exploits been specified (e.g., SQL injection)

**Self Completeness**

* Requirements specification contains every requirement necessitated by any other requirement
  + For calendar app, “app will retain all information entered for each appointment”
    - Implies requirement describing how information is entered
  + Need to follow implication chains to verify
* Possible metric
  + # of missing necessary requirements/# of requirements

**Testability**

* Can’t determine whether untestable requirements have been met
* Include tests for requirements and   
  count percentage of requirements accompanied by tests
* Checklist to improve testability includes
  + Is requirement clear enough to identify test data and actions?
  + Would two different client representatives agree that proposed tests really do test the requirement?

**Traceability**

* Forward traceability from requirement to
  + Design element that accommodates it
  + Code that implements it
  + Test that verifies it is satisfied
* Metric might be to rate traceability of each requirement on a 0 – 2 scale
  + Divide sum of ratings of all requirements by 2\*total number of detailed requirements