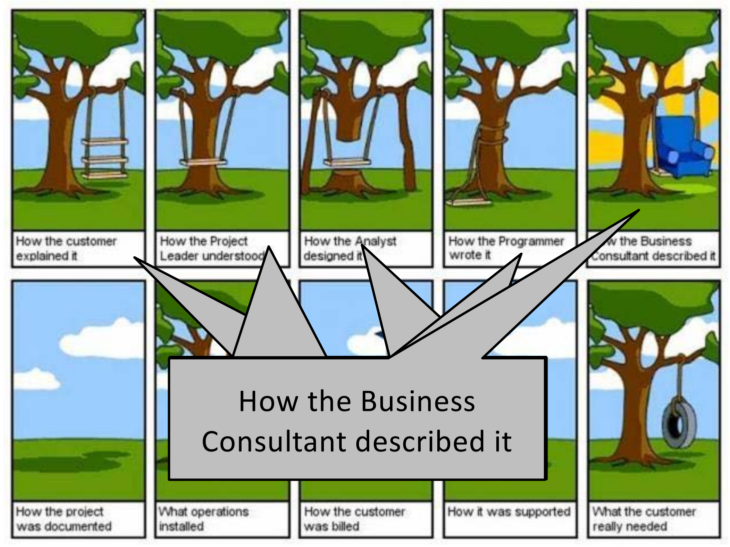
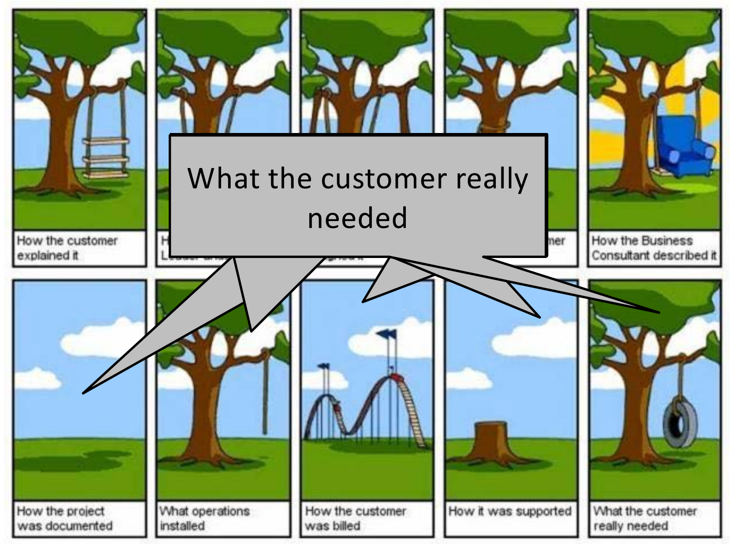
CS 326 Requirements

Stephen Freund


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
w the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed
How the Business Consultant described it


How the customer explained it
What the customer really needed
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Requirements

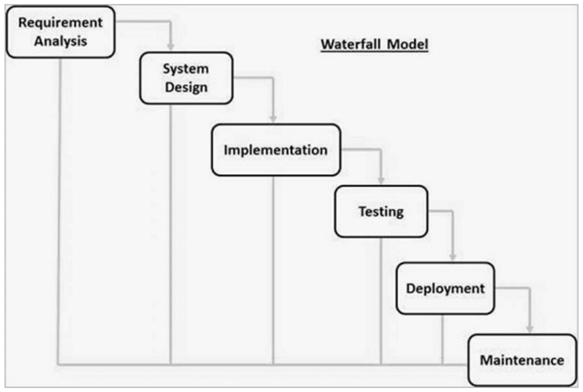
- A necessary function, attribute, capability, characteristic, or quality of a system.
- Build the System Right
  - everything until now
- Build the Right System
  - today

How To Build A Bridge

1. Determine the **requirements** the bridge must meet.
2. Make a complete **design.**
   – blueprints, materials, building method, etc.
3. Build (**implement**) it.
4. **Test** it to make sure its going to work.

Waterfall Model

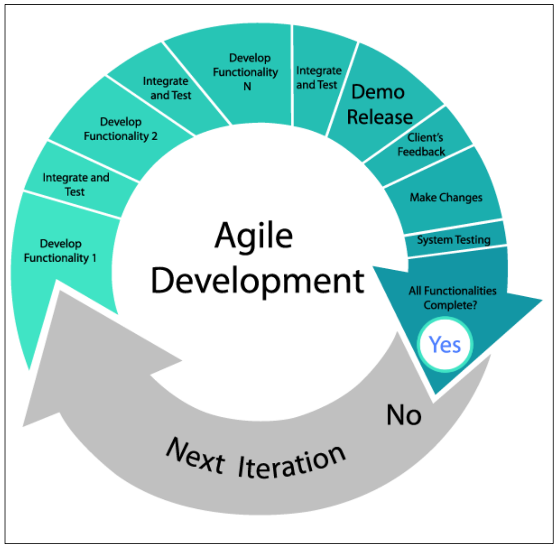
- Foundation for all software design methodologies

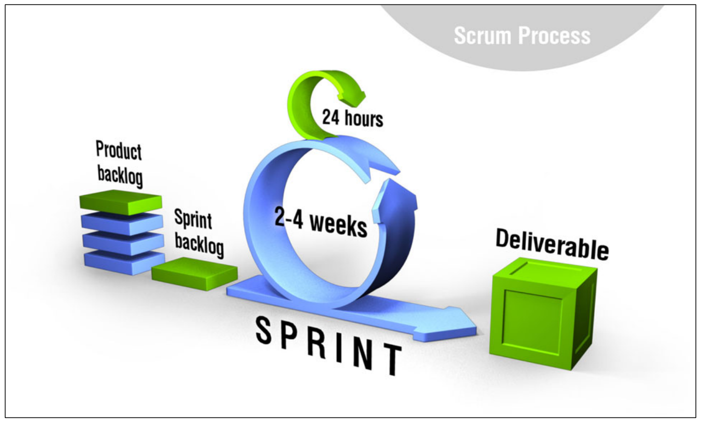


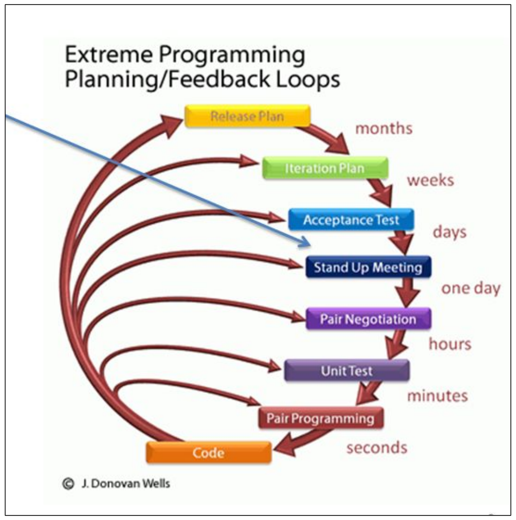
Contemporary Software Engineering

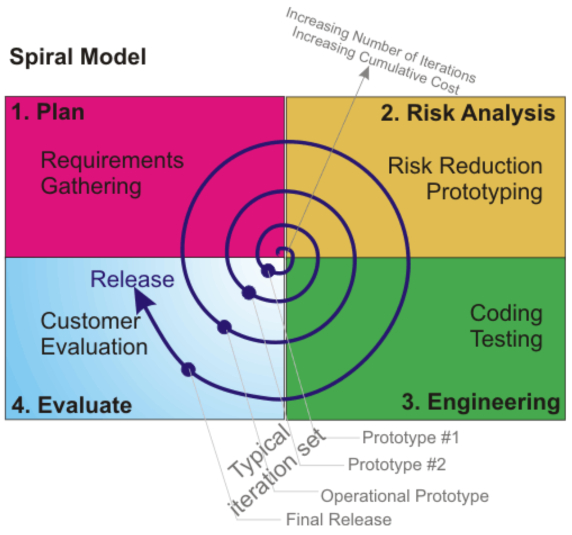
Can be far more flexible due to zero materials cost.

1. Make first pass at requirements and design.
   – Will not be complete.
2. Implement key features and basic functionality.
3. Use initial implementation to refine the requirements and design.
4. Implement more features, continue to refine requirements/design, and provide continual releases of the running application.









Requirements

- **Who are the stakeholders?**
- **Customers**: show what should be delivered; contractual base
- **Managers**: a scheduling / progress indicator
- **Designers**: provide a spec to design
- **Programmers**: list a range of acceptable implementations
- **QA / Testers**: a basis for testing, validation, verification

Digging For Requirements

- Do:
  - Talk to the users (or work with them) to learn how they work.
  - Ask questions throughout the process to "dig" for requirements.
  - Think about why users do something, not just what.
  - Allow (and expect) requirements to change later.
- Don't:
  - Describe complex business logic or rules of the system.
  - Be too specific or detailed.
  - Describe the exact UI used to implement a feature.
  - Try to think of everything ahead of time. You will fail.
  - Add unnecessary features not wanted by the customers.

cultural anthropology!

Positive/Negative Examples

- The system will enforce sales tax on all purchases.
- The system shall display the elapsed time for the car to make one circuit around the track within 5 seconds, in hh:mm:ss format.
- The product will never crash and be secure against hacks.
- The server backend will be written using PHP or Ruby on Rails.
- The system will support a large number of connections at once, and no user will experience slowness or lag.
- The user can choose a document type from the drop-down list

Requirements Give Shared Vision

- Software project involving *n* people starts out with *n* different impressions on how the app should be structured.
- Primary goal is to unify those *n* visions into one cohesive vision.
  - Extensive discussion, questioning, sketching.
  - All parties involved need to be at the table: management, developers, **customers**.

Requirements Docs: Your Vision

- **Vision Statement**
  - a paragraph or two on the shared vision of customer and developers on what the app should do.
- **Feature List**
  - A bullet list of features the application should have
- **Sketches/storyboards**
  - user interfaces and how they are used
- **Use cases**
  - step-by-step sequences of how a particular scenario of user-app interaction should play out

Requirements Docs: Your Vision

- **Domain Analysis**
  - Understand the underlying domain of the app
- **Software Architecture**
  - Not really a requirement...
  - But good think about top-level architecture early on.

Other Activities

- **Prototypes**
  - verify feasibility of core requirements
  - ensure libraries/frameworks will do what you need
  - Tracer bullets, Prototyping, Requirements Pit (PP)
- **User Perspective**
  - Identify needs of users (actors) that will be interacting with the system

Big Picture Questions

- What is the market for the app?
- Who will actually use it?
- Do similar apps already exist?
- What is the feasibility of coding it in the available time frame?
- Can it logically be split amongst the team?
- What is the team organizational structure?
  - Strong leader, collection or leaders, or highly democratic?

Vision Statement

- What is the basic problem or opportunity?
- What is your app's primary goal?
- How does the app achieve your goal?
  - What are the primary features?
  - Who are the intended users?
  - What outcomes do you expect?
- Refer back to this to remain focused on primary objectives.

Feature List

- Simple bullet list of single-sentence descriptions
- Describe from users perspective
- Identify: core vs optional/extended features
- Should be self-consistent
- Avoid Feature Creep
  - Gradual accumulation of features as project progresses
  - PP: Stone Soup, Boiled Frogs, ...

Student Services Porthole Features

- Transcript
  - Calculate the average grade on a transcript.
  - List the classes of a student on a transcript.
  - Display the name/address of a student on a transcript.
- Enrollment
  - List the prerequisites for a class.
  - Enroll a student in a class.
  - Drop a student from a class.
  - Add at student to a class waiting list.
- Meal Plans
  - List available meal plans
  - Calculate fees for meal plan

CampusPaths App Features

More details in CampusPaths handout Appendix

- Show Map
  - Displays map of campus with buildings marked.
  - User can navigate around the map.
- Finding Paths
  - User can select two buildings.
  - The shortest path between selected buildings is shown.
  - *[Optional] Alternative paths are also shown.*
- Detailed Instructions
  - User can view shortest path as step-by-step instructions.
  - Each step conveys a distance and compass direction.
  - *[Optional] The steps are described from the user's perspective (eg: turn left)*

Automatic Dog Door Features

From "Head-First Object-Oriented Analysis & Design"

- The dog door opening must be at least 12" tall.
- A button on a remote control opens the dog door if it is closed, and closes it if it is open.
- Once the dog door has been opened, it closes automatically if the door isn't already closed.

Use Cases

- Sequences of events that represent behaviors the functioning system should have.
  - **Actor**-centric, not code-centric
- Benefits:
  - Writing use-cases exposes details of expected functionality.
  - Can incrementally elaborate (or shrink/remove) use-cases until they are stable and sensible enough to implement.
  - Can help guide initial coding.

Dog Door Use Case, Version 1

1. Fido barks to be let out.
2. Todd or Gina hears Fido barking.
3. Todd or Gina presses the button on the remote control.
4. The dog door opens.
5. Fido goes outside
6. Fido does his thing.
7. Fido goes back inside.
8. The door shuts automatically

Dog Door Use Case, Version 2

1. Fido barks to be let out.
2. Todd or Gina hears Fido barking.
3. Todd or Gina presses the button on the remote control.
4. The dog door opens.
5. Fido goes outside
6. Fido does his thing.
   - 6.1 The door shuts automatically.
   - 6.2 Fido barks to be let back inside.
   - 6.3. Todd or Gina hears Fido barking.
   - 6.4. Todd or Gina presses the button on the remote control.
   - 6.5. The dog door opens.
7. Fido goes back inside.
8. The door shuts automatically

Optional Path: may or may not happen

Automatic Dog Door Feature List

From "Head-First Object-Oriented Analysis & Design"

- The dog door opening must be at least 12" tall.
- A button on a remote control opens the dog door if it is closed, and closes it if it is open.
- Once the dog door has been opened, it closes automatically if the door isn't already closed.
- **Door opens automatically when Fido barks.**

Dog Door Use Case, Version 3

Alternative Path: can replace steps on main path

Main Path

1. Fido barks to be let out.
2. The bark recognizer "hears" a bark.
3. The bark recognizer sends a request to the door to open.
4. The dog door opens.
5. Fido goes outside
6. Fido does his thing.
   6.1 The door shuts automatically.
   6.2 Fido barks to be let back inside.
   6.3. The bark recognizer "hears" a bark.
   6.4. The bark recognizer sends a request to the door to open.
   6.5. The dog door opens.
7. Fido goes back inside.
8. The door shuts automatically

Alternative Paths

2.1 Todd or Gina hears Fido barking.
3.1 Todd or Gina presses the button on the remote control.
6.3.1 Todd or Gina hears Fido barking.
6.4.1 Todd or Gina presses the button on the remote control.

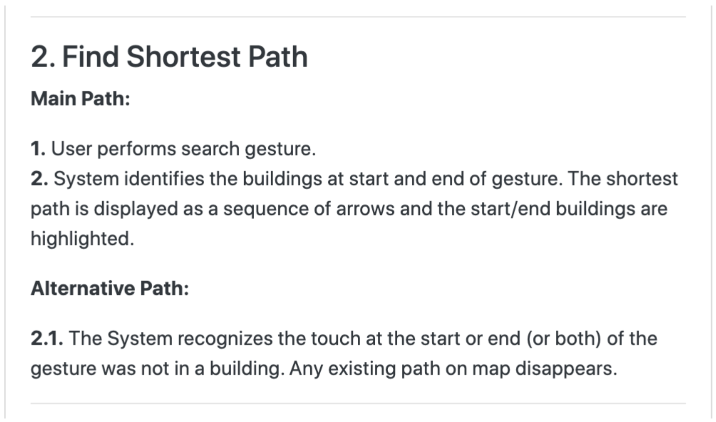
Customer Purchase Example

http://ontolog.cim3.net/cgi-bin/wiki.pl?UseCasesSimpleTextExample

1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

- Alternative: **Authorization Failure**
  – At step 6, system fails to authorize credit purchase
  – Allow customer to re-enter credit card information and re-try
- Alternative: **Regular Customer**
  – 3a. System displays current shipping information, pricing information, and last four digits of credit card information
  – 3b. Customer may accept or override these defaults . Return to primary scenario at step 6

CampusPaths: Find Shortest Path



2. Find Shortest Path

Main Path:

1. User performs search gesture.
2. System identifies the buildings at start and end of gesture. The shortest path is displayed as a sequence of arrows and the start/end buildings are highlighted.

Alternative Path:

2.1. The System recognizes the touch at the start or end (or both) of the gesture was not in a building. Any existing path on map disappears.

Finding the Right Use Cases

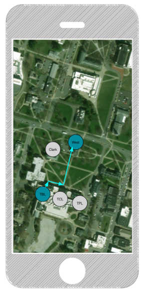
- Use feature lists, GUI sketches, etc.
- Focus on non-trivial features or behaviors.
- Features vs Use Case:
  – *Feature* of a library book checkout application: "the ability to charge late fees for overdue books".
  – *Use-case*: "return book and calculate late fee"
    1. a patron returns a physical book,
    2. the book would be scanned,
    3. it would be marked as available in the database, and
    4. a late fee if any would be computed and added to the patron's account.

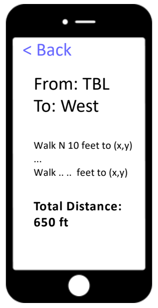
Format For Use Cases

- **Title** that is a capsule summary
- **Main Body** is the primary path for the use-case
  - a numbered list of actions described in English
- Optional **Alternate Paths**
  - other ways the use case could proceed (take either 2. then 3., or take 2.1 then 3.1).
- Optional **Optional Paths**
  - some steps that could be skipped (steps 6.1-6.5)
- Only include alt/opt paths if they give a more clear specification than making separate use-cases.

UI Sketches

- You don't have to make them pretty
  - but it does help make the app feel real if you put some time into them



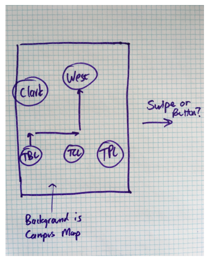


< Back

From: TBL
To: West

Walk N 10 feet to (x,y)
...
Walk .. .. feet to (x,y)

Total Distance:
650 ft



UI Sketches

- Carefully go through features and use-cases
  - make sure you covered features/uses with sketches.
- UI prototyping and storyboarding tools
  - many exist
  - let you sequence a path through the different UIs
  - help with Use Case sequences

Domain Analysis

- Understand the underlying domain of the app before you begin to design it.
- Examples:
  - Music Streaming: Music file formats and Copyright rules
  - Restaurant: Orders, Menu Items, Tables, Payments, Receipts, Waitron, etc
  - Social Network: relationship groups? Friends? Family? Friends-of-Friends? Visibility?

Iterative Process

- Real world:
  - multiple rounds of requirements gathering
  - refine ideas, vision, features, use cases, etc.
- Continue to refine requirements even as move to later stages.
  - early prototypes point to flaws in vision.
  - features become more/less important.
  - clients change their minds…

Requirements Summary

- Vision Statement
- Feature List
- Sketches/storyboards
- Use cases
- Domain Analysis
- Software Architecture (at least in 326…)