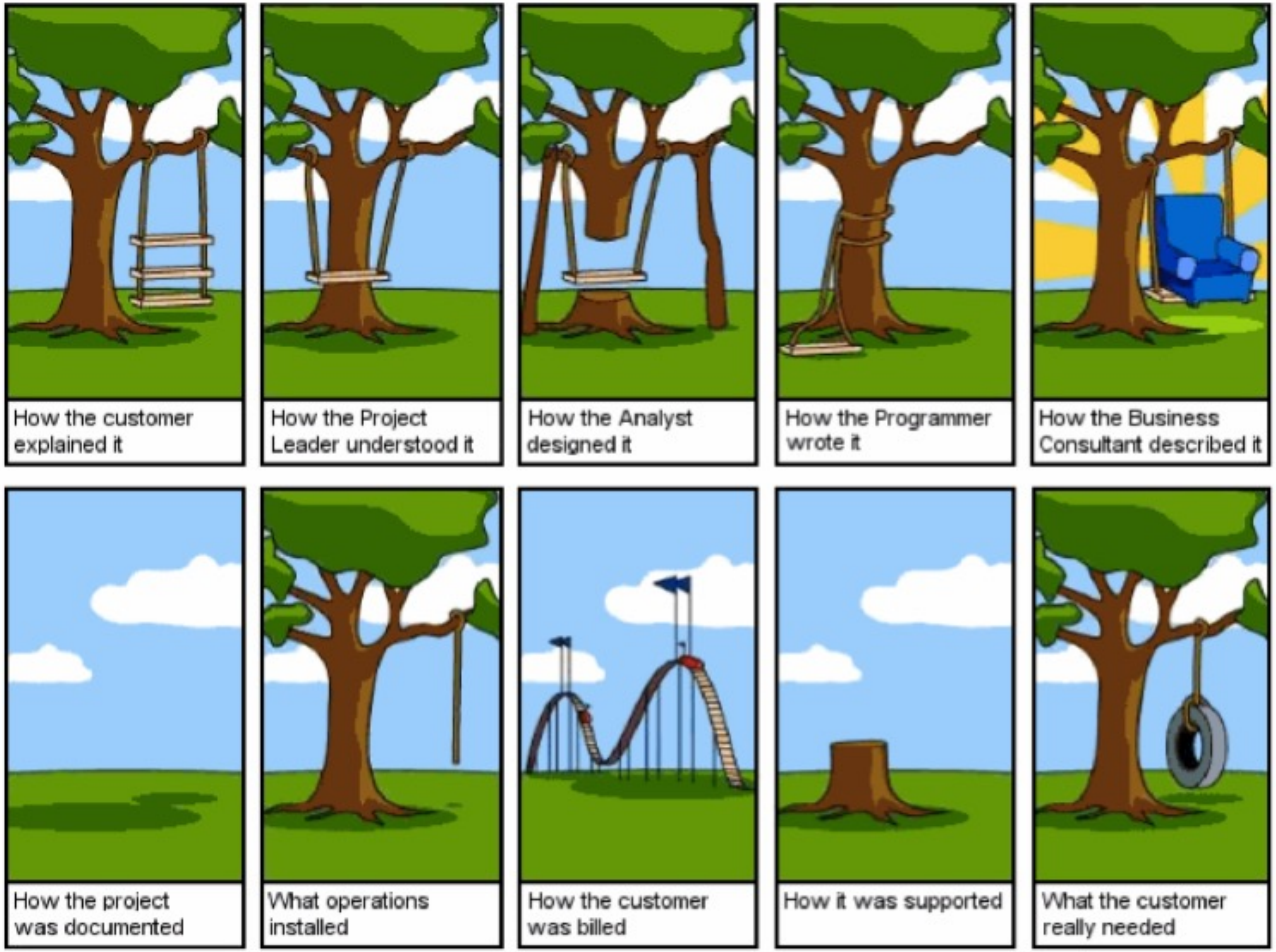

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Software Development Prerequisites

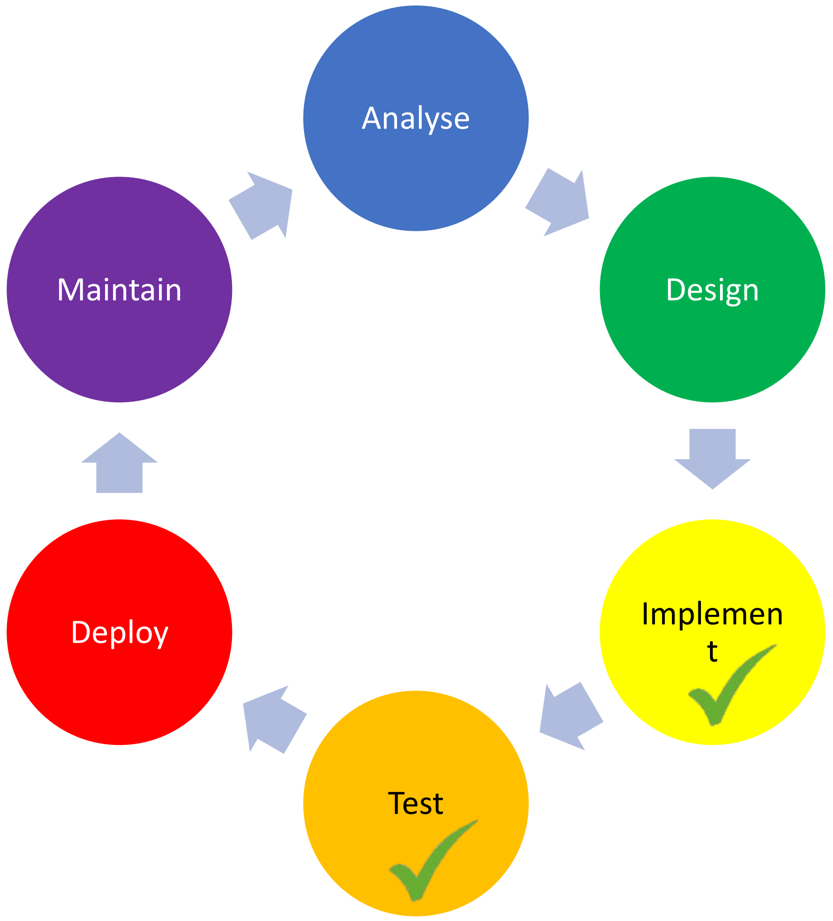
CSCI 2134: Software Development

Agenda

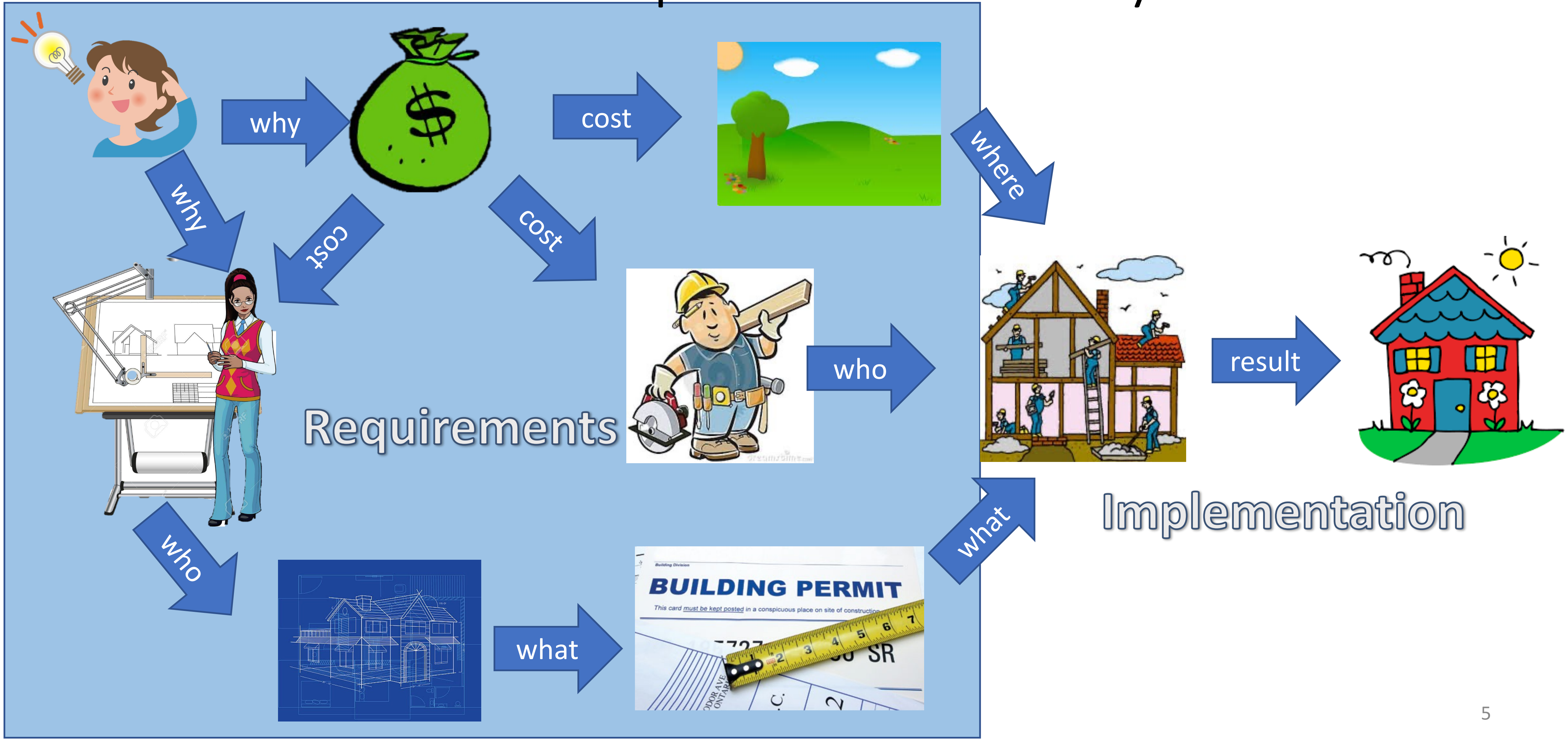
- Lecture Contents
  - Motivation
  - Software Pre-requisites: What and Why
  - Software Processes: Sequential vs Iterative
  - Problem Definition
  - Software Requirements
  - Architecture: The Bones of the Design
- Brightspace Quiz
- Readings:
  - This Lecture: Chapter 3
  - Next Lecture: Chapter 4

Where Are We So Far?

- Requirements gathering
- Design
- Implementation
- Quality Assurance
- Deployment
- Maintenance

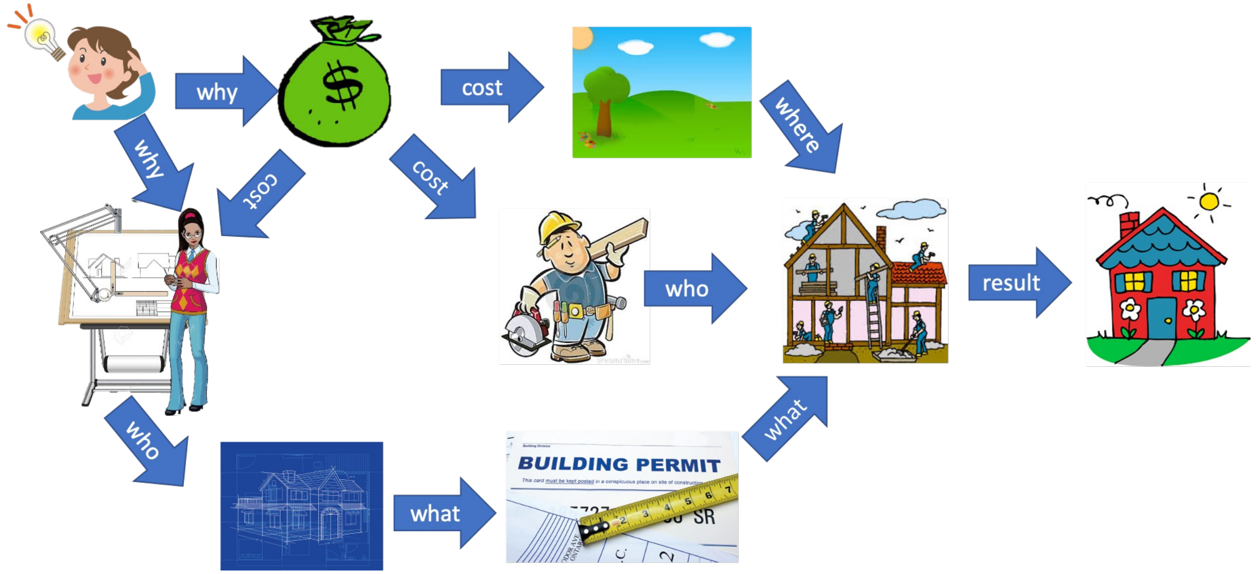


Motivation for Requirements Analysis



Process is Important

- Software development is as much about process as programming
  (For better or for worse.)
- Getting the process right ensures a more productive implementation phase
- Process is boring, unexciting, repetitive, and seems redundant
- But, it is necessary.
  Warning: possible boredom ahead!

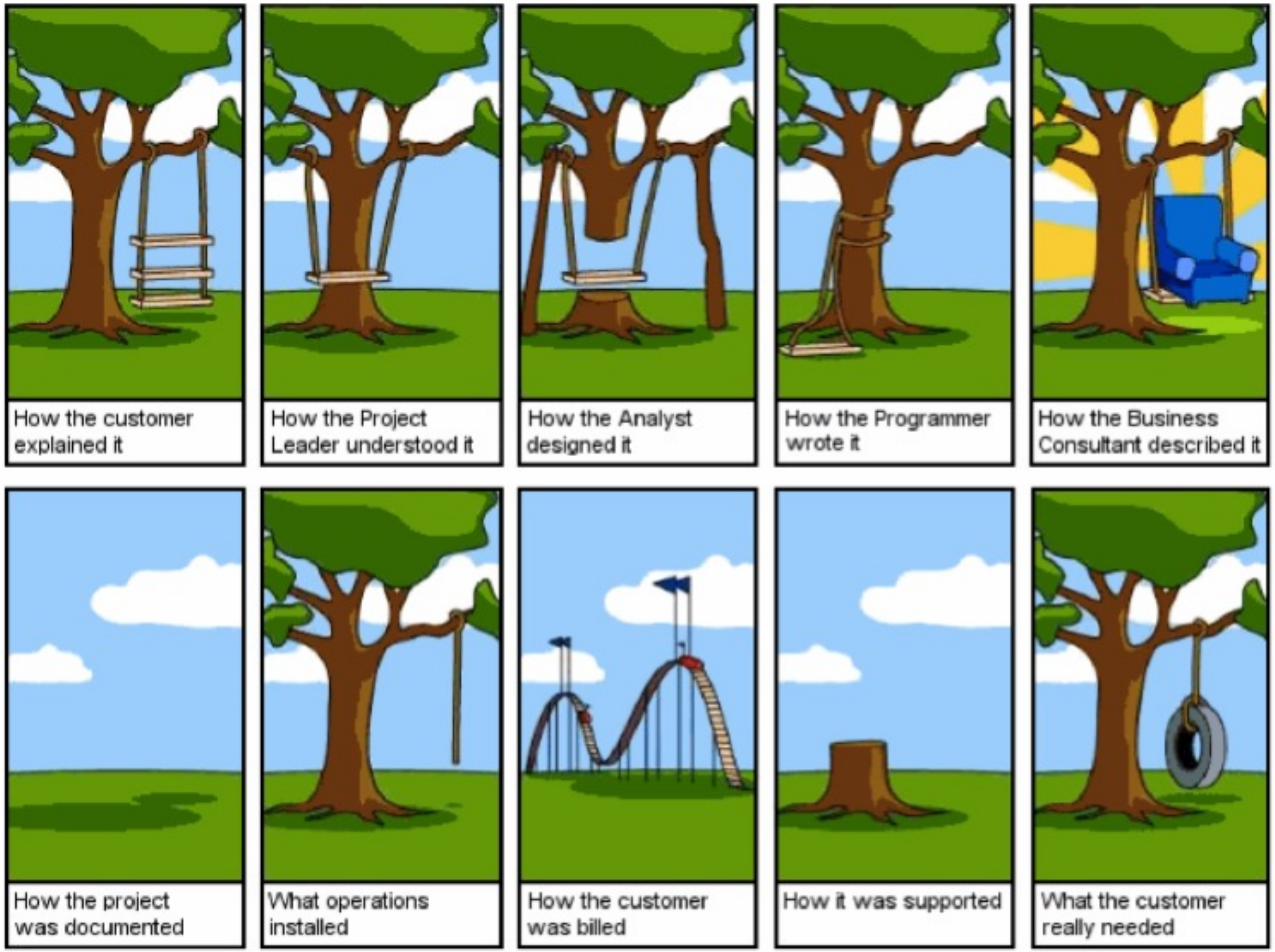


Pre-requisites: What Do We Need to Know?

- What type of software are we building?
  - Business system
  - Mission-critical
  - Life-critical
- What approach are we using?
  - Sequential
  - Iterative
- What problem are we solving? (Why are we building this software?)
- What are the requirements the software must meet?
- What architecture are we using?

  What are the design constraints?
- What tools will we need?
- What can go wrong? (What are the risks?)
- How do we know when we are done?
  - How do we know we have met the client’s requirements?

Why Do I Care?


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
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

The Software Pre-requisite Checklist

- Determine type of software being built
  - Business system
  - Mission-critical
  - Life-critical
- Select appropriate development process
  - Sequential
  - Iterative
- Define the Problem
- Determine the requirements
- Select architecture and determine corresponding constraints

What Kind of Software Are We Building?

- Key Idea:
  - Different kinds of software requires different approaches
  - Different approaches perform requirements analysis differently
- Before beginning, identify:
  - Type of software to be built
  - Type of approach to be used

Common Types of Software Projects

(McConnel, CC2, 2004)

	Business Systems	Mission Critical Systems	Embedded Life-Critical Systems
Typical Applications	• Internet site • Intranet site • Inventory management • Games • Management information systems • Payroll system	• Embedded software • Games • Internet site • Packaged software • Software tools • Web services	• Avionics software • Embedded software • Medical devices • Operating systems • Packaged software
Cost of things going wrong	Minor costs or delays resulting in minor financial losses	Significant costs resulting resulting in major financial losses	Potential loss of life or major injury
Development Process	Iterative	Sequential (sometimes)	Sequential

Types of Software Development Processes

Sequential Development Process

- Do each step of software development to completion before going to the next step
- Tends to have strict deliverables at the end of each step so we know what was decided and we don't revisit it.

Iterative Development Process

- Do a part of each development step per iteration

  End with something working

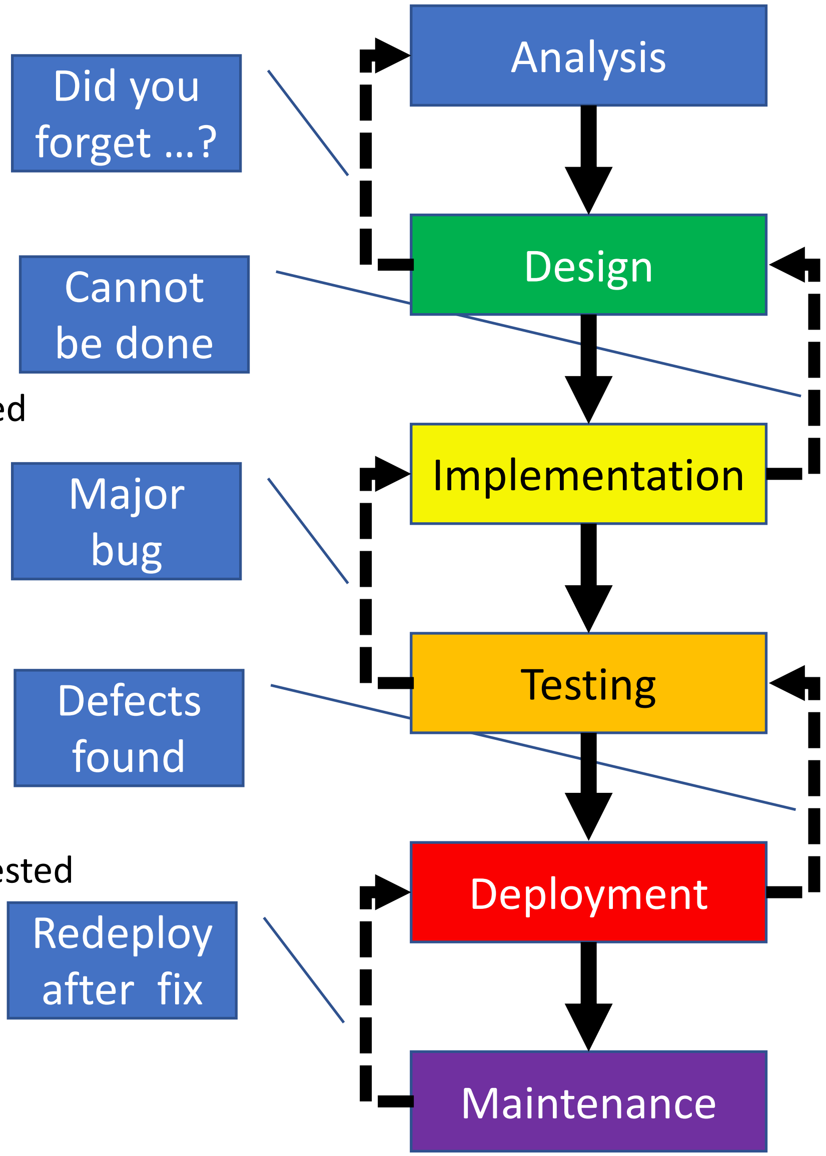
- Assess progress so far
- Determine what part to do next
- Repeat until finished

- Similar to test-driven development

Sequential Development (Also called Waterfall)

Idea: Each stage is completed before the next commences.

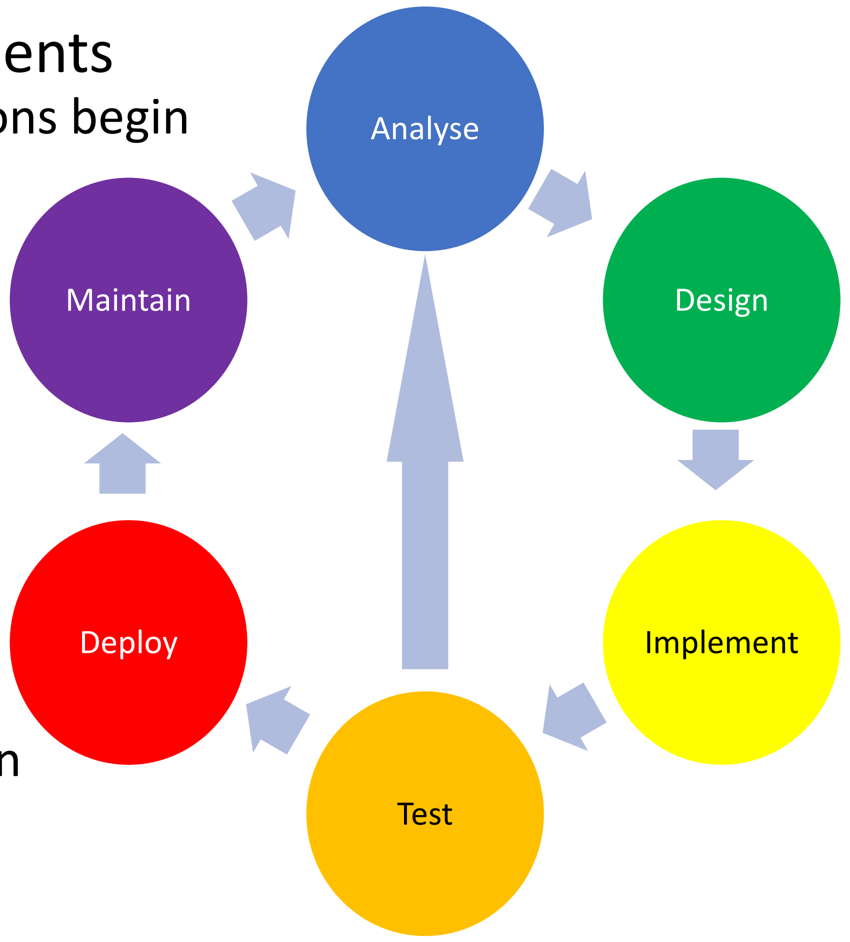
- Analysis:
  - All requirements are elicited and documented
  - Architecture and constrains are determined and documented
  - Once this stage completes, changes to requirements are strongly discouraged
- Design:
  - All major classes, components, packages, APIs, interfaces, etc are specified
  - Design is reviewed for completeness to ensure it meets all requirements
  - Once this stage completes, major design changes are difficult to make
- Implementation:
  - The design is implemented in full
  - The implementation is reviewed
  - Any defects discovered during the implementation are fixed
  - Once this stage completes, all components are implemented but not fully tested
- Testing
  - The implementation is fully tested by both developers and testers
  - Discovered defects are fixed by the developers
  - Once this phase completes the software is released
- Deployment
- Maintenance



Iterative Development Process

- **Idea:** Iteratively develop one feature or use case at a time
- Perform initial analysis to determine general requirements
  - About 80% of the analysis should be done before the iterations begin
  - Further analysis is done in each iteration
- In each iteration
  - Select a feature to implement for the software
  - Analyze requirements for the feature

    All the general constraints are already determined
  - Design the feature
  - Implement the feature
  - Test the feature
    - Do regression testing
    - Test-Driven Development fits well here
  - Additional analysis and design can follow in the next iteration
- Deployment
- Maintenance



Comparison of the Sequential and Iterative

Sequential Development Process

- Advantages
  - All planning is done in advance
  - Predictable time-line
  - Gating: Each phase is completed before next can commence
- Disadvantages
  - All planning is done in advance
  - Does not deal well with changing or evolving requirements
  - Stages can take a long time
    - Need to get everything right before the next stage commences

Iterative Development Process

- Advantages
  - Some planning can be deferred till later
  - Can accommodate (some) changes to the requirements and design
  - Stages take a short time
  - Complements strategies such a test-driven-development
- Disadvantages
  - Harder to plan predictably
  - More unknowns

Which Process Should We Use? … It Depends.

(McConnel, CC2, 2004)

Sequential Development Process

- The requirements are fairly stable.
- The design is straightforward and fairly well understood.
- The development team is familiar with the applications area.
- The project contains little risk.
- Long-term predictability is important.
- The cost of changing requirements, design, and code downstream is likely to be high.

Iterative Development Process

- The requirements are not well understood or you expect them to be unstable for other reasons.
- The design is complex, challenging, or both.
- The development team is unfamiliar with the applications area.
- The project contains a lot of risk.
- Long-term predictability is not important.
- The cost of changing requirements, design, and code downstream is likely to be low.

Key Idea: The Iterative development process manages risk of the unknown, by building it into the process

Software Requirements

- Regardless of whether we are using sequential or iterative approaches, we still need to know what needs to be built
- The first step is to define what problem we are trying to solve
- If we don't understand the problem, the rest of the process won't make sense.

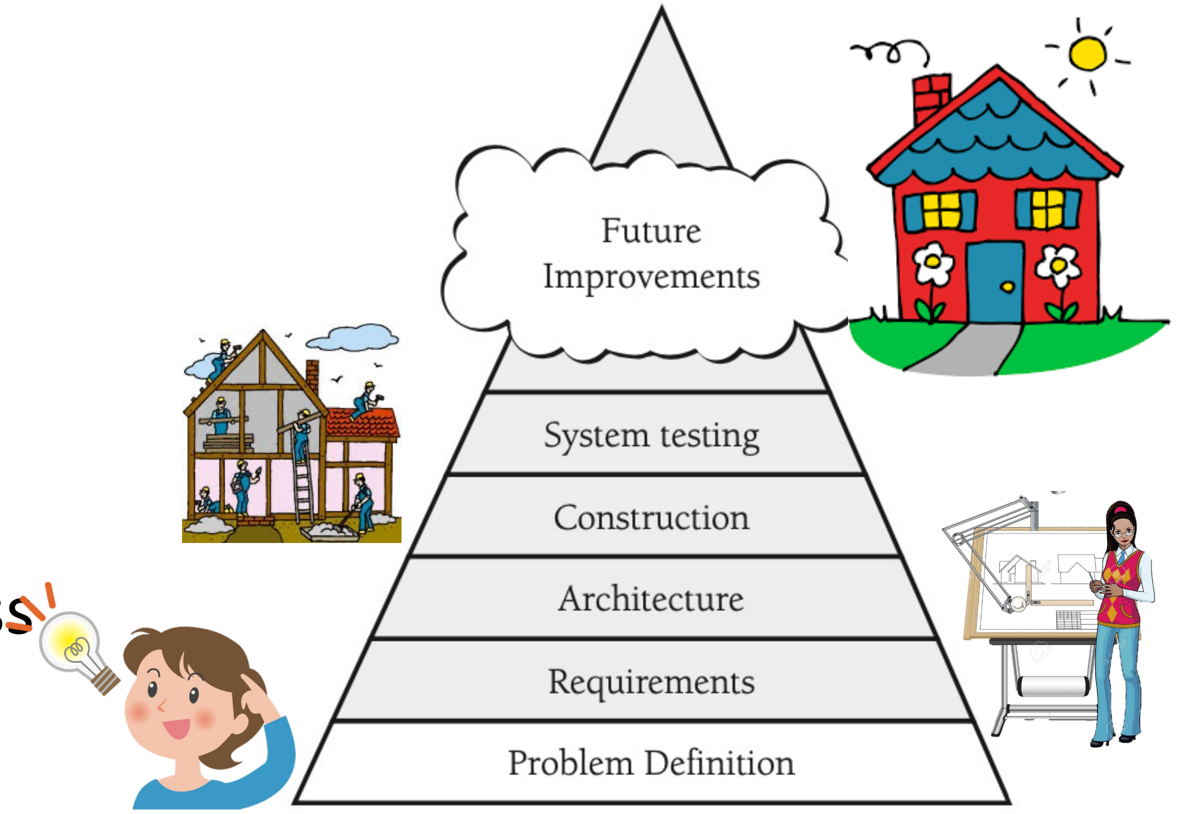


Figure 3.4 of McConnel, CC2, 2004.

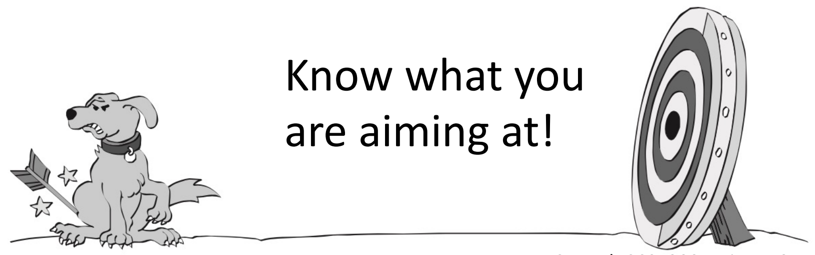
Problem Definition

- Definition: A *problem definition* describes the problem without making any reference to a solution
  - Non-technical statement (user's language)
  - Stated from the user's point of view
  - Short (no more than a couple pages)
  - Does not assume that a piece of software is needed to solve the problem
- Also known as "problem statement," "product vision," "vision statement," "mission statement," etc.
- Examples:
  - We need to attract more customers
    - Possible solution: A website
  - We have too many orders to easily keep track of them
    - Possible solution: A database
  - We are sooooo bored
    - Possible solution: A video game

Problem Definitions Do Not Specify Solutions

- These are poor examples of problem definitions:
  - We need a website to advertise our company
  - We would like to use a database to track our customers
  - We need software to run our factory
- Exception: When the problem to be solved is computer related
  - Example: I need an easy way to write my program in Java
- Why should a problem definition not include a solution?
  - Avoid constraining choice of solutions
- If there is no problem definition, there is a risk that the wrong problem will be solved

Know what you are aiming at!



McConnel, CC2, 2004, Figure 3-5

Requirements

- **Definition**: Requirements are a detailed specification of what the software is supposed to do
  - Functional requirements: What the software is supposed to do
  - Nonfunctional requirements: How the software system is
    - Implemented
    - Constrained
    - Etc
- Why do we need requirements?
  - Ensures that the client/user rather than programmer has a say in system functionality
  - Avoids disagreements about what the system is and is not supposed to do
    - Between developers
    - Between developers and clients
  - Minimizes number of changes that may occur after development commences

How Do I Create Software Requirements?

- This is beyond the scope of this course
- This is something discussed in CSCI 3130: Software Engineering
- Note: Entire books have been written on how to properly elicit software requirements

- We can get a sense of when the requirements are complete. ☺
- The following are some checklists: PLEASE DON'T MEMORIZE THEM!

Functional Software Requirements Checklist

(McConnell, CC2, 2004)

- Are all the tasks the user wants to perform specified?
- Is the data used in each task and the data resulting from each task specified?
- Are all the inputs to the system specified?
- Are all the outputs from the system specified?

Quality Software Requirements Checklist

(McConnell, CC2, 2004)

- Are the requirements written in the user’s language? Do the users think so?
- Does each requirement avoid conflicts with other requirements?
- Are acceptable trade-offs between competing attributes specified?
- Do the requirements avoid specifying the design?
- Are the requirements at the required level of detail?
- Are the requirements clear enough to be turned over to an independent group for construction and still be understood? Do the developers think so?
- Is each item relevant to the problem and its solution? Can each item be traced to its origin in the problem environment?
- Is each requirement testable? Will it be possible for independent testing to determine whether each requirement has been satisfied?
- Are all possible changes to the requirements specified, including the likelihood of each change?

Nonfunctional Software Requirements Checklist

(McConnell, CC2, 2004)

- Is the expected response time, from the user's point of view, specified for all necessary operations?
- Are other timing considerations specified, such as processing time, data-transfer rate, and system throughput?
- Is the level of security specified?
- Is the reliability specified, including the consequences of software failure, the vital information that needs to be protected from failure, and the strategy for error detection and recovery?
- Are minimum machine memory and free disk space specified?
- Is the maintainability of the system specified, including its ability to adapt to changes in specific functionality, changes in the operating environment, and changes in its interfaces with other software?
- Is the definition of success included? Of failure?

Complete Software Requirements Checklist
(McConnell, CC2, 2004)

- Where information isn’t available before development begins, are the areas of incompleteness specified?
- Are the requirements complete in the sense that if the product satisfies every requirement, it will be acceptable?
- Are you comfortable with all the requirements?
    - Have you eliminated requirements that were impossible to implement?
    - Were requirements included just to appease your customer or your boss?

The Myth of Stable Software Requirements

- Unless the project is small or very targeted, the requirements will change.
- Why?
  - Clients are almost never sure of what they want or need
  - Clients change their minds

    Clients learn what they need/want during the development process
  - Technologies evolve and change
- Requirements change can be as high as 25% on some projects.

  Imagine trying to design and implement when ¼ of the requirements change!

Next Step: Choosing a Software Architecture Constraints and Considerations

- Software architecture refers to the system-wide constraints that have a direct impact on the design and implementation.
- Software architecture defines the framework for the design
- Example: A web-based architecture means that the application
  - Is network based
  - Uses a RESTful protocol
  - Must use specific protocol standards
  - May be restricted to running in a web browser
  - Etc
- A software architecture specification should answer questions about how the software will be designed
  - The specification is typically divided into components.

Examples of Software Architecture Components

- Program organization:
  What are the major components / building blocks?
- Major classes:
  What are the main classes, their purpose, and collaborations?
- Data design:
  How will the data be organized?
- Business rules
  What business rules must be enforced?
- User interface design
  What user interfaces are required?
- Etc..
- **Note**: The answers to these questions guide the choice of the appropriate software architecture

Key Points

- Software pre-requisites is a process to prepare for software construction
- Understanding what software we are building allow us to choose the appropriate development approach (sequential or iterative)
- The problem definition is a description of the problem to be solved, without any reference to the solution
- Software requirements are elicited from the client and specify what the software is supposed to do
- Architecture pre-requisites specify the constraints the software’s design and implementation must meet.

Image References

Retrieved February 29, 2020

- http://pengetouristboard.co.uk/vote-best-takeaway-se20/
- https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcRoUb4LYIeDlkV1bj5XYan2OlGvP8uEn-Tl9X0UluW9DsjHn4yL
- https://www.pinterest.ca/pin/543950461239636468/?lp=true
- https://www.clipart.email/clipart/land-background-clipart-77106.html
- https://cdn.clipart.email/7792ce3439f4bbcef5824087e8440162_permits-applications_600-401.jpeg
- https://webstockreview.net/images/contractor-clipart.jpg
- https://www.kindpng.com/picc/m/496-4962852_light-bulb-idea-clipart-hd-png-download.png
- https://cliparting.com/free-money-clipart-1926/
- https://www.istockphoto.com/ca/illustrations/blueprint?sort=mostpopular&mediatype=illustration&phrase=blueprint

Software Architecture Components

- Program organization
- Major classes
- Data design
- Business rules
- User interface design
- Resource management
- Security
- Performance
- Scalability
- Interoperability
- Internationalization/Localization
- Input/Output
- Error processing
- Fault tolerance
- Architectural feasibility
- Overengineering
- Buy-vs-Build decisions
- Reuse decisions
- Change strategy