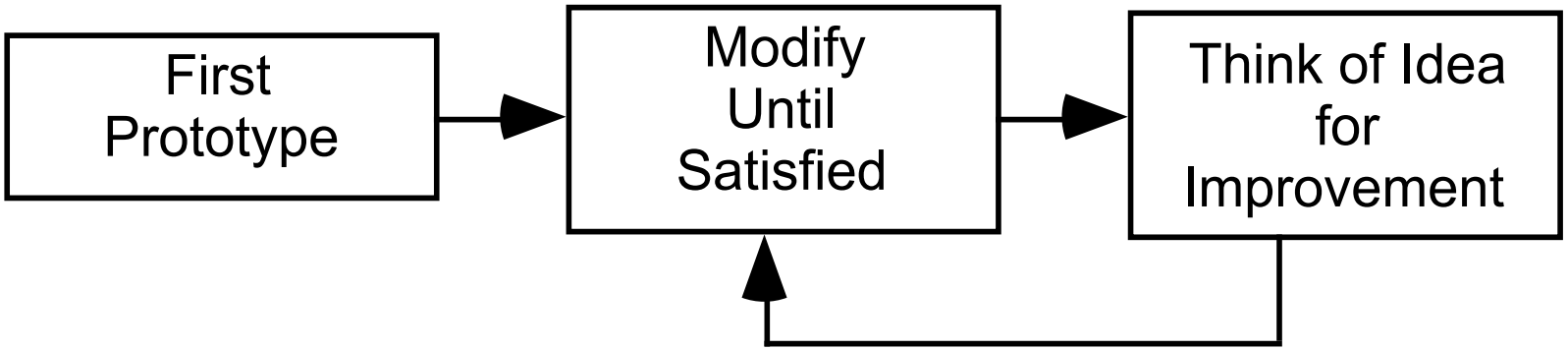
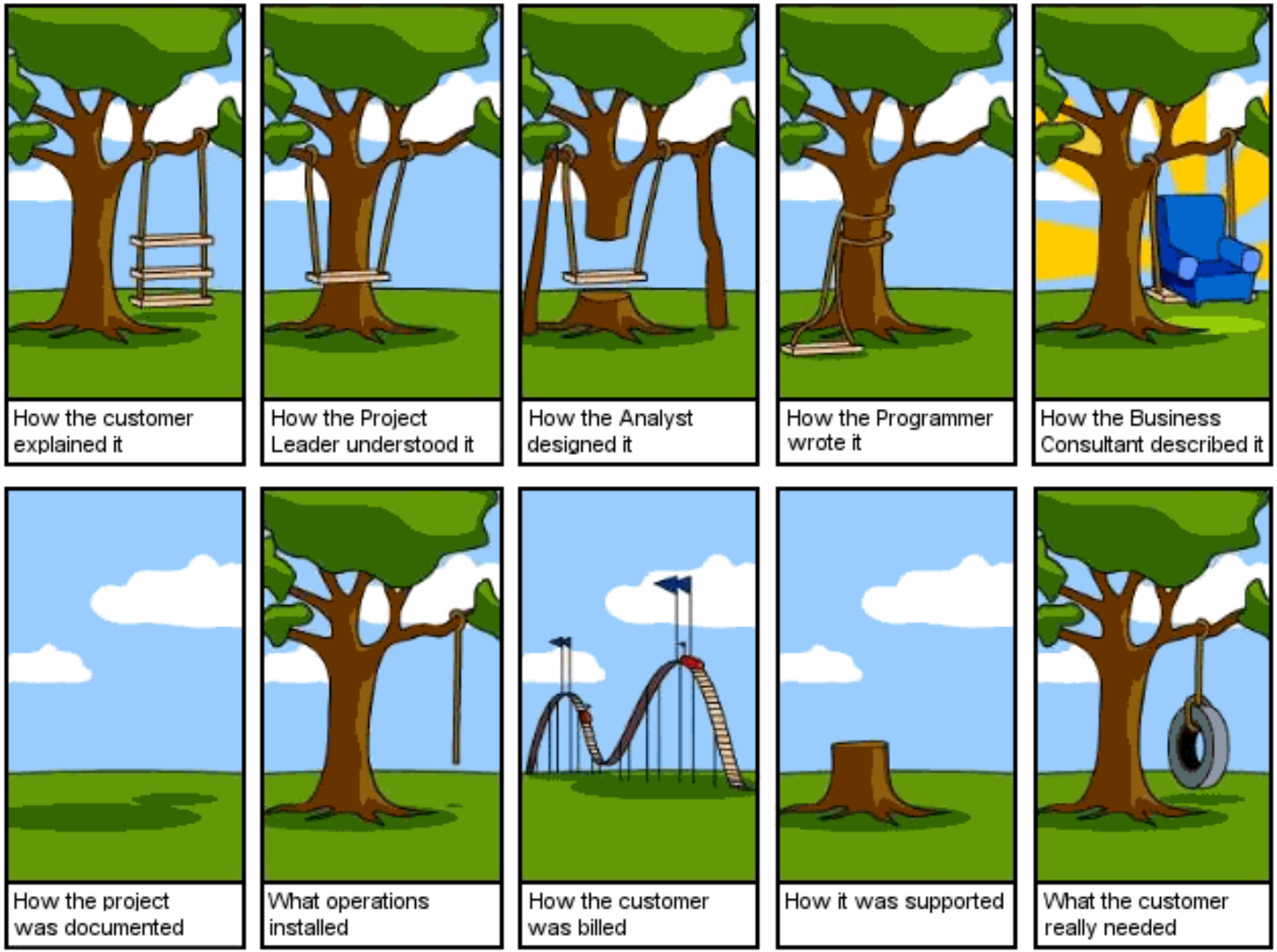
Software Development Life Cycle

DASS, Monsoon 2024
IIIT Hyderabad

The Opportunistic approach



- OK for small, informal projects
- Inappropriate for professional environments/complex software where on-time delivery and high quality are expected


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

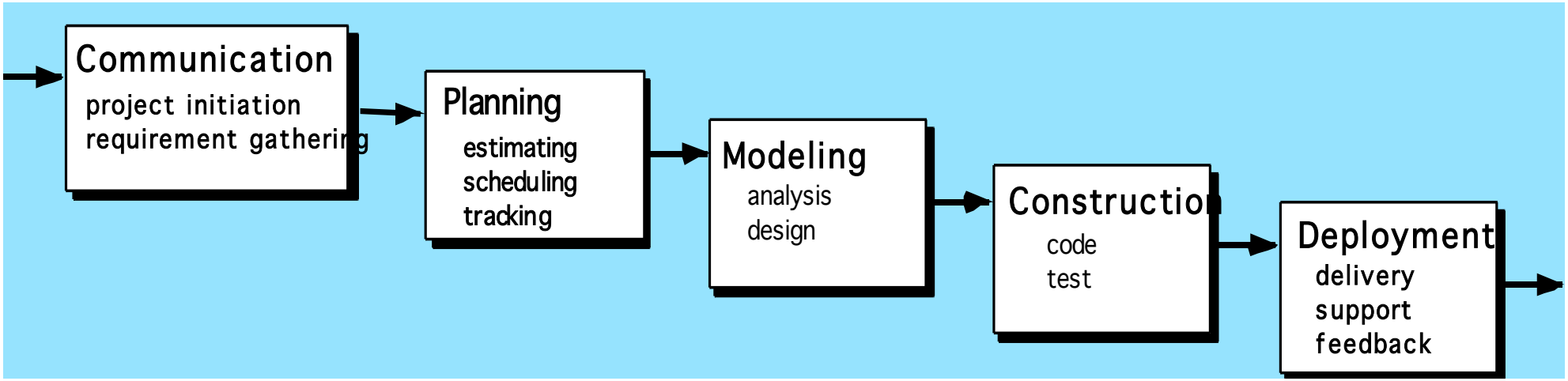
Why Lifecycle Model?

- A software project will never succeed if activities are not coordinated:
  - one engineer starts writing code,
  - another concentrates on writing the test document first,
  - yet another engineer first defines the file structure
  - another defines the I/O for his portion first

A software life cycle model (or process model):

- a descriptive and diagrammatic model of software life cycle:
- identifies all the activities required for product development
- establishes a precedence ordering among the different activities
- divides life cycle into phases.

Software Development Life Cycle (SDLC)



SDLC (simplified):

- Feasibility study/Planning (involves business case)
- Requirements Engineering
- Architecture/Design
- Implementation
- Testing
- Maintenance

Image Courtesy: Sofware Engineering: A Practioner's approach by Roger Pressman

Requirements Engineering

- Aim of this phase:
  - understand the exact requirements of the customer,
  - document them properly.

- Consists of following activities:
  - Elicitation/Gathering
  - Analysis
  - Specification
  - Verification
  - Management

Design

- Design phase transforms requirements specification:
  - into a form suitable for implementation in some programming language.

High-level design (Architecture):

- decompose the system into *modules*,
- represent invocation relationships among the modules.

Detailed design:

- different modules designed in greater detail:
  - data structures and algorithms for each module are designed

IMPLEMENTATION

- During the implementation phase:
  - each module of the design is coded,

The end product of implementation phase:

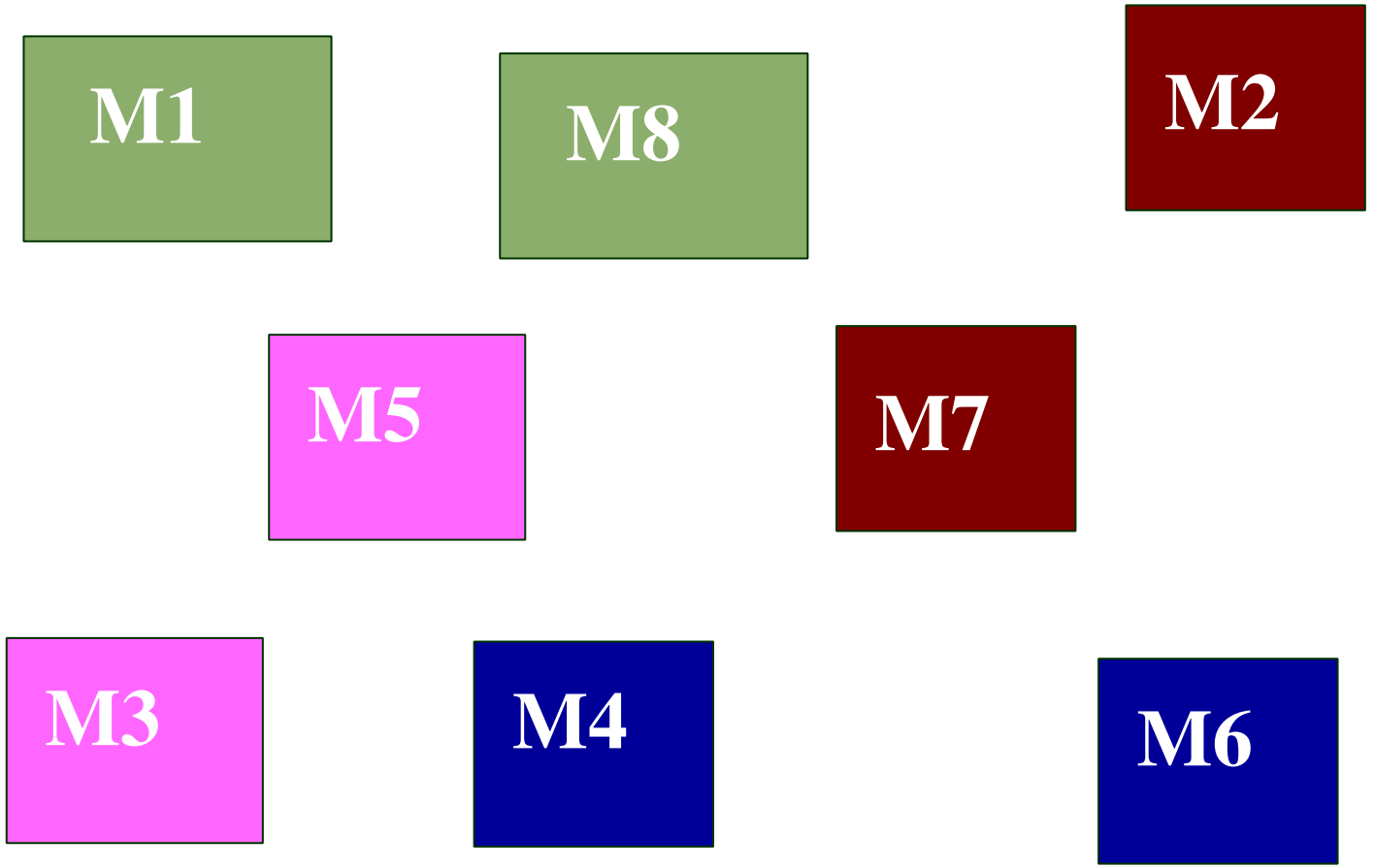
- a set of program modules that have been tested individually.

INTEGRATION AND SYSTEM TESTING

- Different modules are integrated in a planned manner:
  - modules are almost never integrated in one shot.
  - Normally integration is carried out through a number of steps.

- During each integration step,
  - the partially integrated system is tested.

INTEGRATION AND SYSTEM TESTING



SYSTEM TESTING

- After all the modules have been successfully integrated and tested:
  - system testing is carried out.
- Goal of system testing:
  - ensure that the developed system functions according to its requirements as specified in the SRS document.

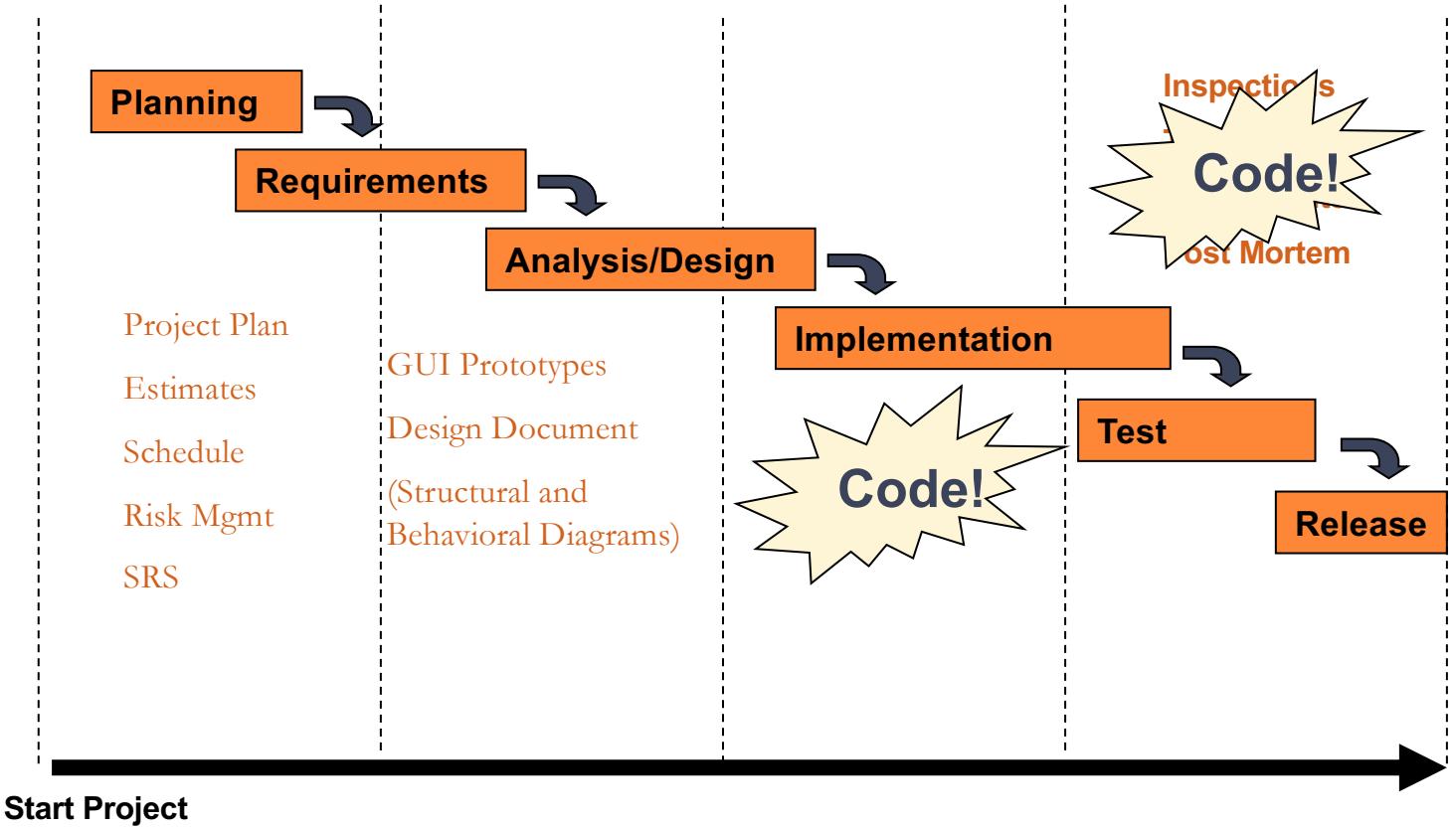
V Model – Popular lifecycle model

MAINTENANCE

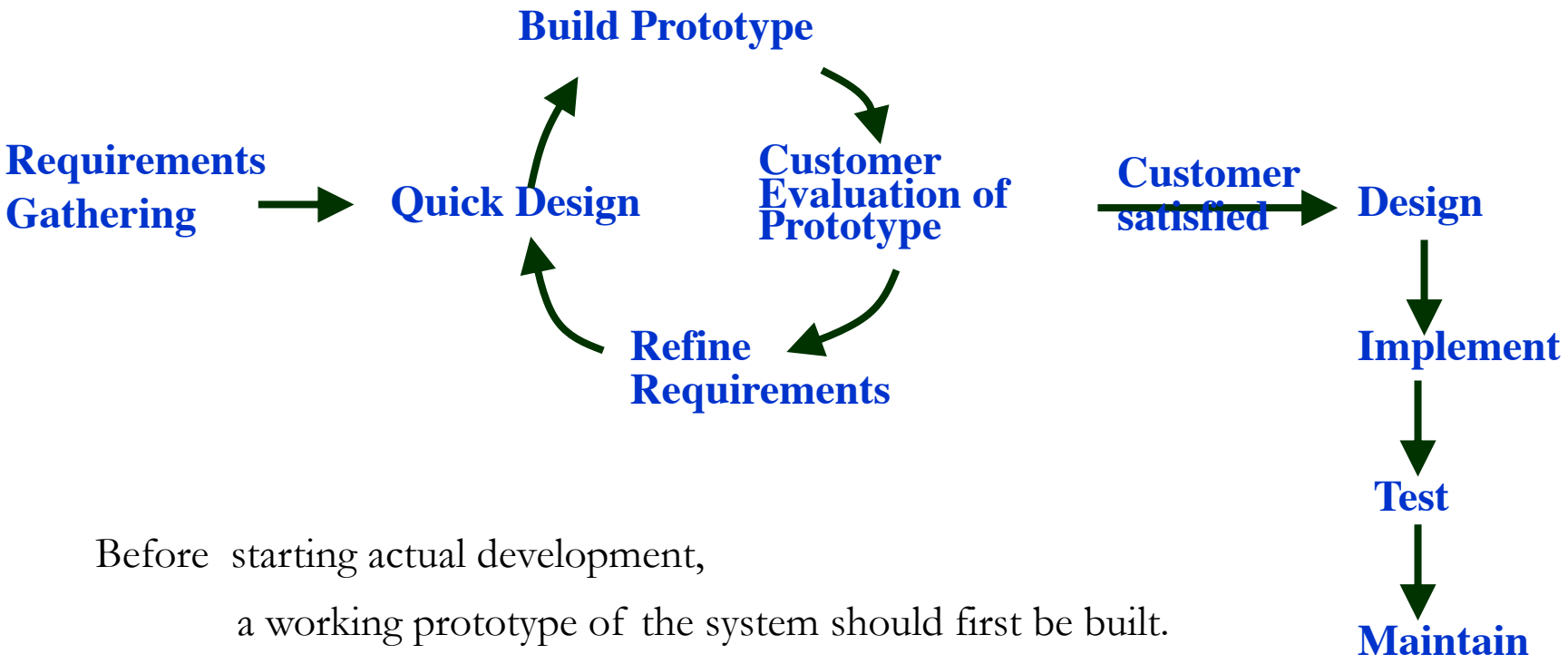
- Preventive maintenance
  - Making appropriate changes to prevent the occurrence of errors
- Corrective maintenance
  - Correct errors which were not discovered during the product development phases
- Perfective maintenance
  - Improve implementation of the system
  - enhance functionalities of the system
- Adaptive maintenance
  - Port software to a new environment

Process Models

Traditional SDLC. (e.g. waterfall process)



PROTOTYPING MODEL



Before starting actual development,

a working prototype of the system should first be built.

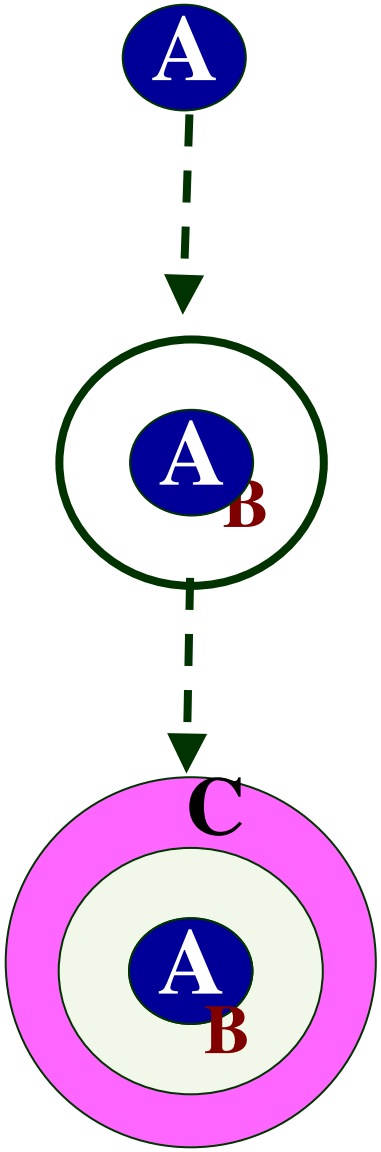
A prototype is a toy implementation of a system:

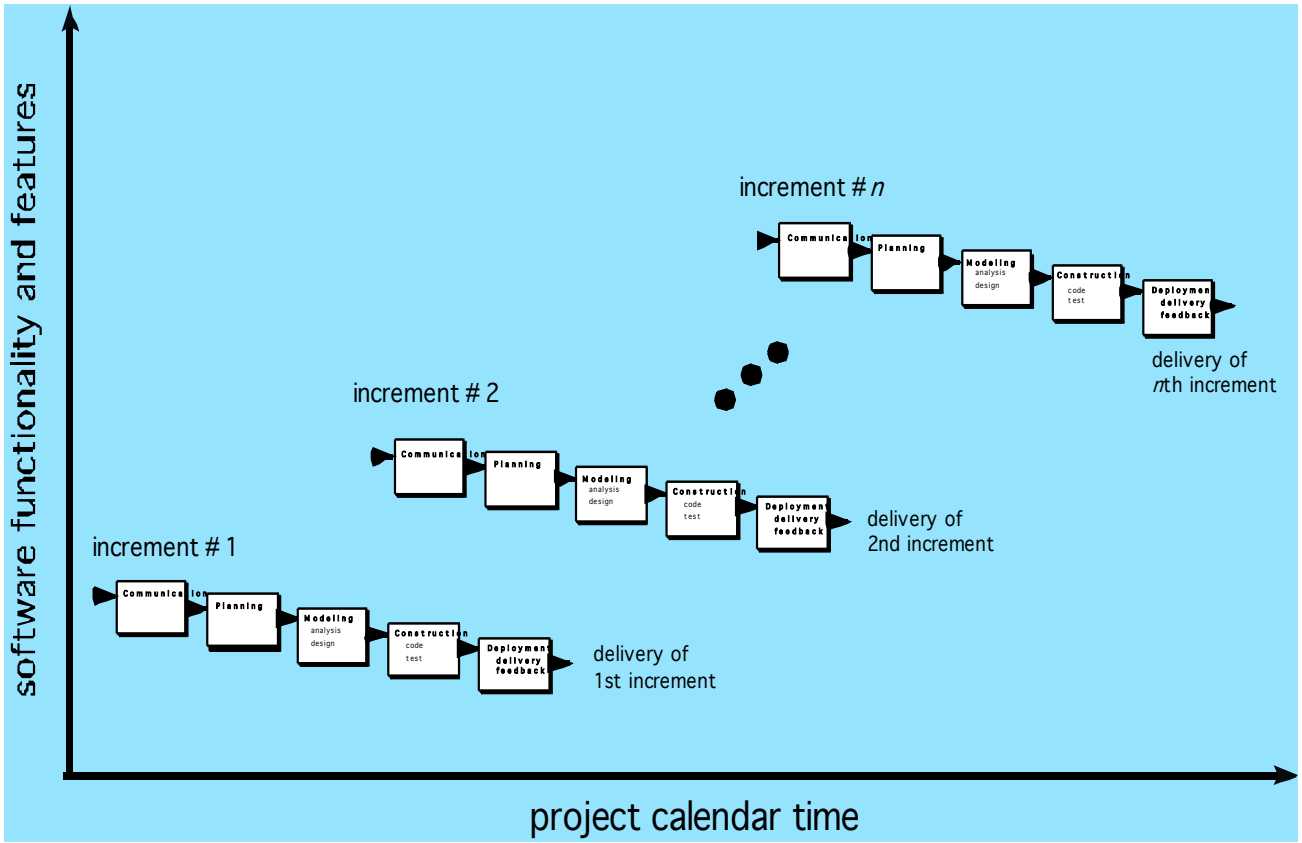
limited functional capabilities,

low reliability,

inefficient performance.

INCREMENTAL MODEL





Challenges with traditional (plan-driven) approaches

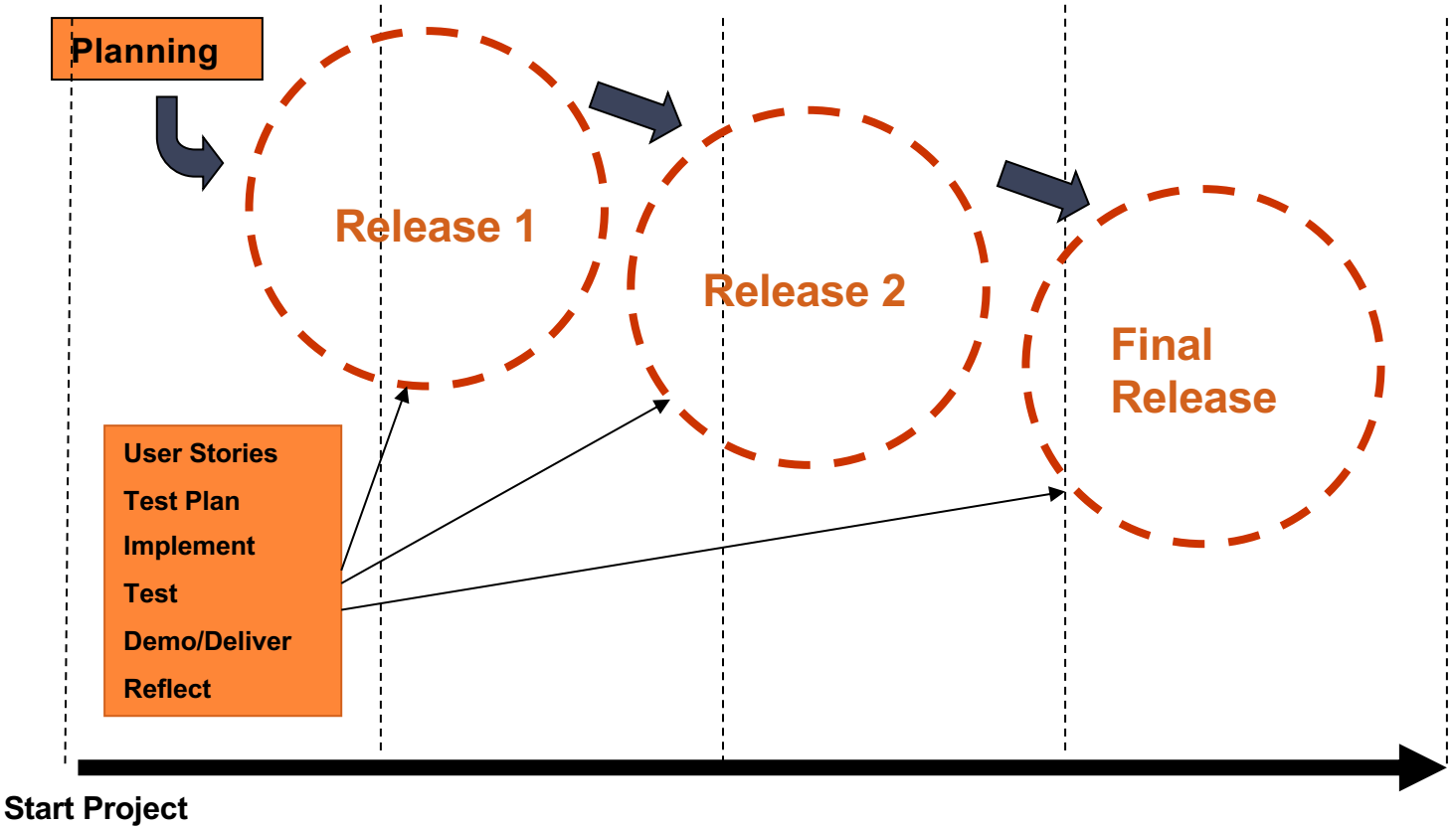
- Lightweight applications/heavyweight process
- Document intensive (perceived)
- Less flexible design
- Big bang approach to coding/integration
- Testing short-shifted
- One-shot delivery opportunity
- Limited opportunity for process improvement

What Is Agile Software Development?

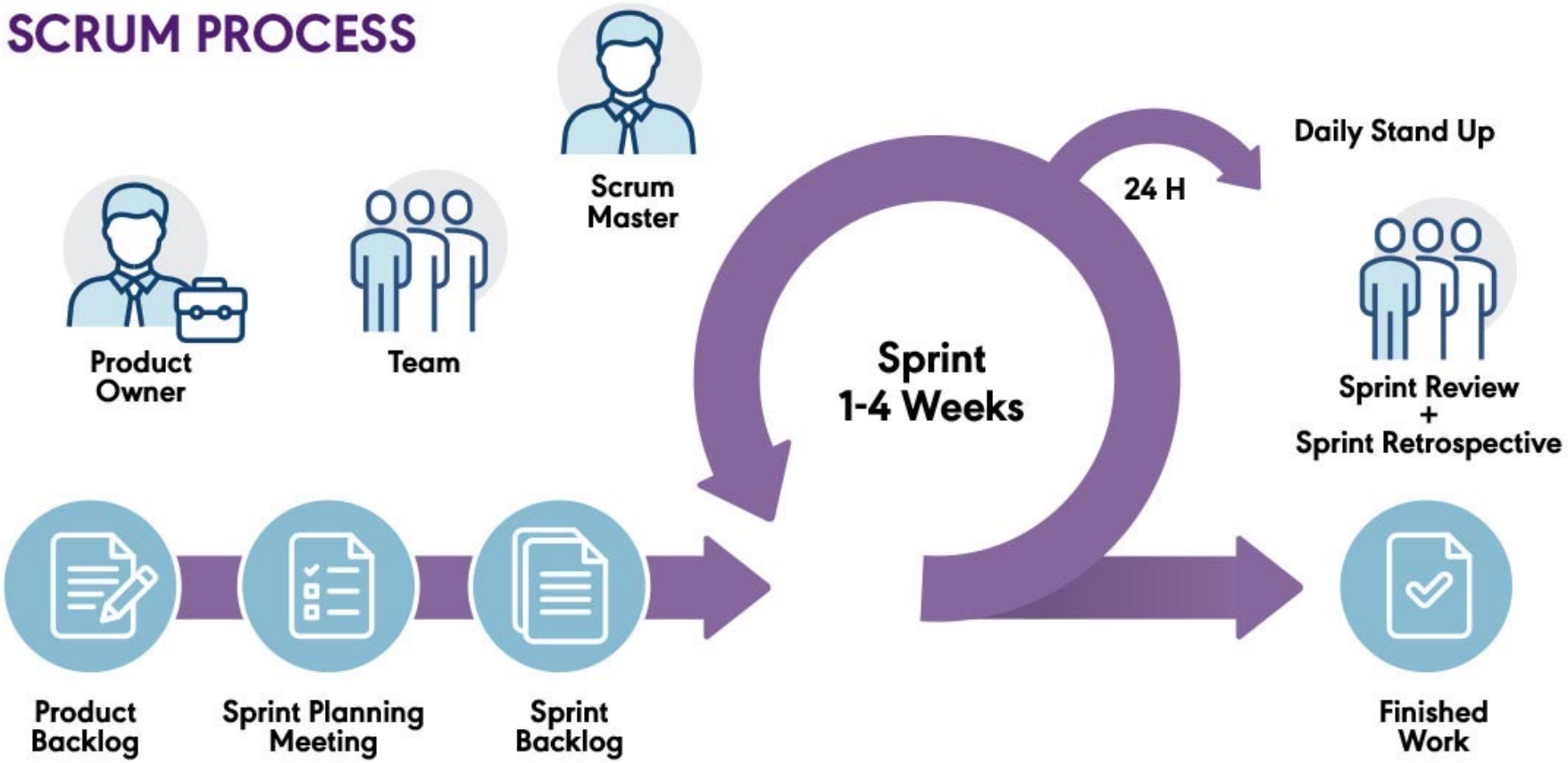
- In the late 1990's several methodologies began to get increasing public attention. All emphasized:
  - Close collaboration between developers and business experts
  - Face-to-face communication (as more efficient than written documentation)
  - Frequent delivery of new deployable business value
  - Tight, self-organizing teams
  - Ways to craft the code and the team such that the inevitable requirements churn was not a crisis.

- 2001 : Workshop in Snowbird, Utah, Practitioners of these methodologies met to figure out just what it was they had in common. They picked the word "agile" for an umbrella term and crafted the
  - Manifesto for Agile Software Development,

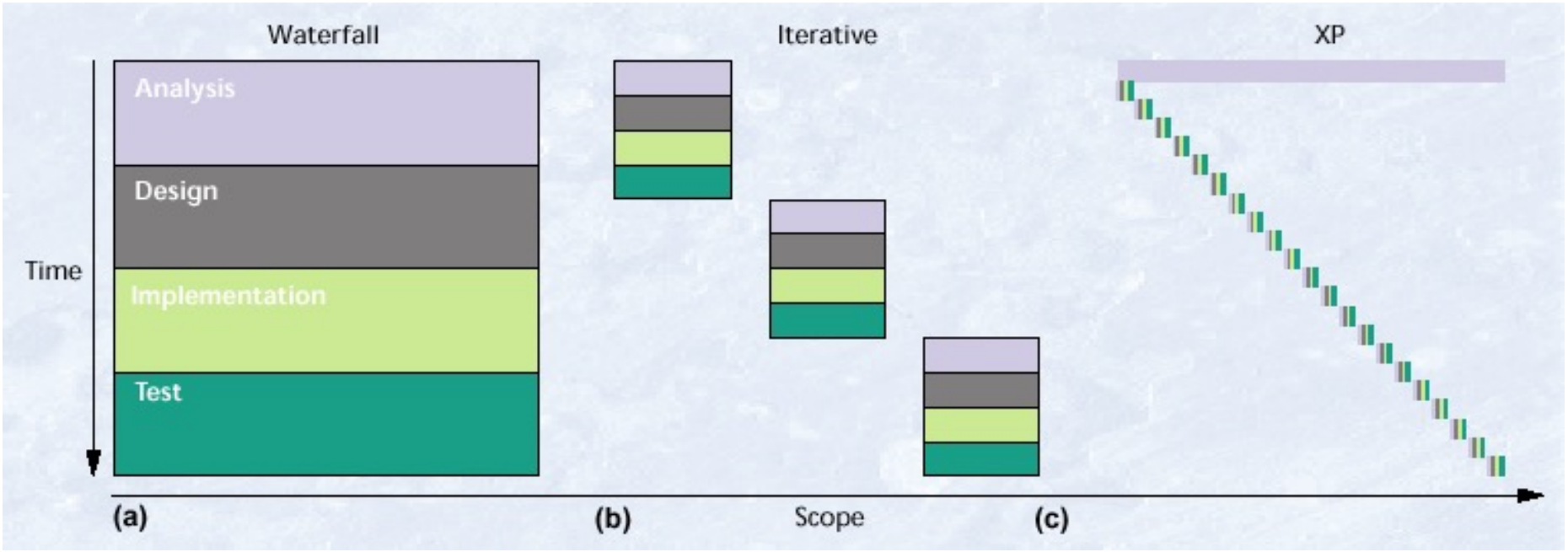
Applying Agility



SCRUM PROCESS

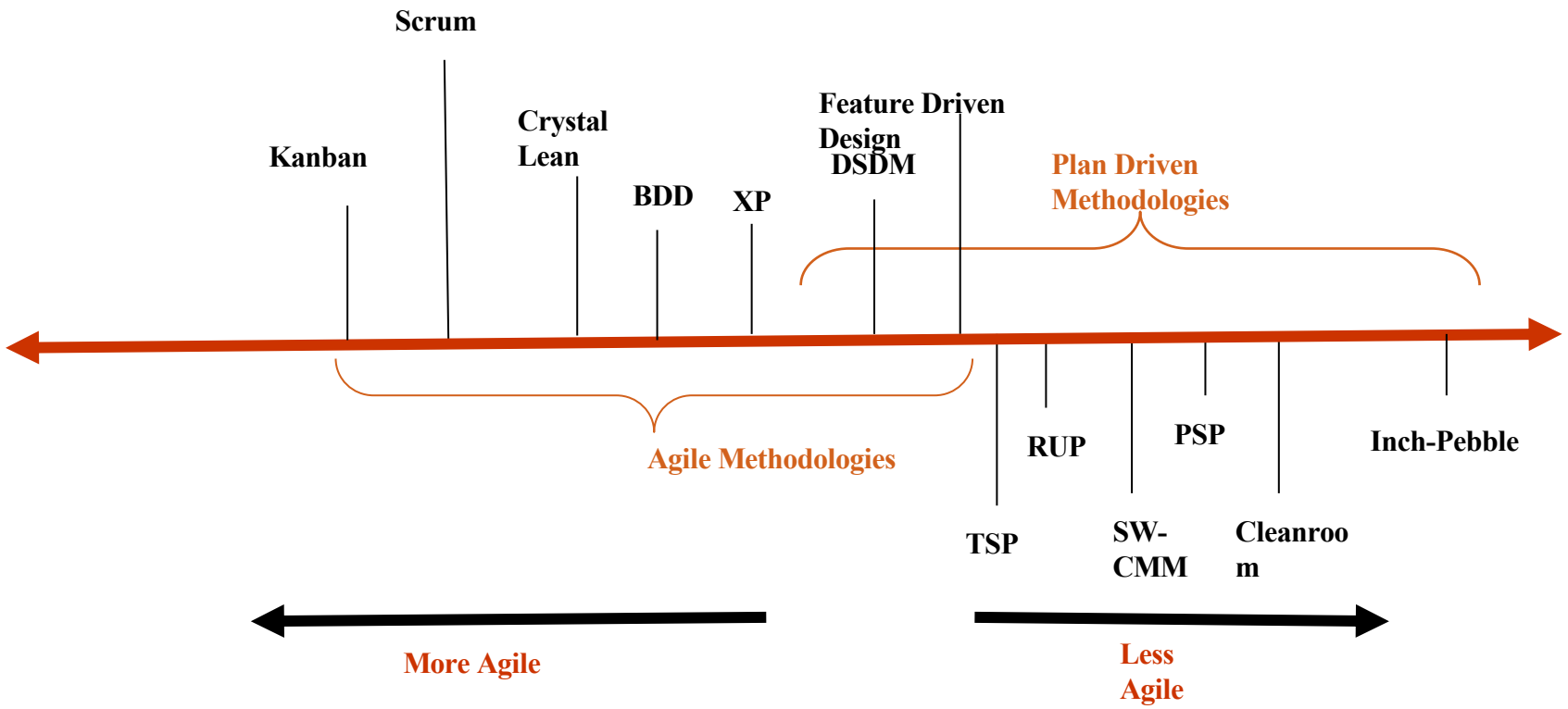


Waterfall to XP Evolution



Source: "Embracing change with extreme programming" by Kent Beck, *IEEE Computer*, October 1999.

The Process Methodology Spectrum



It's not that black and white. The process spectrum spans a range of grey !

@ (Barry Boehm & Turner)

Agile Characteristics

- Incremental development – several releases
- Planning based on user stories
- Each iteration touches all life-cycle activities
- Testing – unit testing for deliverables; acceptance tests for each release
- Flexible Design – evolution vs. big upfront effort
- Reflection after each release cycle
- Several technical and customer focused presentation opportunities

Key Agile Components

- User Stories
  - Requirements elicitation
  - Planning – scope & composition
- Evolutionary Design
  - Opportunity to make mistakes
- **Test driven development**
  - Dispels notion of testing as an end of cycle activity
- **Continuous Integration**
  - Code (small booms vs big bang)
- **Refactoring**
  - Small changes to code base to maintain design entropy
- Team Skills
  - Collaborative Development (Pair programming)
  - Reflections (process improvement)
- Communication/shared ownership
  - Interacting with customer / team members

Test Driven Development (TDD)

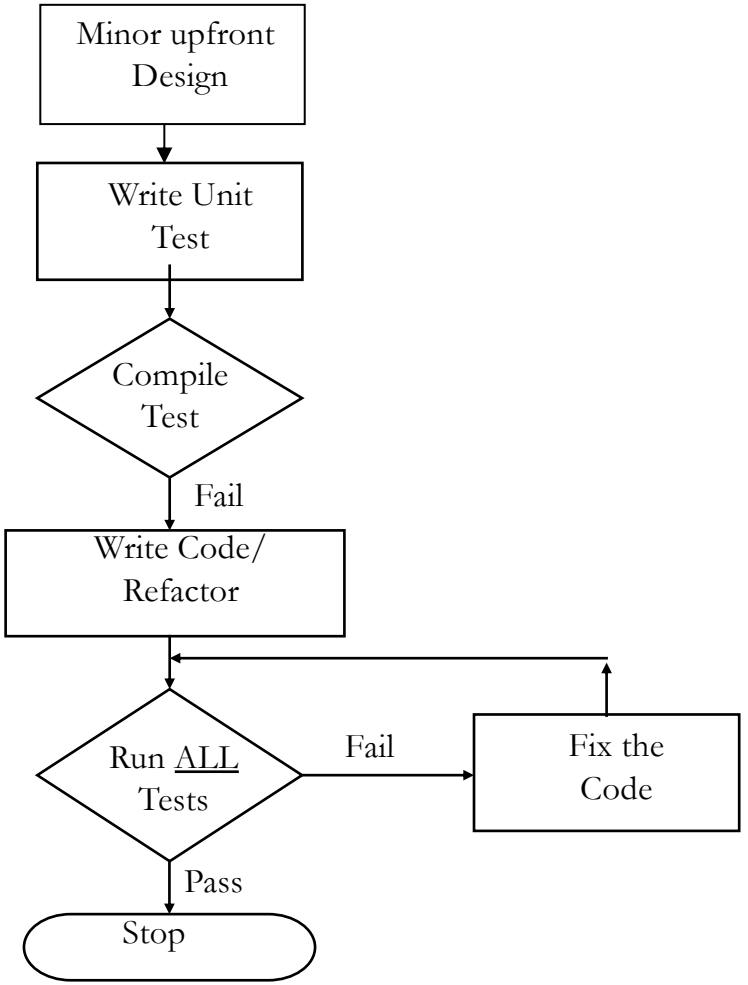
- "State of the art" → test-last
- "State of the practice" → test-whenever needed
- TDD → test-first

- Design evolves through coding/feedback

- Write unit tests for every piece of code that could possibly break

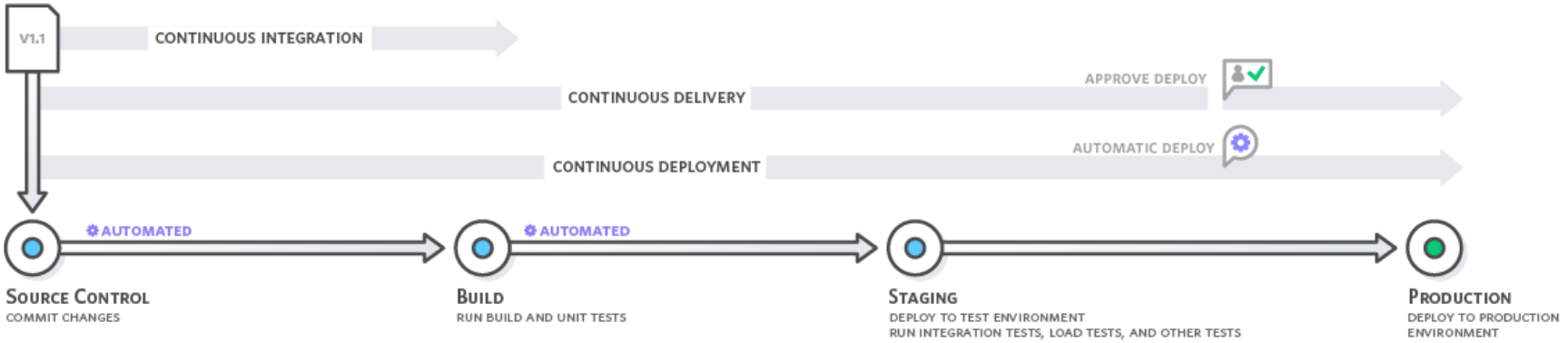
- Preferred testing tool are xUnits (open source)

TDD Explained



Continuous Integration

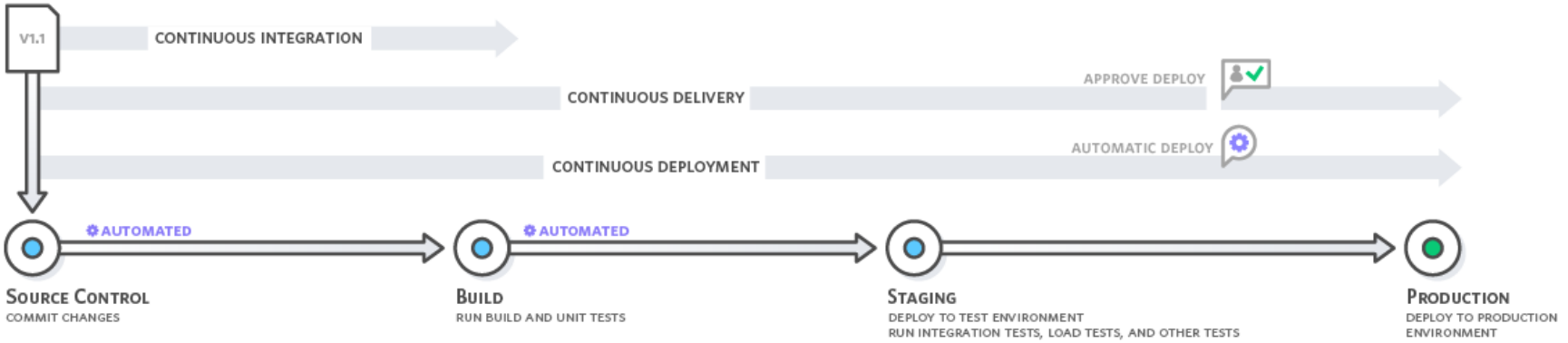
- Developers merge their code into the central repository regularly
- Automated builds and testing



Imagesource:aws.amazon.com

Continuous Delivery

- Deploys all code changes to a testing and/or production environment after the build stage
- Deployment is manual



Imagesource:aws.amazon.com

Development problems addressed – What about Release problems ?

- Database issues
- OS issues
- Too slow in real settings
- Infrastructure issues
- Source from many repositories
- Different versions (libraries, compilers, local utilities, etc)
- Missing dependencies
- …

Developers

- Designing
- Coding
- Testing, bug tracking, reviews
- Continuous Integration
- ...

Operations

Managing/Allocating hardware/OS updates/resources, database

Monitoring load spikes, performance, crashes hardware updates

Backups, Rollback releases

etc.

- ✓ Can there be better coordination between Developers and Operators?
- ✓ Reduce issues while moving changes from development to production
- ✓ Configurations as code
- ✓ Automation (Delivery and Monitoring)

DevOps

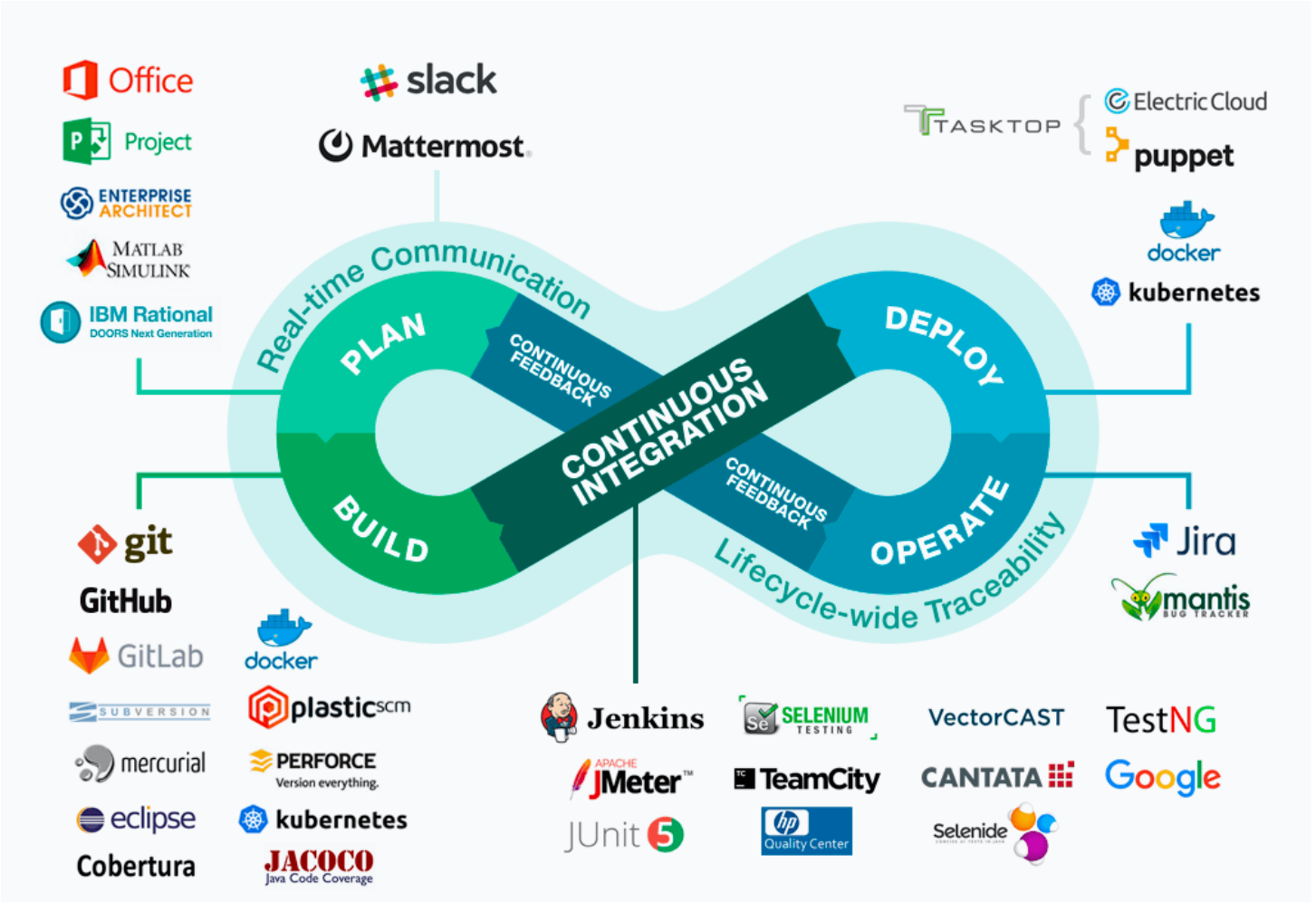
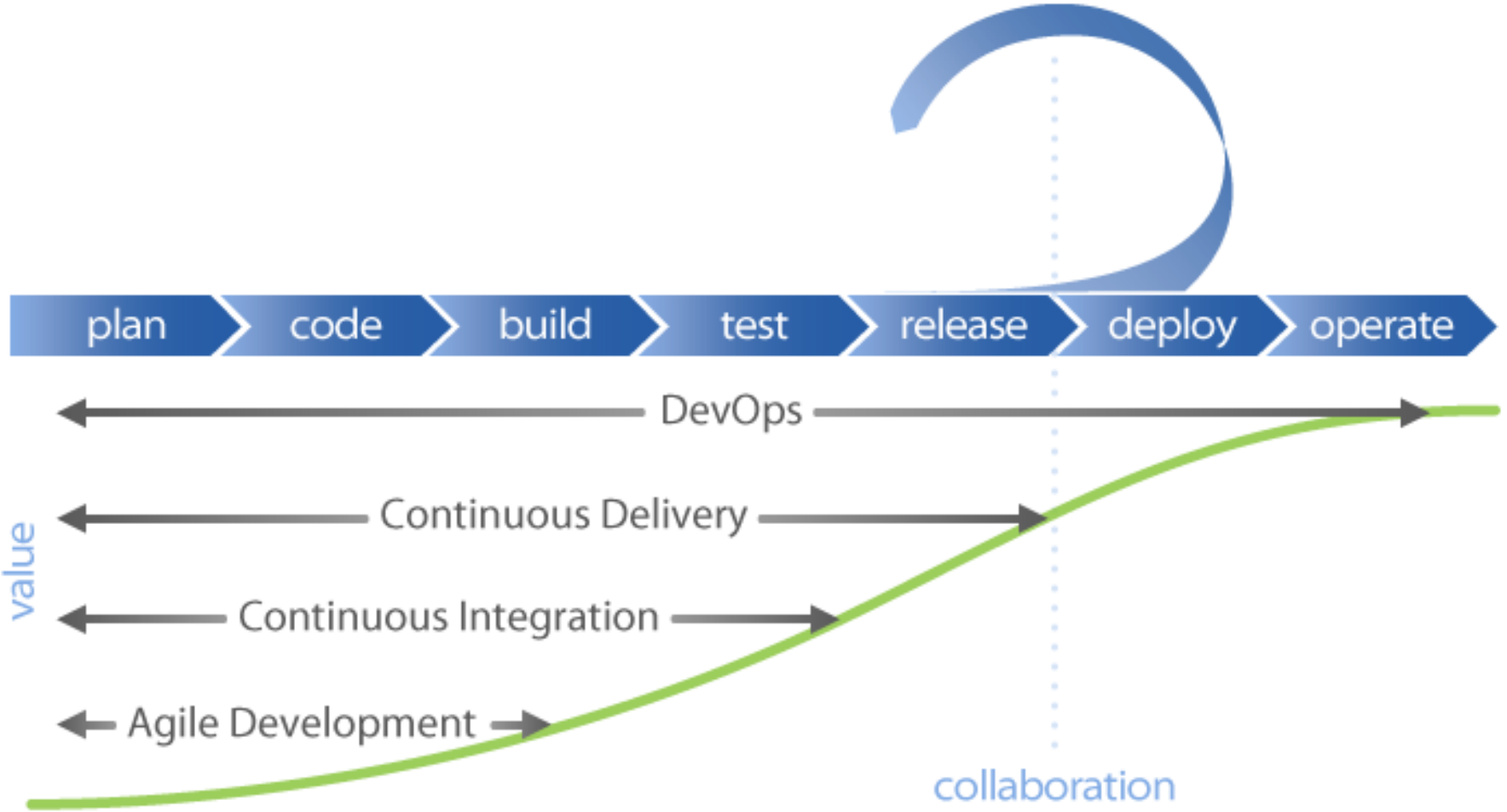


Image Source: https://intland.com/wp-content/uploads/2019/07/devops-infinity-1-1.png

DevOps – Common practices

- Continuous Integration
- Continuous Delivery
- Infrastructure as code, test and deploy in containers
- Monitoring and logging
- Microservice architecture
- Communicate and Collaborate

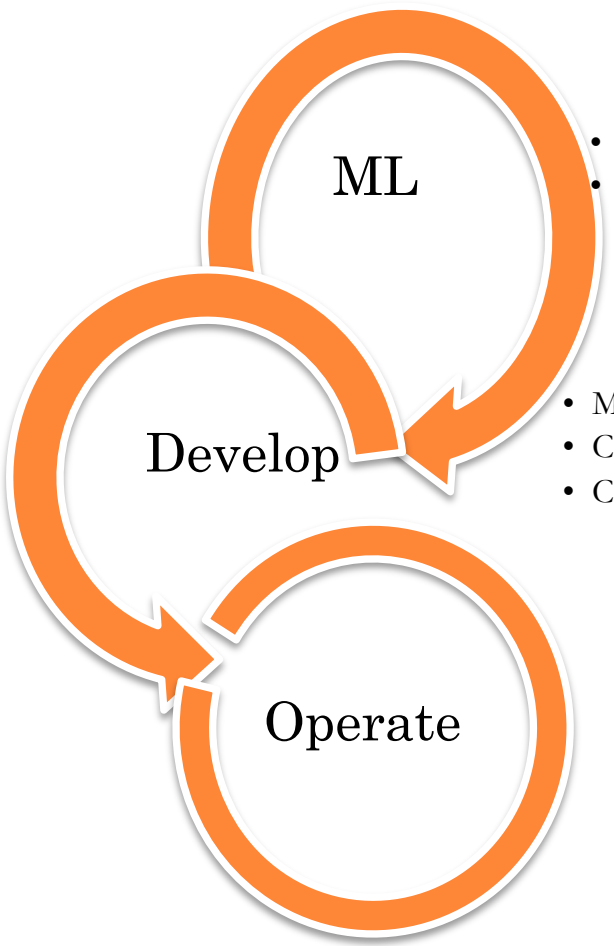
Agile, CI, CD, DevOps…



image@CollabNet

MLOps

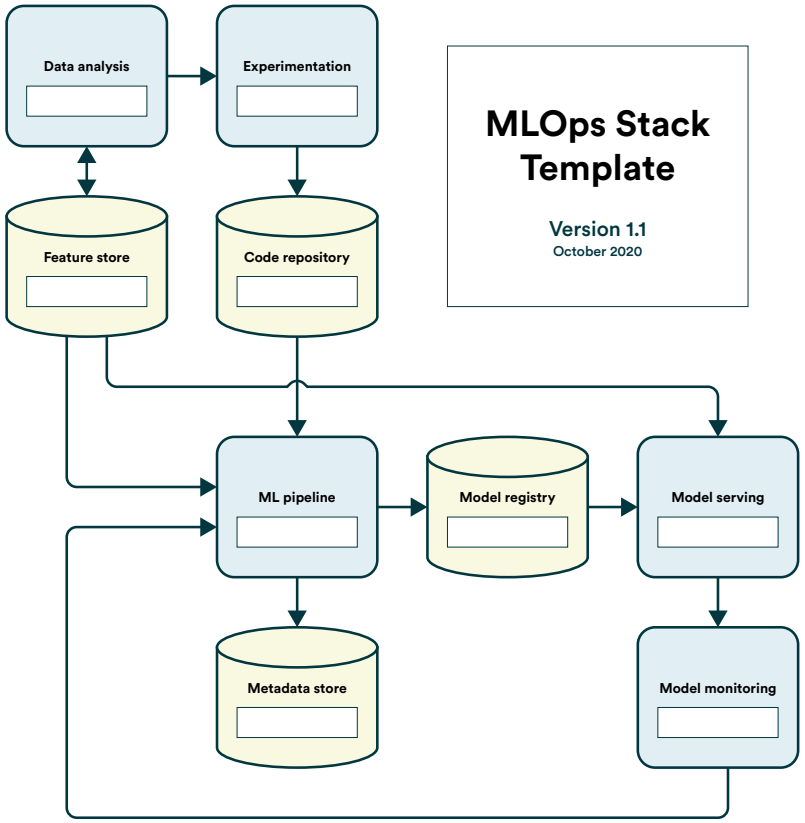
- ML + Dev + Ops
- Adopts best practices of DevOps.
- Allows for experimentation (Canary Releases)



- Data Acquisition
- Initial Modeling

- Modeling + Testing
- Continuous Integration
- Continuous Deployment

- Continuous Delivery
- Data Feedback Loop
- System + Model Monitoring



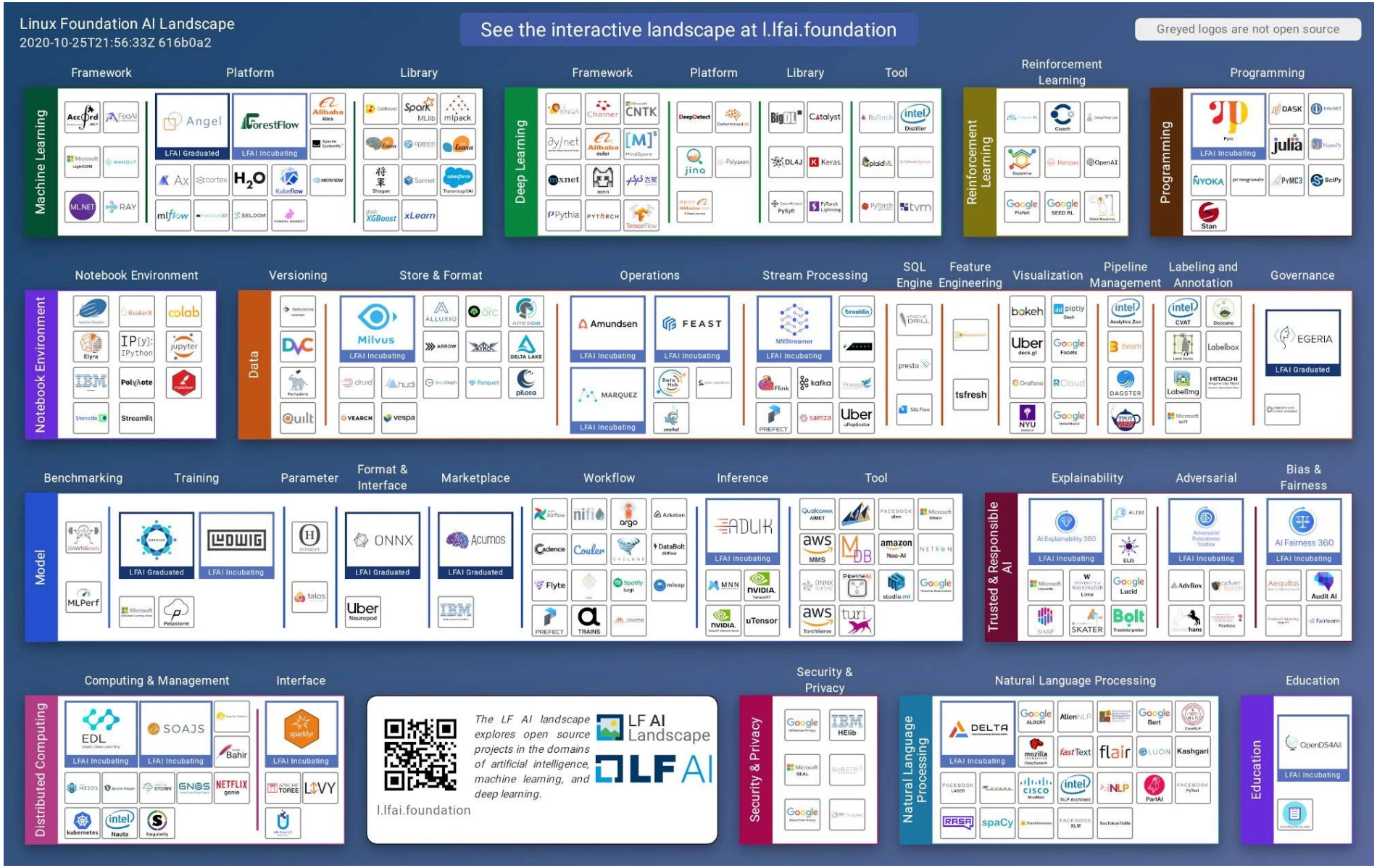
MLOps Stack Template

Version 1.1
October 2020

Component	Requirements	Tooling
Data analysis		
Experimentation		
Feature store		
Code repository		
ML pipeline		
Metadata store		
Model registry		
Model serving		
Model monitoring		

MLOps Setup Components	Tools
Data Analysis	Python, Pandas
Source Control	Git
Test & Build Services	PyTest & Make
Deployment Services	Git, DVC
Model & Dataset Registry	DVC [aws s3]
Feature Store	Project code library
ML Metadata Store	DVC
ML Pipeline Orchestrator	DVC & Make

Source: ml-ops.org


Linux Foundation AI Landscape
2020-10-25T21:56:33Z 616b0a2
See the interactive landscape at l.lfai.foundation
Greyed logos are not open source
Framework
Platform
Library
Machine Learning
Angel
LFAI Graduated
ForestFlow
LFAI Incubating
Framework
Platform
Library
Tool
Deep Learning
Reinforcement Learning
Programming
Notebook Environment
Versioning
Store & Format
Operations
Stream Processing
SQL Engine
Feature Engineering
Visualization
Pipeline Management
Labeling and Annotation
Governance
Data
Milvus
LFAI Incubating
Amundsen
FEAST
MARQUEZ
EGERIA
LFAI Graduated
Benchmarking
Training
Parameter
Format & Interface
Marketplace
Workflow
Inference
Tool
Model
ONNX
LFAI Graduated
Acumos
LFAI Graduated
ADLIK
LFAI Incubating
Explainability
Adversarial
Bias & Fairness
Trusted & Responsible AI
AI Explainability 360
LFAI Incubating
AI Fairness 360
LFAI Incubating
Computing & Management
Interface
Distributed Computing
EDL
LFAI Incubating
SOAJS
LFAI Incubating
sparklyr
LFAI Incubating
The LF AI landscape explores open source projects in the domains of artificial intelligence, machine learning, and deep learning.
LF AI Landscape
LF AI
l.lfai.foundation
Security & Privacy
Security & Privacy
Natural Language Processing
Natural Language Processing
DELTA
LFAI Incubating
Education
Education
OpenDS4All
LFAI Incubating

Estimation, Scheduling & Tracking (Week 3)

Software Estimation

Some content adapted from “Rapid Development” by Steve McConnell

Can you estimate these?

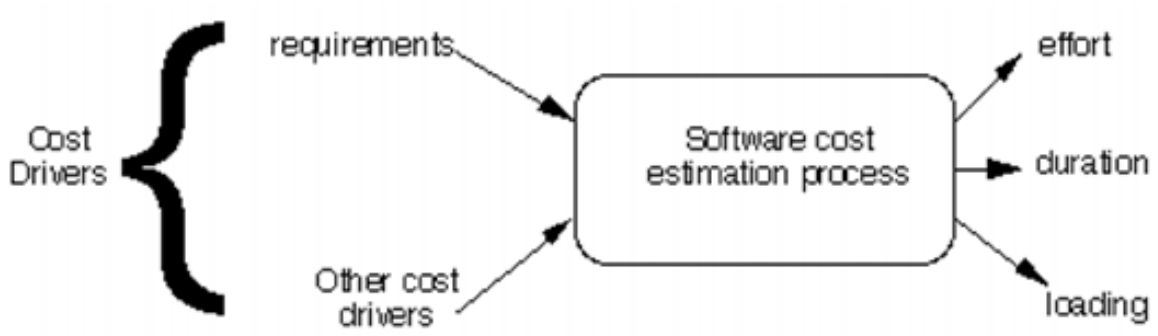
1. Surface temperature of the sun (in degrees C)
2. Latitude of Hyderabad (in degrees)
3. Surface area of Asia (in km^2)
4. Birth date of Alexander The Great (year)
5. Global revenue of "Titanic" (in $)
6. Length of the Pacific coastline (Ca, Or, Wa) (in km)
7. Weight of the largest whale (in tonnes)

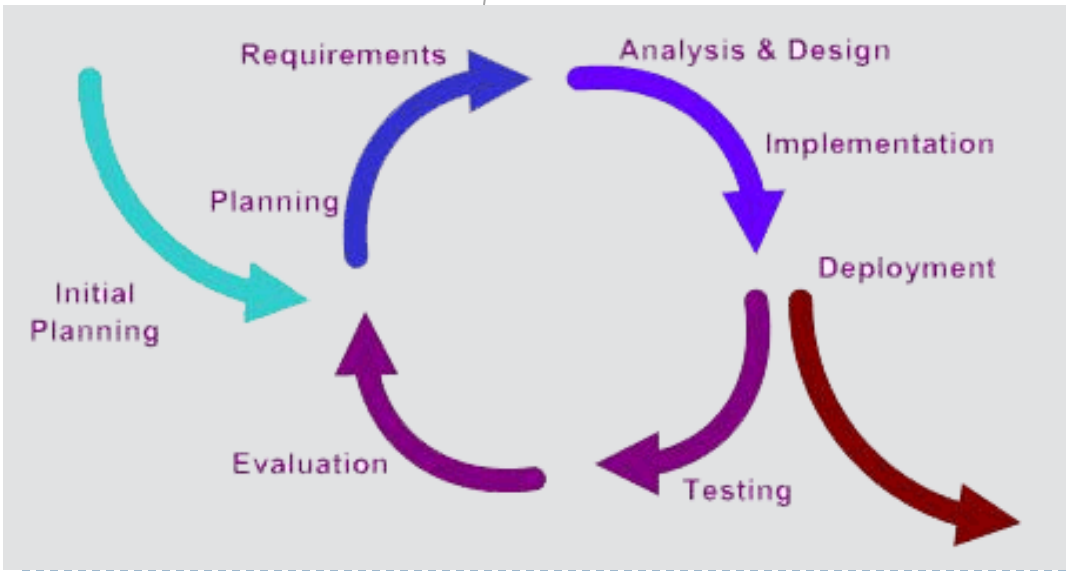
Adapted from "Software Estimation" by Steve McConnell

1

Always give a range
Never give them a number

Approach

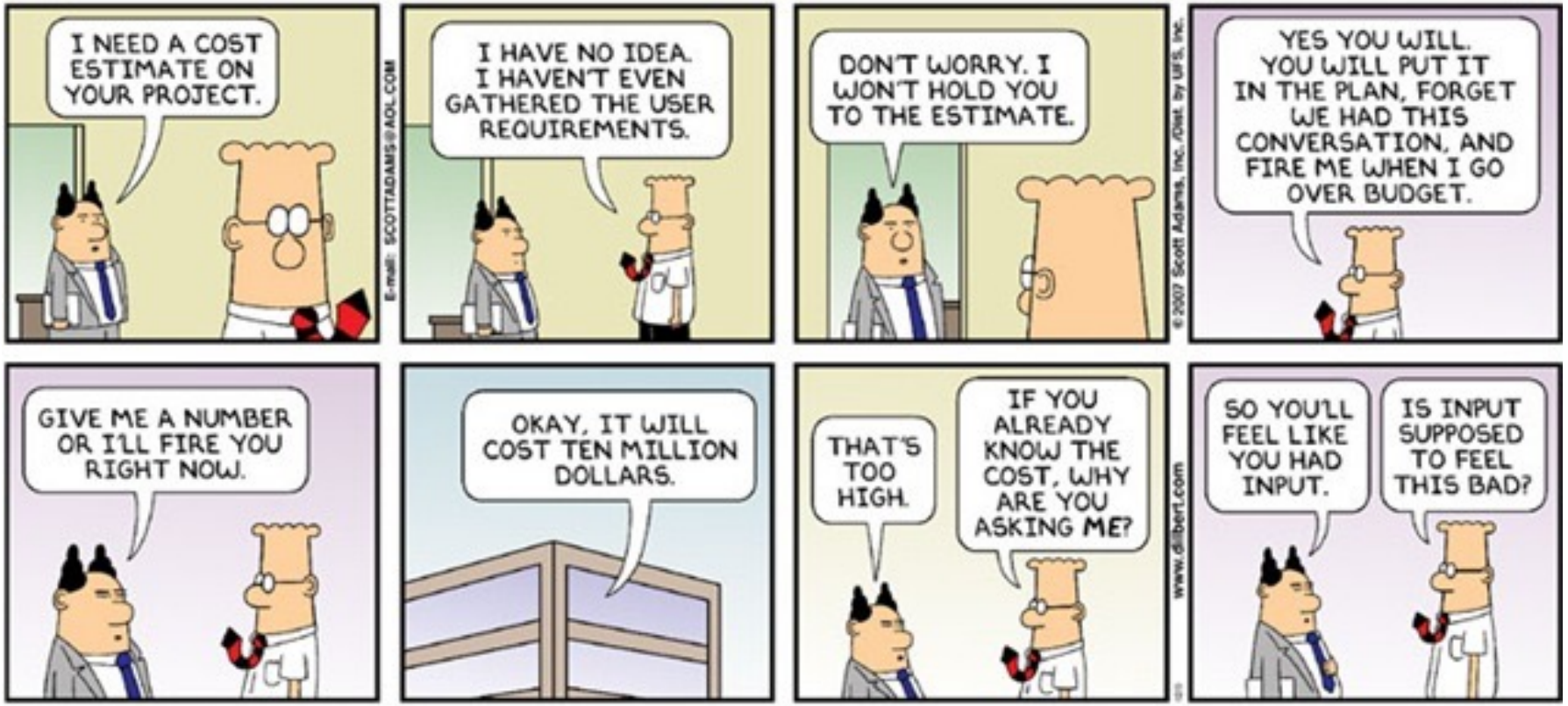

Cost
Drivers
requirements
Other cost
drivers
Software cost
estimation process
effort
duration
loading


Requirements
Analysis & Design
Implementation
Planning
Deployment
Initial
Planning
Evaluation
Testing

#2

Always ask what the estimate will be used for

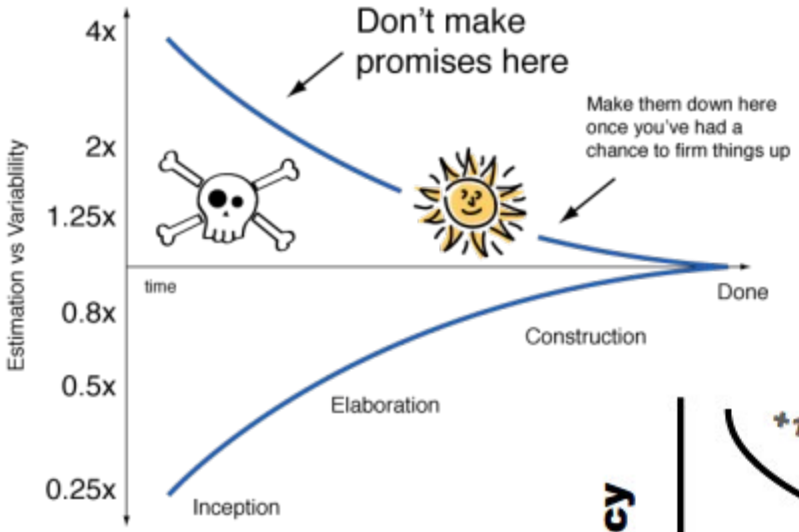
Requirements is key

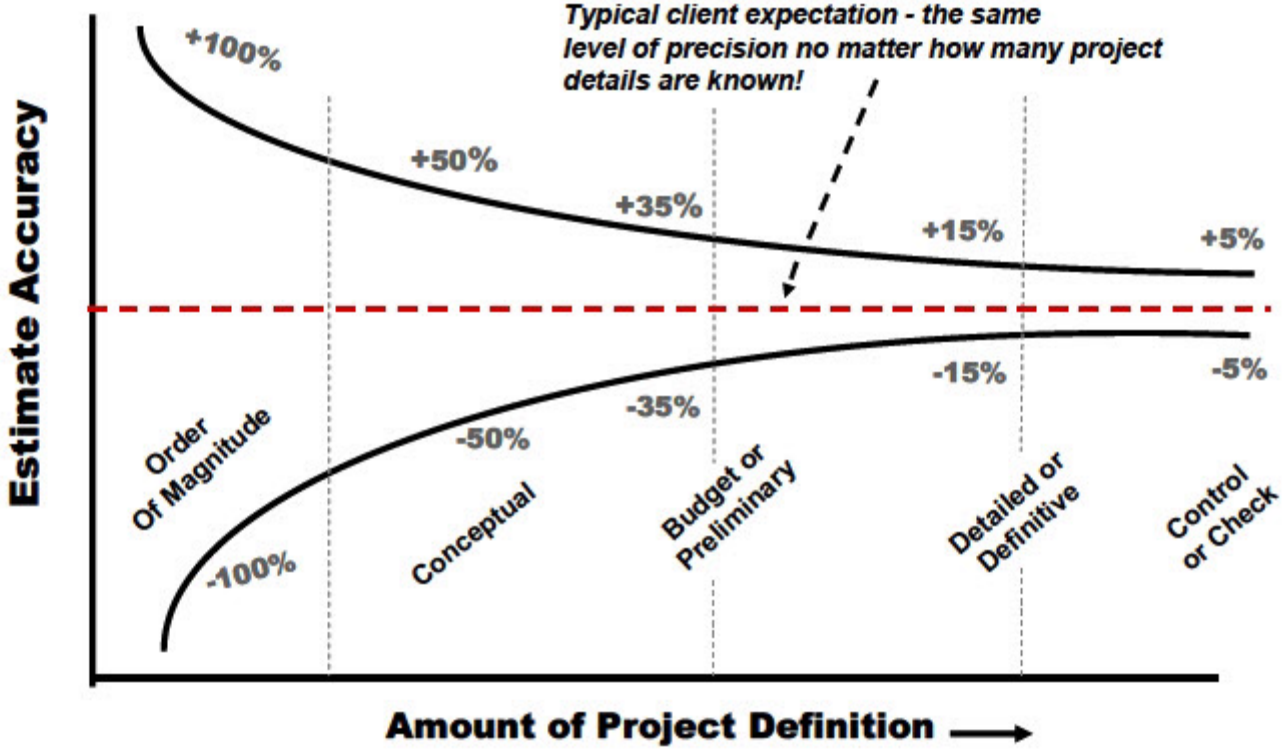


#3

Estimation != Commitment

Iteratively increasing clarity



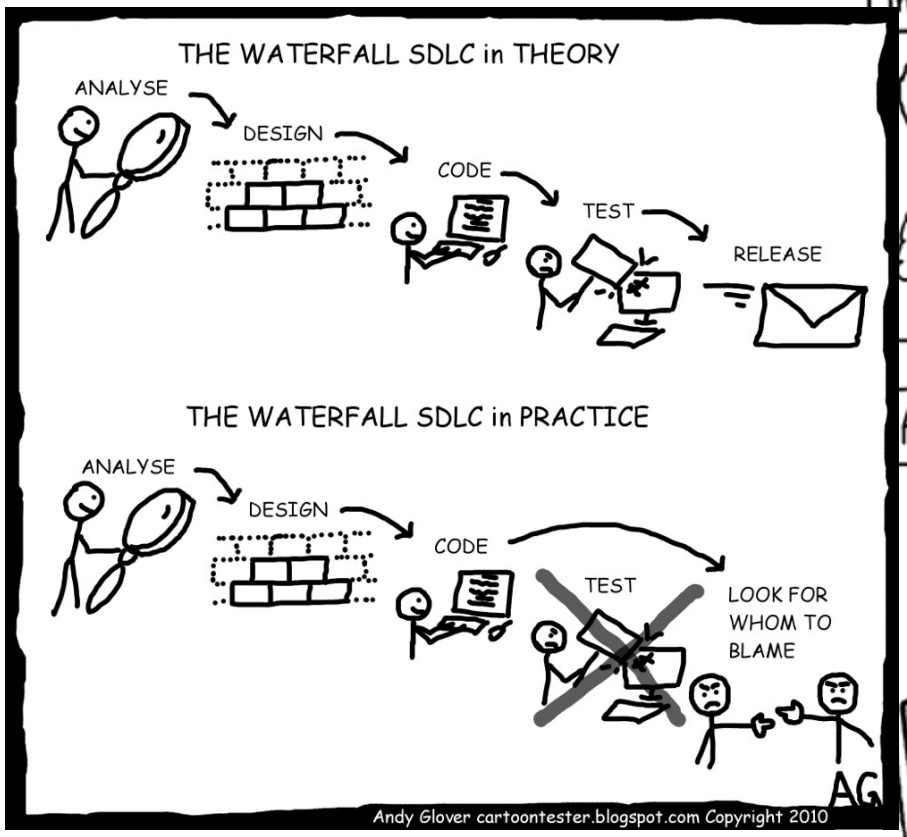


#4

First try to measure, count and compute

Estimate only when necessary

Reality


THE WATERFALL SDLC in THEORY
ANALYSE
DESIGN
CODE
TEST
RELEASE
THE WATERFALL SDLC in PRACTICE
ANALYSE
DESIGN
CODE
TEST
LOOK FOR WHOM TO BLAME
AG
Andy Glover cartoontester.blogspot.com Copyright 2010


Estimation
Inexperienced Developer
that'll take 2 days, tops.
Experienced Developer
that'll take at least 2 weeks.
Inexperienced PM
2 days? Sounds ok.
but I only scheduled a day and a half.
Experienced PM
2 weeks? Sounds ok.
It's going to be at least three.
AND THEN SALES COMES ALONG:
GOOD NEWS! I GOT THE CONTRACT SIGNED! 1 DAY!
Shit.

#5

Aggregate independent estimates

"Wisdom of the Crowds"

Team effort


Project Approval Board
Don't worry! My team is hard at work coming up with an accurate cost estimation for the project.
I say we each get 3 tries and then we average the results.
J. KING
MODERN ANALYST

The law of large numbers
(or: statistics is on our side, for once)

- If we estimate with an error of x%
- The estimate of each scope item will have an error of x%
- But...
- Some items will be over-estimated, others under-estimated (maybe....)

=> The error on the total estimate is likely < x%

Estimation Methodologies

- Top-down
- Bottom-up
- Analogy
- Expert Judgment
- Priced to Win (request for quote – RFQ)
- Parametric or Algorithmic Method
  - Using formulas and equations

Function Point Analysis (FPA)

- Software size measured by number & complexity of functions it performs
- More methodical than LOC counts
- House analogy
  - House's Square Feet ~= Software LOC
  - # Bedrooms & Baths ~= Function points
  - Former is size only, latter is size & function
- Five basic steps

Function Point Analysis - Process

- 1. Count # of business functions per category
  - Categories: **outputs, inputs, DB inquiries, files or data structures, and interfaces**
- 2. Establish Complexity Factor for each and apply
  - Low, Medium, High
  - Set a weighting multiplier for each (0 →15)
  - This results in the "unadjusted function-point total"
- 3. Compute an "influence multiplier" and apply
  - It ranges from 0.65 to 1.35; is based on 14 factors
- 4. Results in "function point total"
  - This can be used in comparative estimates
- 5. Estimating effort and time
  - Calculate based on per function point effort

Example - Online Bookstore Application

- **User Management**: Registration, Login, Profile Update, etc.
- **Product Management**: Adding new books, Editing book details, etc.
- **Order Management**: Placing orders, Viewing order history, etc.
- **Search Functionality**: Searching for books, etc.
- **Reviews and Ratings**: Adding and viewing reviews, etc.

Step 1: Identify and Classify Functions

- Inputs - e.g., User Registration, Adding new books
- Outputs - e.g., Displaying order confirmation
- Inquiries - e.g., Searching for books
- Logical Files - e.g., User account details, Book details
- Interface Files- e.g., Interfaces with a payment gateway

Step 2: Assign Complexity Weights

For example (for inputs):

- Low complexity: 3 FP
- Average complexity: 4 FP
- High complexity: 6 FP

Function point multipliers

	Function Points		
Program Characteristic	Low Complexity	Medium Complexity	High Complexity
Number of Inputs	x 3	x 4	x 6
Number of Outputs	x 4	x 5	x 7
Inquiries	x 3	x 4	x 6
Logical internal files	x 7	x 10	x 15
External interface files	x 5	x 7	x 10

From Rapid Development by Steve McConnell

Counting the Number of Function Points

Step 3: Calculate Unadjusted Function Points (UFP)

Step 4: Calculate Adjusted Function Points (AFP)

Influence multiplier is based on 14 general system characteristics like data communications, distributed functions, performance requirements, etc.

Each characteristic is rated on a scale from 0 (no influence) to 5 (strong influence), and the total is summed up

	Function Points		
Program Characteristic	Low Complexity	Medium Complexity	High Complexity
Number of Inputs	5 x 3 = 15	2 x 4 = 8	3 x 6 =18
Number of Outputs	6 x 4 = 24	6 x 5 = 30	0 x 7 = 0
Inquiries	0 x 3 = 0	2 x 4 = 8	4 x 6 = 24
Logical internal files	5 x 7 = 35	2 x 10 =20	3 x 15 = 45
External interface files	8 x 5 = 40	0 x 7 = 0	2 x 10 = 20
Unadjusted function-point total	287		
Influence multiplier	1.20		
Adjusted function-point total	344		

From Rapid Development by Steve McConnell

Over and Under Estimation

- Over estimation issues
  - The project will not be funded
    - Conservative estimates guaranteeing 100% success may mean funding probability of zero.
  - Danger of feature and scope creep
  - Be aware of "double-padding": team member + manager
- Under estimation issues
  - Quality issues (short changing key phases like testing)
  - Inability to meet deadlines
  - Morale and other team motivation issues

Wideband Delphi

- Group consensus approach
- Present experts with a problem and response form
- Conduct group discussion, collect anonymous opinions, then feedback
- Conduct another discussion & iterate until consensus
- Advantages
  - Easy, inexpensive, utilizes expertise of several people
  - Does not require historical data
- Disadvantages
  - Difficult to repeat
  - May fail to reach consensus, reach wrong one, or all may have same bias

PROBLEM STATEMENT

The system is an on-line version of the popular Monopoly board game. The game provides many of the features found in the board version of the game. Unless otherwise specified this game follows the standard rules of the board game.

FEATURES

Game Initialization
Move Player
Move Players in Turns
Pass Go
Free Parking
Go to Jail
Get Out of Jail
Purchase Property
Pay Rent to Property Owner
Unable to Pay Rent
Trade Properties
Buy Railroad
Pay Rent to Railroad Owner
Buy Utility
Pay Rent to Utility Owner
Buy House
Draw Jail Card
Draw Lose Money Card
Draw Gain Money Card
Draw Move Player Card

Scheduling & Tracking

How To Schedule

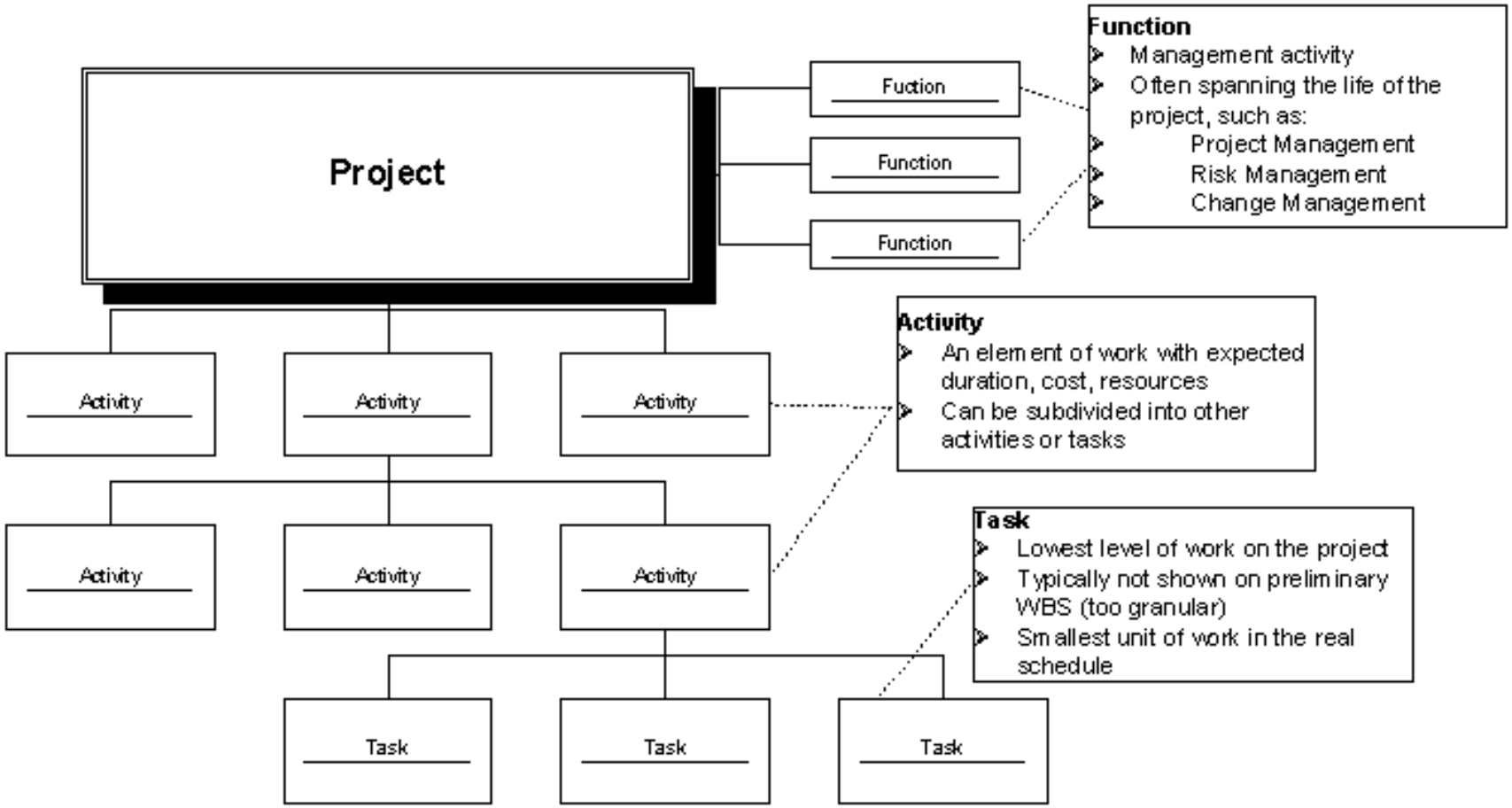
- 1. Identify "what" needs to be done
  - Work Breakdown Structure (WBS)
- 2. Identify "how much" (the size)
  - Size estimation techniques
- 3. Identify the dependency between tasks
  - Dependency graph, network diagram
- 4. Estimate total duration of the work to be done
  - The actual schedule

Partitioning Your Project

- You need to decompose your project into manageable chunks
- ALL projects need this step
- Divide & Conquer
- Two main causes of project failure
  - Forgetting something critical
  - Ballpark estimates become targets
- How does partitioning help this?

Project Elements

- A Project: functions, activities, tasks

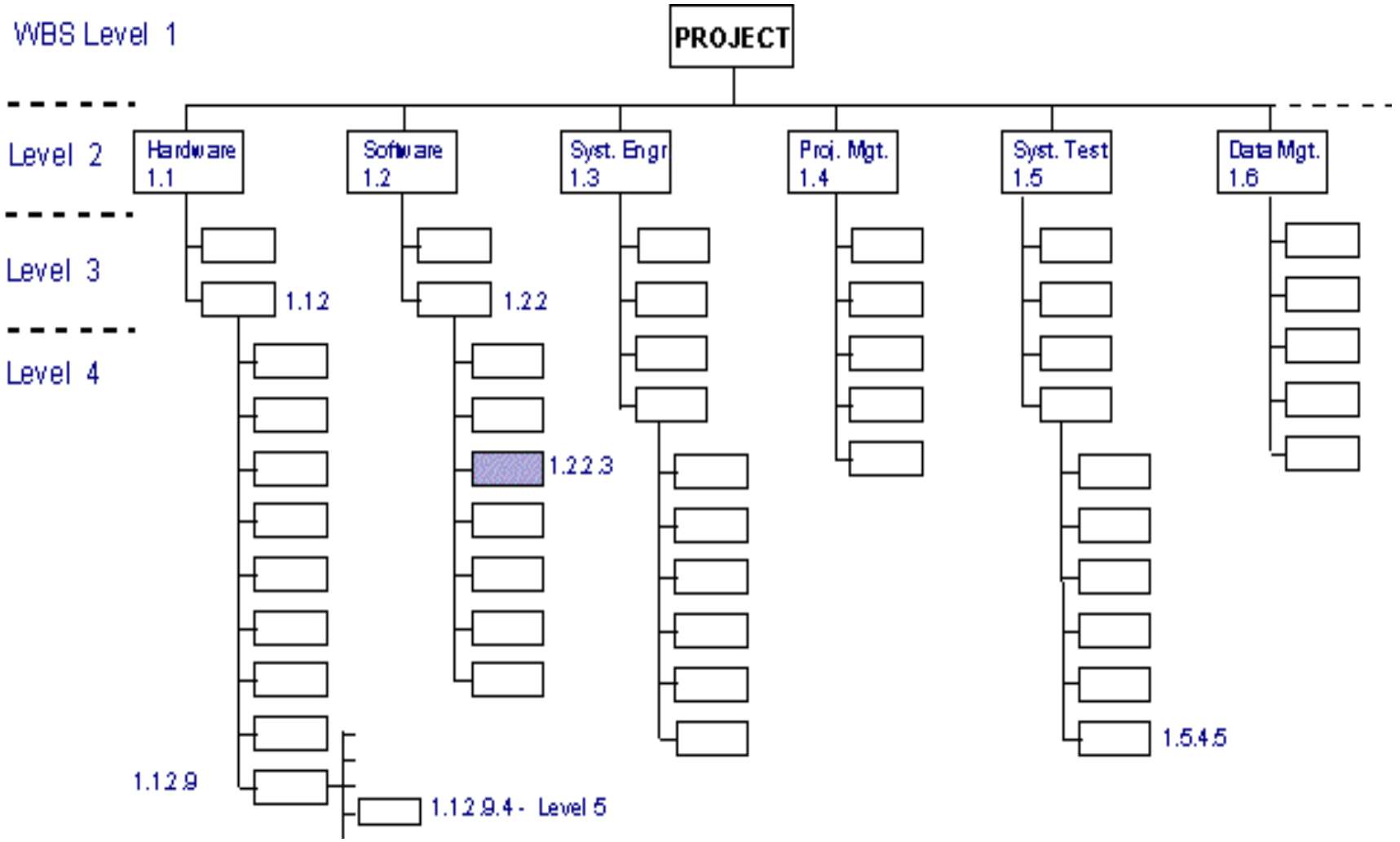


Work Break Down Structure (WBS)

- ***Work Break Down Structure*** – a check list of the work that must be accomplished to meet the project objectives.
- The WBS lists the major project outputs and those departments or individuals primarily responsible for their completion.

WBS Outline Example

0.0 Retail Web Site
1.0 Project Management
2.0 Requirements Gathering
3.0 Analysis & Design
4.0 Site Software Development
    4.1 HTML Design and Creation
    4.2 Backend Software
        4.2.1 Database Implementation
        4.2.2 Middleware Development
        4.2.3 Security Subsystems
        4.2.4 Catalog Engine
        4.2.5 Transaction Processing
    4.3 Graphics and Interface
    4.4 Content Creation
5.0 Testing and Production

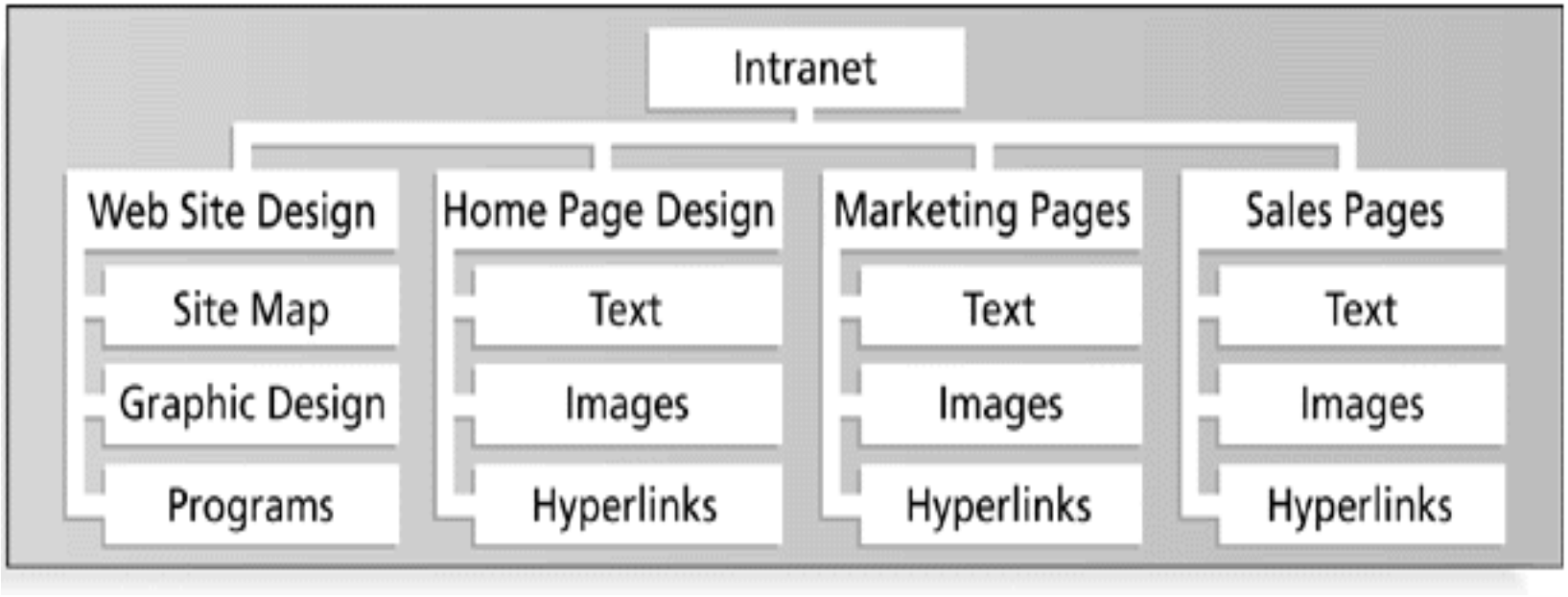


From: http://www.hyperthot.com/pm_wbs.htm

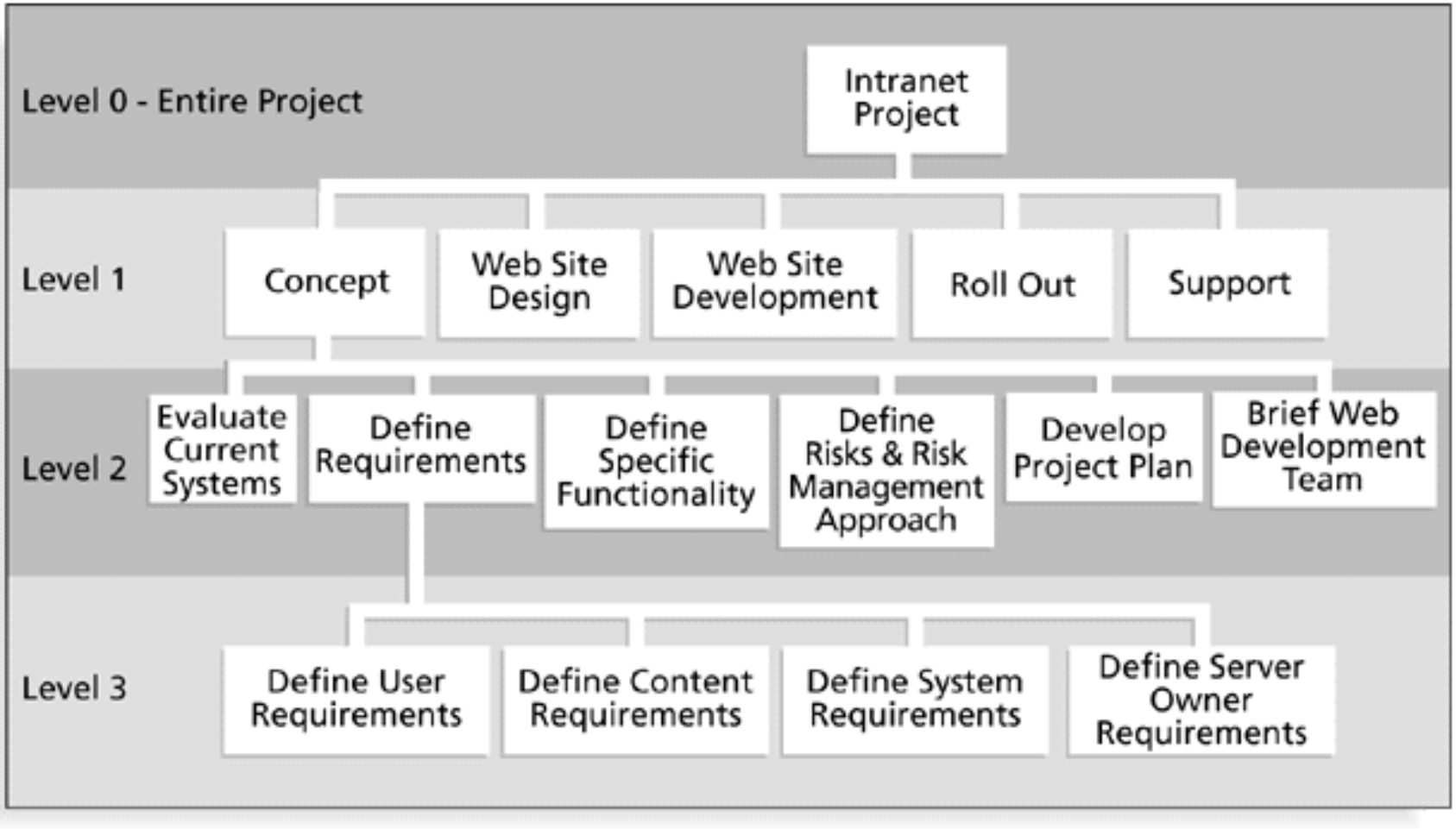
WBS Types

- Process WBS
  - a.k.a Activity-oriented
  - Ex: Requirements, Analysis, Design, Testing
  - Typically used by PM
- Product WBS
  - a.k.a. Entity-oriented
  - Ex: Financial engine, Interface system, DB
  - Typically used by engineering manager
- Hybrid WBS: both above
  - This is not unusual
  - Ex: Lifecycle phases at high level with component or feature-specifics within phases
  - Rationale: processes produce products

Product WBS


Intranet
Web Site Design
Site Map
Graphic Design
Programs
Home Page Design
Text
Images
Hyperlinks
Marketing Pages
Text
Images
Hyperlinks
Sales Pages
Text
Images
Hyperlinks

Process WBS


Level 0 - Entire Project
Intranet Project
Level 1
Concept
Web Site Design
Web Site Development
Roll Out
Support
Level 2
Evaluate Current Systems
Define Requirements
Define Specific Functionality
Define Risks & Risk Management Approach
Develop Project Plan
Brief Web Development Team
Level 3
Define User Requirements
Define Content Requirements
Define System Requirements
Define Server Owner Requirements

WBS

- List of Activities, not Things
- List of items can come from many sources
  - SOW, Proposal, brainstorming, stakeholders, team
- Describe activities using "bullet language"
  - Meaningful but terse labels
- All WBS paths do not have to go to the same level
- Do not plan more detail than you can manage

Work Packages (Tasks)

- Generic term for discrete **tasks** with definable end results
- The "one-to-two" rule
  - Often at: 1 or 2 persons for 1 or 2 weeks
- Basis for monitoring and reporting progress
  - Can be tied to budget items (charge numbers)
  - Resources (personnel) assigned
- Ideally shorter rather than longer
  - Not so small as to micro-manage

WBS Techniques

- Top-Down
- Bottom-Up
- Analogy
- Rolling Wave
  - 1st pass: go 1-3 levels deep
  - Gather more requirements or data
  - Add more detail later
- Post-its on a wall

WBS Techniques

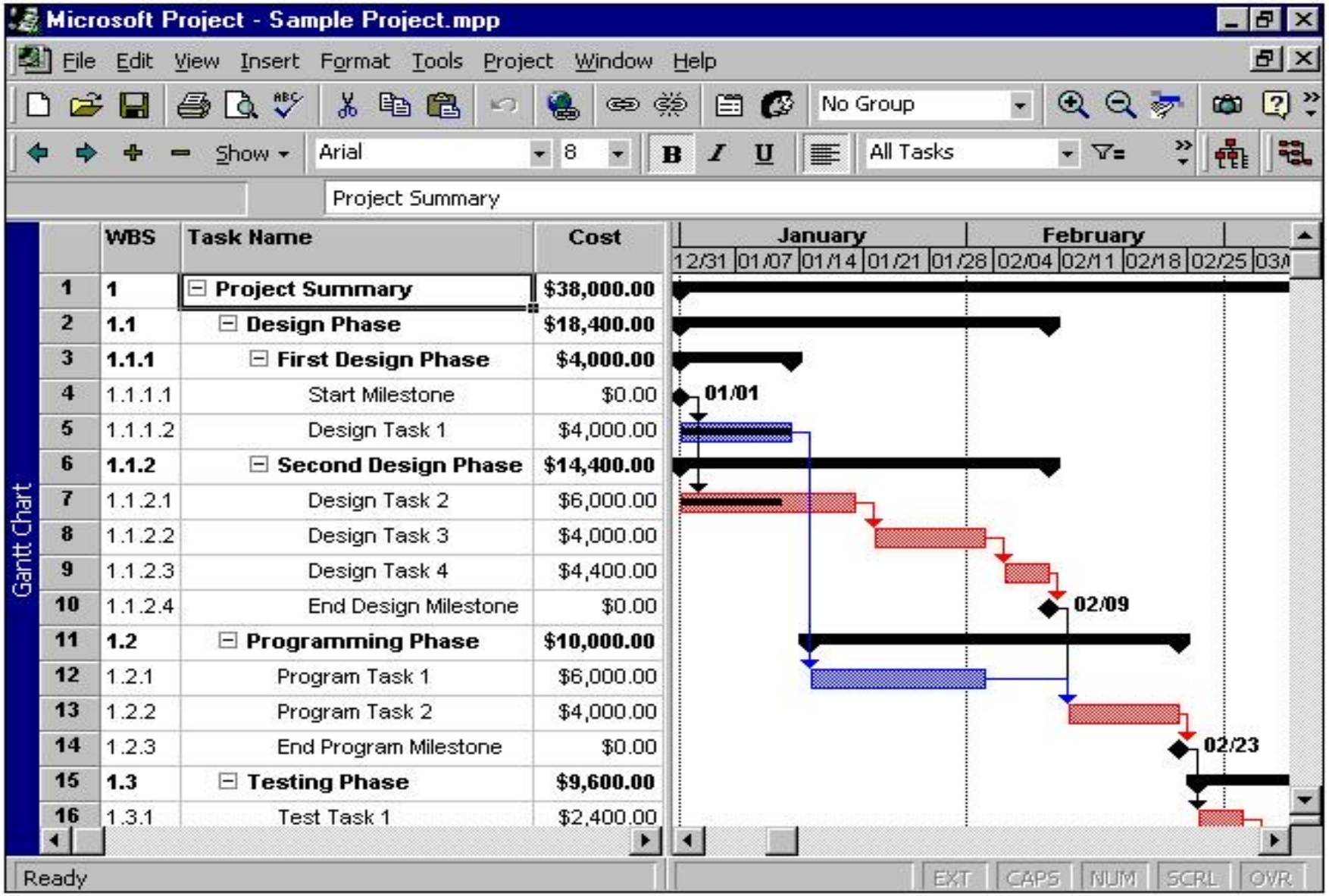
- Analogy
  - Base WBS upon that of a "similar" project
  - Use a template
  - Analogy also can be estimation basis
  - Pros
    - Based on past actual experience
  - Cons
    - Needs comparable project

Sequence the Work Activities

- Milestone Chart
- Gantt chart
- Network Techniques
  - CPM (Critical Path Method)
  - PERT (Program Evaluation and Review Technique)

Gantt Chart

- Gantt chart is a means of displaying simple activities or events plotted against time or dollars
- Most commonly used for exhibiting program progress or for defining specific work required to reach an objective
- Gantt charts may include listing of activities, activity duration, scheduled dates, and progress-to-date


Microsoft Project - Sample Project.mpp
File Edit View Insert Format Tools Project Window Help
No Group
Show
Arial
8
All Tasks
Project Summary
Gantt Chart
WBS
Task Name
Cost
1 1 Project Summary $38,000.00
2 1.1 Design Phase $18,400.00
3 1.1.1 First Design Phase $4,000.00
4 1.1.1.1 Start Milestone $0.00
5 1.1.1.2 Design Task 1 $4,000.00
6 1.1.2 Second Design Phase $14,400.00
7 1.1.2.1 Design Task 2 $6,000.00
8 1.1.2.2 Design Task 3 $4,000.00
9 1.1.2.3 Design Task 4 $4,400.00
10 1.1.2.4 End Design Milestone $0.00
11 1.2 Programming Phase $10,000.00
12 1.2.1 Program Task 1 $6,000.00
13 1.2.2 Program Task 2 $4,000.00
14 1.2.3 End Program Milestone $0.00
15 1.3 Testing Phase $9,600.00
16 1.3.1 Test Task 1 $2,400.00
January
February
12/31 01/07 01/14 01/21 01/28 02/04 02/11 02/18 02/25
01/01
02/09
02/23
Ready
EXT CAPS NUM SCRL OVR

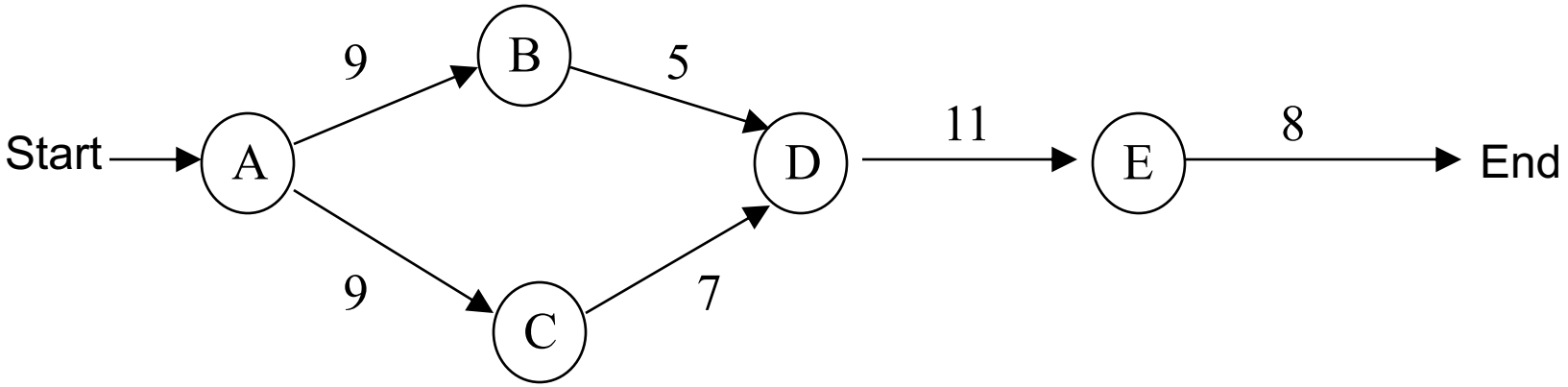
Gantt Chart

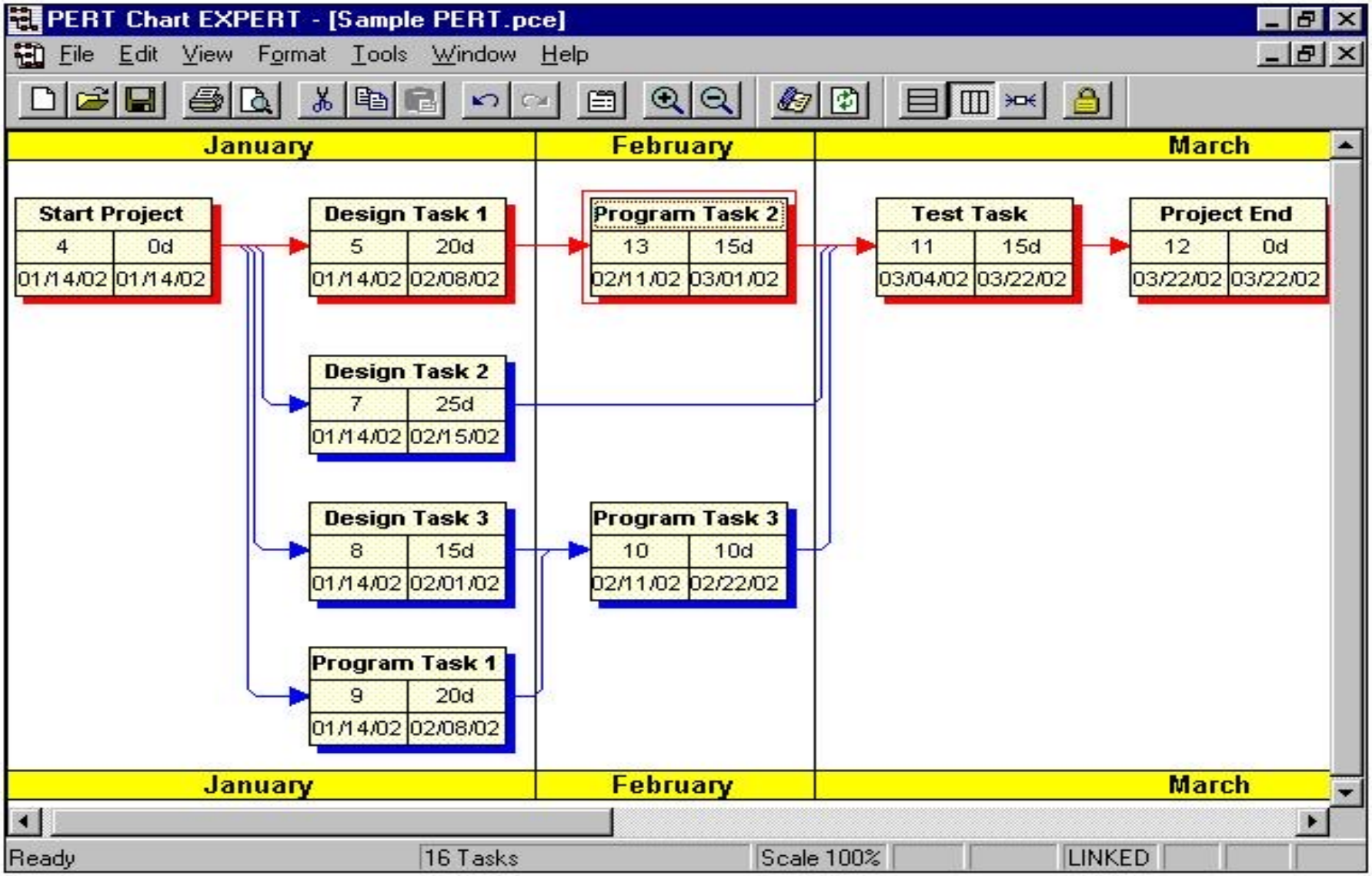
- Advantages:
  - Easy to understand
  - Easy to change
- Disadvantages:
  - only a vague description of the project
  - does not show interdependency of activities
  - cannot show results of an early or late start of an activity

Network Techniques

- A ***precedence network*** diagram is a graphic model portraying the sequential relationship between key events in a project.
- Initial development of the network requires that the project be defined and thought out.
- The network diagram clearly and precisely communicates the plan of action to the project team and the client.

Task	Duration	Dependencies
A - Architecture & design strategy	9	start
B - Decide on number of releases	5	A
C - Develop acceptance test plan	7	A
D - Develop customer support plan	11	B,C
E - Final sizing & costing	8	D




PERT Chart EXPERT - [Sample PERT.pce]
File Edit View Format Tools Window Help
January
February
March
Start Project
4 0d
01/14/02 01/14/02
Design Task 1
5 20d
01/14/02 02/08/02
Program Task 2
13 15d
02/11/02 03/01/02
Test Task
11 15d
03/04/02 03/22/02
Project End
12 0d
03/22/02 03/22/02
Design Task 2
7 25d
01/14/02 02/15/02
Design Task 3
8 15d
01/14/02 02/01/02
Program Task 3
10 10d
02/11/02 02/22/02
Program Task 1
9 20d
01/14/02 02/08/02
January
February
March
Ready
16 Tasks
Scale 100%
LINKED

Critical Path Method (CPM)

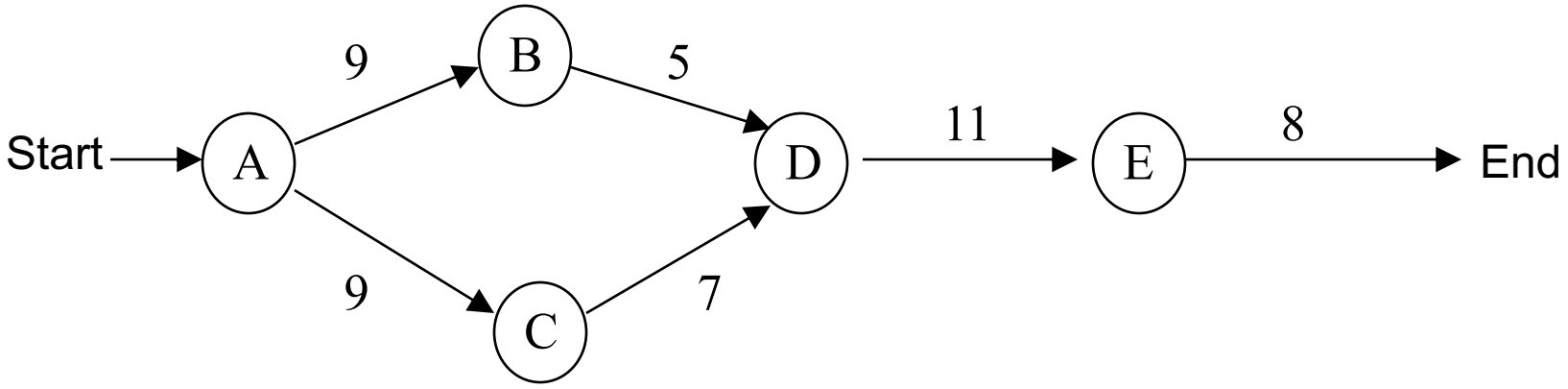
CPM tries to answer the following questions:

1. What is the duration of the project?
2. By how much (if at all) will the project be delayed if any one of the activities takes N days longer?
3. How long can certain activities be postponed without increasing the total project duration?

Critical Path

- Sequence of activities that have to be executed one after another
- Duration times of these activities will determine the overall project time, because there is no slack/float time for these activities
- If any of the activities on the critical path takes longer than projected, the entire project will be delayed by that same amount
- Critical path = Longest path in the precedence network (generally, the longest in time)

Critical Path

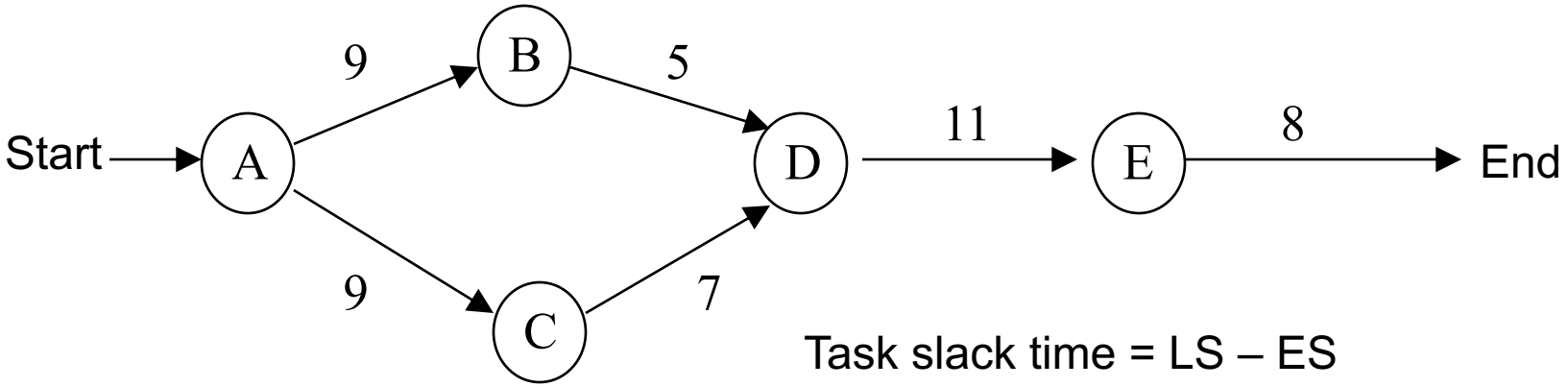


Critical Path = A – C – D – E (35 time units)

Critical Tasks = A,C,D,E

Non-Critical Path = A-B-D-E

Task	Duration	Depend	Earliest Start	Earliest Finish	Latest Start	Latest Finish
A	9	none	0	9	0	9
B	5	A	9	**14**	**11**	16
C	7	A	9	16	9	16
D	11	B,C	16	27	16	27
E	8	D	27	35	27	35



Slack time – maximum allowable delay for a non-critical activity.

Task slack time = LS – ES

- or -

Task slack time = LF - EF

Task B has 2 time units of ***slack time***

Requirements Gathering and Analysis (Week 4)

Some content adapted from Rajib Mall's book and Craig Larman's book.

Requirements Phase

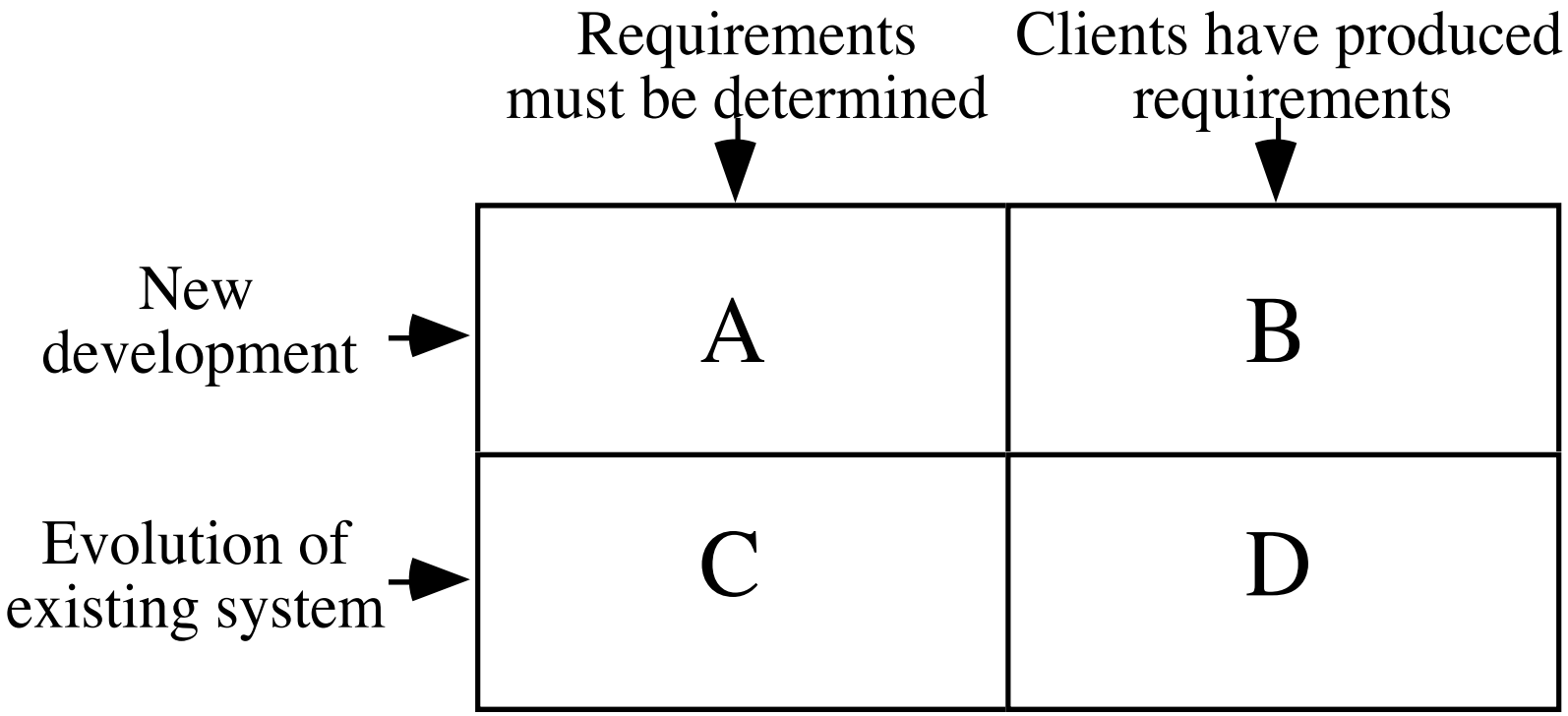
- Many projects fail:
  - Because they start implementing the system.
  - Without determining whether they are building what the customer really wants.

Why Requirements analysis and specification?

- Factors that cause projects to fail:
  - Lack of User Input 12.8%
  - Incomplete Requirements & Specifications 12.3%
  - Changing Requirements & Specifications 11.8%
  - Lack of Executive Support 7.5%
  - Technology Incompetence 7.0%
  - Lack of Resources 6.4%
  - Unrealistic Expectations 5.9%
  - Unclear Objectives 5.3%
  - Unrealistic Time Frames 4.3%
  - New Technology 3.7%
  - Other 23.0%

• Standish Group, *The Standish Group Report: Chaos*, 1995, http://www.scs.carleton.ca/~beau/PM/Standish-Report.html

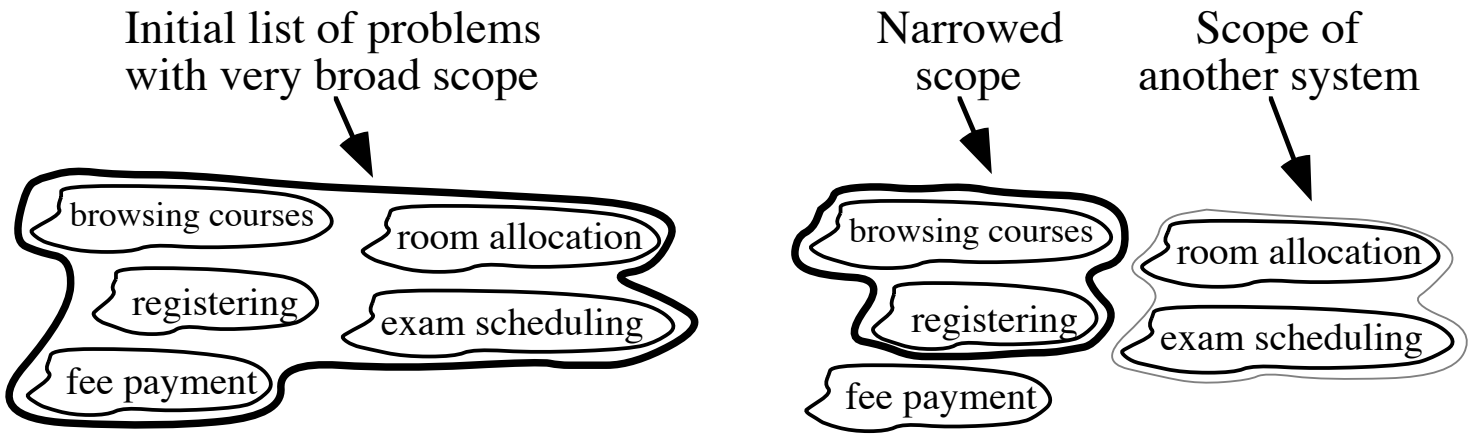
The Starting Point for Software Projects



Defining the Scope

- Narrow the *scope* by defining a more precise problem
  - List all the things you might imagine the system doing
    - Exclude some of these things if too broad
    - Determine high-level goals if too narrow

- Example: A university registration system



What is a Requirement?

- Requirement: A statement about the proposed system that all stakeholders agree must be made true in order for the customer's problem to be adequately solved.
  - Short and concise piece of information
  - Says something about the system
  - All the stakeholders have agreed that it is valid
  - It helps solve the customer's problem

- A collection of requirements is a *requirements document*.

Types of Requirements

- Business requirements
  - High-level objectives of the organization or customer who requests the system.

- Functional requirements
  - Describe *what* the system should do

  For example, features (use cases)

- Non-functional requirements
  - C*onstraints* that must be adhered to during development

  For example, quality constraints, technology constraints, process constraints, etc.

Requirements Phase

- Goals of requirements phase:
  - Fully understand the user requirements.
  - Remove inconsistencies, anomalies, etc. from requirements.
  - Document requirements properly in an SRS document.

Requirements Phase

- Consists of two distinct activities:
  - Requirements Gathering and Analysis
  - Requirements Specification

Requirements Gathering

- Also known as requirements elicitation.
- If the project is to automate some existing procedures
  - e.g., automating existing manual accounting activities,
  - The task of the system analyst is a little easier
  - Analyst can immediately obtain:
    - input and output formats
    - accurate details of the operational procedures

Requirements Gathering (CONT.)

- In the absence of a working system,
  - Lot of imagination and creativity are required.
- Interacting with the customer to gather relevant data:
  - Requires a lot of experience.

Analysis of the gathered requirements

- Main purpose of requirements analysis:
  - Clearly understand the user requirements,
  - Detect inconsistencies, ambiguities, and incompleteness.
- Incompleteness and inconsistencies:
  - Resolved through further discussions with the end-users and the customers.

Inconsistent Requirement

- Some part of the requirement:
  - contradicts with some other part.
- Example (E-commerce site):

1. **Product Availability**:
   1. Requirement A: Products are displayed as available even if there's only one left in stock to create a sense of urgency.
   2. Requirement B: Users cannot add a product to their cart if the stock quantity is less than five.
2. **Pricing Rules**:
   1. Requirement C: Discounted prices should be displayed with the original price crossed out and the new discounted price prominently shown.
   2. Requirement D: Prices are automatically adjusted based on user behavior, with the goal of maximizing revenue

Incomplete Requirement

- Some requirements have been omitted:
  – Possibly due to oversight.

- Example (E-commerce site):

  - Implement user authentication for the e-commerce website.

Enhanced Requirement:

Precondition: User must be registered in the system via an email id and password.

"Users should be able to log in with their registered email and password. After three unsuccessful login attempts, users should be locked out of their accounts for 30 minutes. Implement password reset functionality that sends a password reset link to the user's registered email address"

Analysis of the gathered requirements (contd.)

- Requirements analysis involves:
  - Obtaining a clear, in-depth understanding of the product to be developed,
  - Remove all ambiguities and inconsistencies from the initial customer perception of the problem.

Analysis of gathered requirements (contd.)

- Experienced analysts take considerable time:
  - To understand the exact requirements the customer has in his mind.
- Experienced systems analysts know - often as a result of past (painful) experiences

Analysis of gathered requirements (contd.)

- Several things about the project should be clearly understood by the analyst:
  - What is the problem?
  - Why is it important to solve the problem?
  - What are the possible solutions to the problem?
  - What complexities might arise while solving the problem?

Analysis of gathered requirements (contd.)

- After collecting all data regarding the system to be developed,
  - Remove all inconsistencies and anomalies from the requirements,
  - Systematically organize requirements into a Software Requirements Specification (SRS) document.

Bad Requirements: A Simplified Example

- *A mail should be displayed within 3 seconds of clicking on mail*
- *User should be able to add a new mail server during peak hours within a small downtime*
- *Business services should not be interrupted during the peak hours*
- *User should be able to customize all the mailbox settings*
- *User should be able to change the look and feel of how the mailbox is displayed*

Quality Requirements

- Correct – only user representative can determine
- Feasible – get reality check on what can or cannot be done technically or within given cost constraints.
- Necessary – trace each requirement back to its origin
- Unambiguous – one interpretation
- Verifiable – how to you know if the requirement was implemented properly?
- Prioritized – function of value provided to the customer

Writing Example #1

"The product shall provide status messages at regular intervals not less than every 60 seconds."

Writing Example #1

"The product shall provide status messages at regular intervals not less than every 60 seconds."

- Incomplete – What are the status messages and how are they supposed to be displayed?
- Ambiguous – What part of the product? Regular interval?
- Not verifiable

Alternative #1

1. Status Messages.

1.1. The Background Task Manager shall display status messages in a designated area of the user interface at intervals of 60 plus or minus 10 seconds.

1.2. If background task processing is progressing normally, the percentage of the background task processing that has been completed shall be displayed.

1.3. A message shall be displayed when the background task is completed.

1.4. An error message shall be displayed if the background task has stalled.

Writing Example #2

“The product shall switch between displaying and hiding non-printing characters instantaneously."

Writing Example #2

"T*he product shall switch between displaying and hiding non-printing characters instantaneously."*

- Not Feasible – computers cannot do anything instantaneously.
- Incomplete – conditions which trigger state switch
- Ambiguous – "non-printing character"

Alternative #2

"The user shall be able to toggle between displaying and hiding all HTML markup tags in the document being edited with the activation of a specific triggering condition."

- Note that "triggering condition" is left for design

The Specification Trap

The Landing Pilot is the Non-Landing Pilot until the 'decision altitude' call, when the Handling Non-Landing Pilot hands the handling to the Non-Handling Landing Pilot, unless the latter calls 'go-around,' in which case the Handling Non-Landing Pilot continues handling and the Non-Handling Landing Pilot continues non-handling until the next call of 'land,' or 'go-around' as appropriate . In view of recent confusions over these rules, it was deemed necessary to restate them clearly.

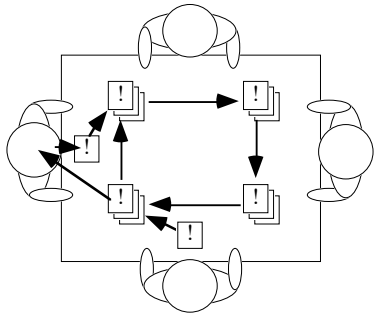
- British Airways memorandum, quoted in *Pilot Magazine*.

Techniques - Gathering and Analyzing Requirements

- Observation
  - Read documents and discuss requirements with users
  - Shadowing important potential users as they do their work
    - ask the user to explain everything he or she is doing
  - Session videotaping
- Interviewing
  - Conduct a series of interviews
    - Ask about specific details
    - Ask about the stakeholder's vision for the future
    - Ask if they have alternative ideas
    - Ask for other sources of information
    - Ask them to draw diagrams

Gathering and Analyzing Requirements

- Brainstorming
  - Appoint an experienced moderator
  - Arrange the attendees around a table
  - Decide on a ‘trigger question’
  - Ask each participant to write an answer and pass the paper to its neighbour



- ***Joint Application Development (JAD)*** **is a technique based on intensive brainstorming sessions**

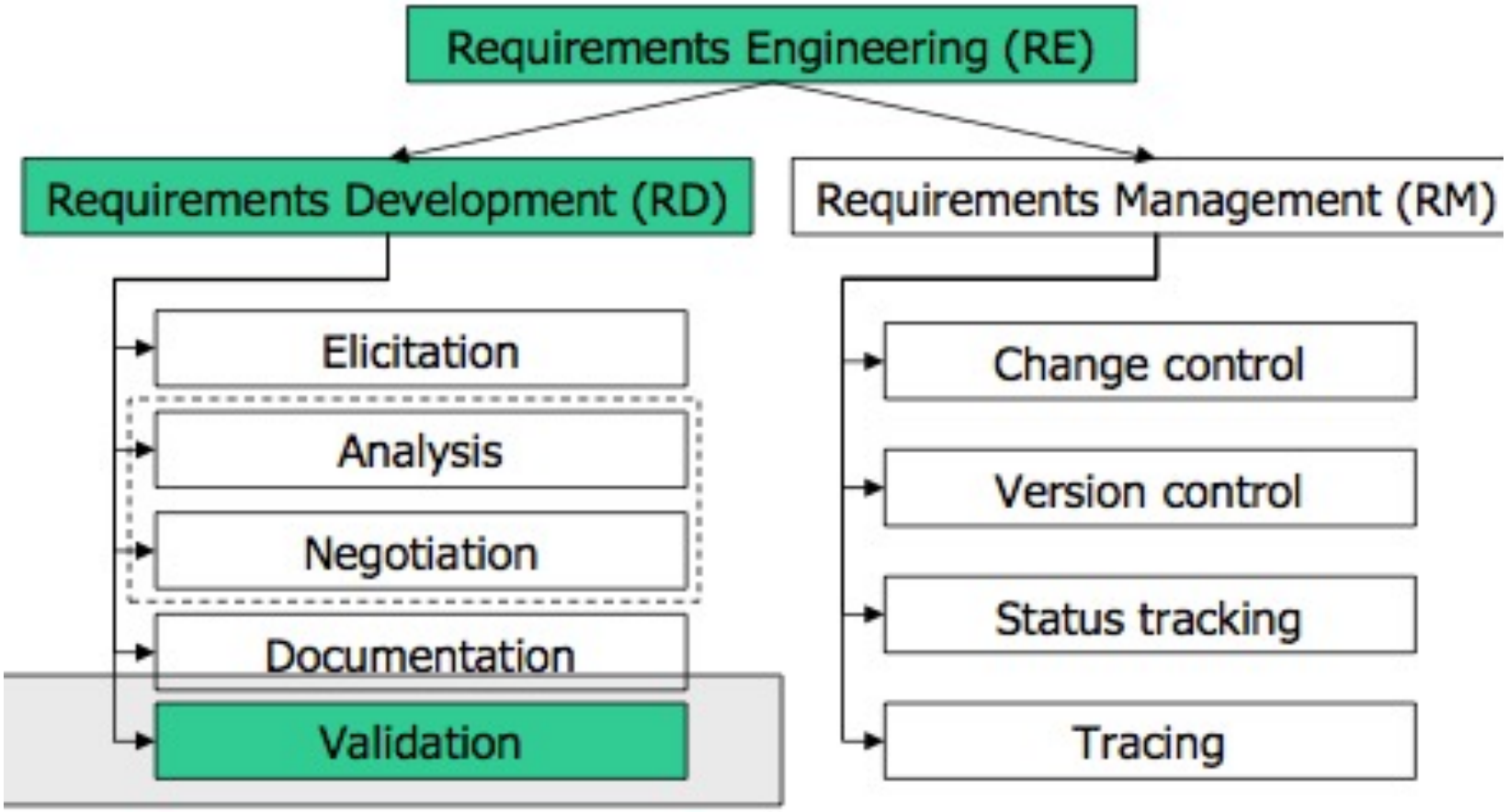
Gathering and Analyzing Requirements

- Prototyping
  - The simplest kind: *paper prototype*.
    - a set of pictures of the system that are shown to users in sequence to explain what would happen
  - The most common: a mock-up of the system's UI
    - Written in a rapid prototyping language
    - Does *not* normally perform any computations, access any databases or interact with any other systems
    - May prototype a particular aspect of the system

Difficulties and Risks in Domain and Requirements analysis

- Lack of understanding of the domain or the real problem
  - *Do domain analysis and prototyping*
- Requirements change rapidly
  - *Perform incremental development, build flexibility into the design, do regular reviews*
- Attempting to do too much
  - *Document the problem boundaries at an early stage, carefully estimate the time*
- It may be hard to reconcile conflicting sets of requirements
  - *Brainstorming, JAD sessions, competing prototypes*
- It is hard to state requirements precisely
  - *Break requirements down into simple sentences and review them carefully, look for potential ambiguity, make early prototypes*

Requirements Processes



Requirements Specification

(Week 4)

Software Requirements Specification (SRS)

- Main aim of requirements specification:
  - Systematically organize the requirements arrived during requirements analysis.
  - Document requirements properly.

Software Requirements Specification

- The SRS document is useful in various contexts:
  - Statement of user needs
  - Contract document
  - Reference document
  - Definition for implementation

Software Requirements Specification: A Contract Document

- Requirements document is a reference document.
- SRS document is a contract between the development team and the customer.
  - Once the SRS document is approved by the customer,
    - Any subsequent controversies are settled by referring the SRS document.

Software Requirements Specification: A Contract Document

- Once customer agrees to the SRS document:
  - Development team starts to develop the product according to the requirements recorded in the SRS document.
- The final product will be acceptable to the customer:
  - As long as it satisfies all the requirements recorded in the SRS document.

SRS Document (CONT.)

- The SRS document is known as black-box specification:
  - The system is considered as a black box whose internal details are not known.
  - Only its visible external (i.e. input/output) behavior is documented.

SRS Document (CONT.)

- SRS document concentrates on:
  - What needs to be done
  - Carefully avoids the solution ("how to do") aspects.
- The SRS document serves as a contract
  - Between development team and the customer.
  - Should be carefully written

SRS Document (CONT.)

- The requirements at this stage:
  - Written using end-user terminology.
- If necessary:
  - Later a formal requirement specification may be developed from it.

Properties of a Good SRS Document

- It should be concise
  - and at the same time should not be ambiguous.
- It should specify what the system must do
  - and not say how to do it.
- Easy to change.,
  - i.e. it should be well-structured.
- It should be consistent
- It should be complete

Properties of a Good SRS Document (cont...)

- It should be traceable
  - You should be able to trace which part of the specification corresponds to which part of the design, code, etc and vice versa.
- It should be verifiable
  - e.g. “system should be user friendly” is not verifiable

SRS Document (CONT.)

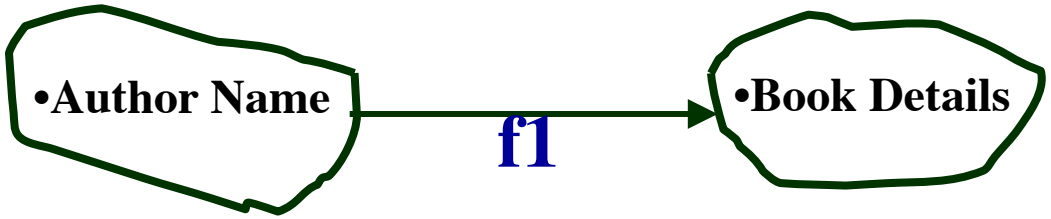
- SRS document, normally contains three important parts:
  - Functional requirements,
  - Non-functional requirements,
  - Goals of Implementation.

SRS Document (CONT.)

- It is desirable to consider every system:
  - Performing a set of functions {fi}.
  - Each function fi considered as:
  - Transforming a set of input data to corresponding output data.

Example: Functional Requirement

- F1: Search Book
  - Input:
    - an author's name:
  - Output:
    - details of the author's books and the locations of these books in the library.



Functional Requirements

- Functional requirements describe:
  - A set of high-level requirements
  - Each high-level requirement:
    - takes in some data from the user
    - outputs some data to the user
  - Each high-level requirement:
    - might consist of a set of identifiable functions

Functional Requirements

- For each high-level requirement:
  - Every function is described in terms of:
    - Input data set
    - Output data set
    - Processing required to obtain the output data set from the input data set.

Nonfunctional Requirements

- Characteristics of the system which can not be expressed as functions:
  - Maintainability,
  - Portability,
  - Usability, etc.

Nonfunctional Requirements

- Nonfunctional requirements include:
  - Reliability issues,
  - Performance issues:
    - Example: How fast the system can produce results
      - so that it does not overload another system to which it supplies data, etc.
  - Human-computer interface issues,
  - Interface with other external systems,
  - Security, maintainability, etc.

Non-Functional Requirements

- Hardware to be used,
- Operating system
  - or DBMS to be used
- Capabilities of I/O devices
- Standards compliance
- Data representations
  - by the interfaced system

Goals of Implementation

- Goals describe things that are desirable of the system:
  - But, would not be checked for compliance.
  - For example,
    - Reusability issues
    - Functionalities to be developed in future

Organization of the SRS Document

- Introduction.
- Functional Requirements
- Nonfunctional Requirements
  - External interface requirements
  - Performance requirements
- Goals of implementation

Functional Requirements

- A high-level function is one:
  - Using which the user can get some useful piece of work done.
- Can the receipt printing work during withdrawal of money from an ATM:
  - Be called a useful piece of work?
- A high-level requirement typically involves:
  - Accepting some data from the user,
  - Transforming it to the required response, and then
  - Outputting the system response to the user.

High-Level Function

- A high-level function:
  - Usually involves a series of interactions between the system and one or more users.
- Even for the same high-level function,
  - There can be different interaction sequences (or scenarios)
  - Due to users selecting different options or entering different data items.

Example Functional Requirements

- List all functional requirements
  - with proper numbering.
- Req. 1:
  - Once the user selects the "search" option,
    - he is asked to enter the key words.
  - The system should output details of all books
    - whose title or author name matches any of the key words entered.
    - Details include: Title, Author Name, Publisher name, Year of Publication, ISBN Number, Catalog Number, Location in the Library.

Example Functional Requirements

- Req. 2:
  - When the “renew” option is selected,
    - The user is asked to enter his membership number and password.
  - After password validation,
    - The list of the books borrowed by him are displayed.
  - The user can renew any of the books:
    - By clicking in the corresponding renew box.

Req. 1:

- R.1.1:
  - Input: “search” option,
  - Output: user prompted to enter the key words.
- R1.2:
  - Input: key words
  - Output: Details of all books whose title or author name matches any of the key words.
    - Details include: Title, Author Name, Publisher name, Year of Publication, ISBN Number, Catalog Number, Location in the Library.
  - Processing: Search the book list for the keywords

Alternatively/Additionally…

- **Use cases** can be used for representing functional requirements

Req. 2:

- R2.1:
  - Input: “renew” option selected,
  - Output: user prompted to enter his membership number and password.
- R2.2:
  - Input: membership number and password
  - Output:
    - list of the books borrowed by user are displayed. User prompted to enter books to be renewed or
    - user informed about bad password
  - Processing: Password validation, search books issued to the user from borrower list and display.

Req. 2:

- R2.3:
  - Input: user choice for renewal of the books issued to him through mouse clicks in the corresponding renew box.
  - Output: Confirmation of the books renewed
  - Processing: Renew the books selected by the in the borrower list.

Examples of Bad SRS Documents

- Unstructured Specifications:
  - Narrative essay --- one of the worst types of specification document:
    - Difficult to change,
    - Difficult to be precise,
    - Difficult to be unambiguous,
    - Scope for contradictions, etc.

Examples of Bad SRS Documents

- Noise:
  - Presence of text containing information irrelevant to the problem.
- Silence:
  - aspects important to proper solution of the problem are omitted.

Examples of Bad SRS Documents

- Overspecification:
  - Addressing "how to" aspects
  - For example, "Library member names should be stored in a sorted descending order"
  - Overspecification restricts the solution space for the designer.
- Contradictions:
  - Contradictions might arise
    - if the same thing described at several places in different ways.

Examples of Bad SRS Documents

- Ambiguity:
  - Literary expressions
  - Unquantifiable aspects, e.g. “good user interface”
- Forward References:
  - References to aspects of problem
    - defined only later on in the text.
- Wishful Thinking:
  - Descriptions of aspects
    - for which realistic solutions will be hard to find.

User Stories – example structure

As a [Type of USER],

[Function to Perform (some goal)]

so that [Business Value (some reason)]

Example: As a user, I can indicate folders not to backup so that my backup isn't filled up with things I don't need saved