

# Comp8081

# Management Issues in Software Engineering

Donna Turner

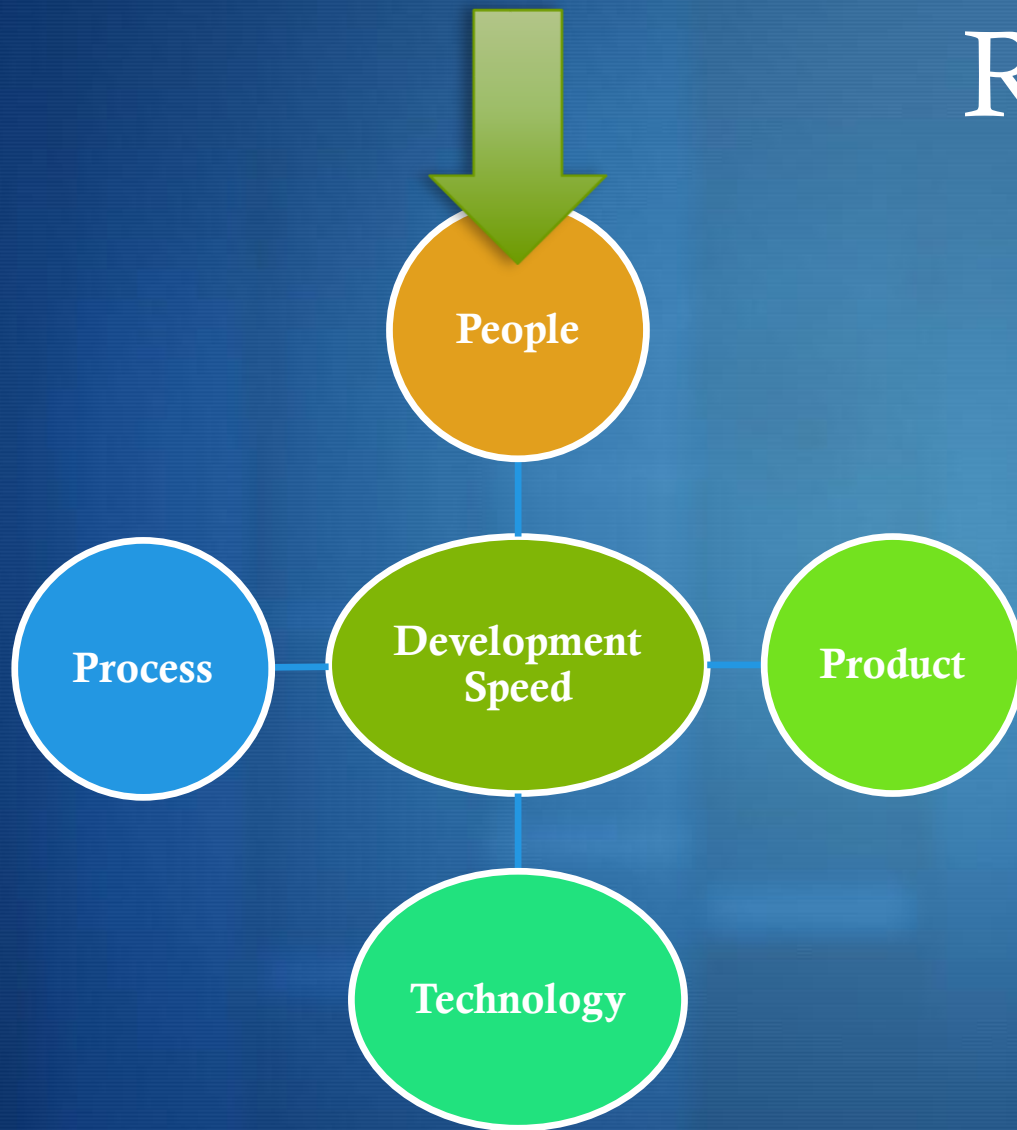


# Agenda

- ◆ Attendance
- ◆ Review: Motivation (McConnell Chapter 11)
- ◆ Feature Set Control and Project Recovery (McConnell Chapters 14 & 16)
- ◆ Form Assignment 3 Groups
- ◆ Next Week

# Review

Motivation

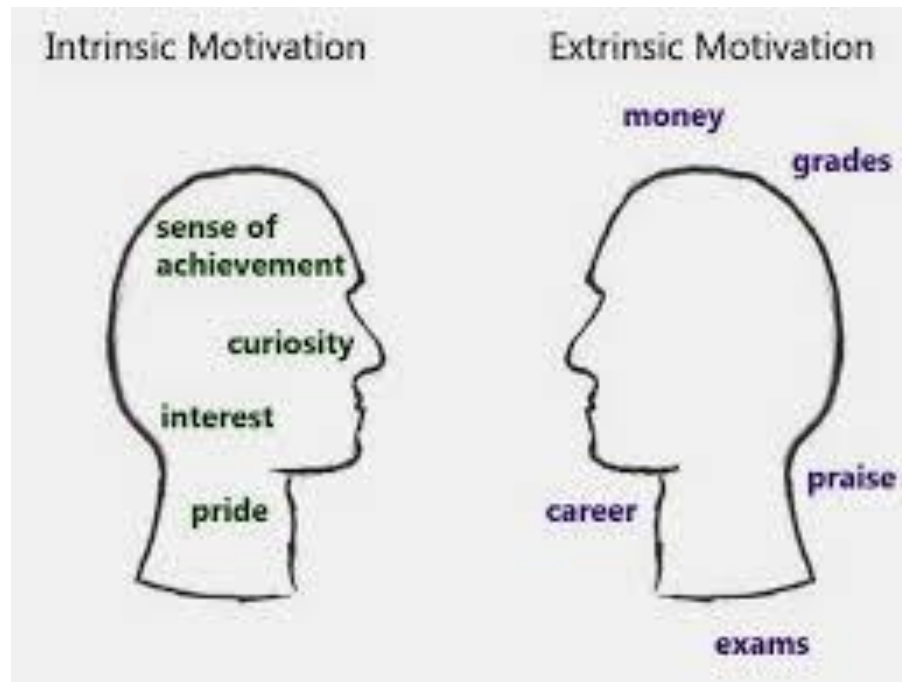


# Motivation

Some questions we discussed

- What do we mean by “motivation”?
- Where does it come from?
- Can you give “it” to, or take it away from someone else?
- Can you create it in yourself when it’s not there? How? Why not?
- What is the most effective motivator and demotivator?

# Types of Motivation



# Videos:

- Dan Pink, Ted Talk
- Atlassian
- Kimberly-Clark

## Engagement:

- 💧 Rewards Work for Focused/Straightforward Work  
(20<sup>th</sup> Century Work)
- 💧 Self Direction Works Better for Creative Work  
(21<sup>st</sup> Century Work)

## Three Building Blocks of Motivation:

1. Autonomy – urge to direct our own lives
2. Mastery – desire to get better and better at something that matters
3. Purpose – yearning to work in the service of something larger than ourselves

# Top 5 “Developer” Motivation Factors

McConnell, p.252

Achievement	Growth Opportunities	The Work Itself	Personal Life	Technical-Supervision
<ul style="list-style-type: none"><li>• Ownership</li><li>• Goal-setting</li></ul>	<ul style="list-style-type: none"><li>• Tuition/PD coverage</li><li>• PD time</li><li>• Projects that provide learning opportunities</li><li>• Mentorship – to be mentored</li></ul>	<ul style="list-style-type: none"><li>• Skill variety</li><li>• Task identity</li><li>• Task significance</li><li>• Autonomy</li><li>• Feedback</li><li>• Low friction on admin</li></ul>	<ul style="list-style-type: none"><li>• We might now call this “work-life balance”</li></ul>	<ul style="list-style-type: none"><li>• Provide opportunities to be a technical lead</li><li>• For a product or for process area</li><li>• Mentoring – to be a mentor</li></ul>

Revisiting this list

- (i) what factor(s) is motivating for you, and
- (ii) which factors are controlled by you, or
- (iii) controlled by the organization you work for?



# De-Motivators

## Herzberg's Two Factor Theory of Hygiene and Motivation



- Hygiene factors
  - Safe work environment
  - Appropriate lighting, heating, and air conditioning
  - Adequate desk and shelf space
  - Readily available office supplies
  - Adequate equipment, communications



# De-Motivators

- 💧 Excessive schedule pressure
- 💧 Micro management
- 💧 Lack of appreciation
- 💧 Heavy handed motivation campaigns
- 💧 Technically inept management
- 💧 Surprise!
- 💧 Public reprimand



# Two Questions

What are your responsibilities around your own motivation and engagement?

What should you do when you recognize you are no longer motivated or engaged?

# Feature Set Control

McConnell Chapter 14



# Aside – Formal Requirements

We will look at two types of requirements:

- ◆ System Requirements
- ◆ User Based Requirements
  - ◆ Use Cases
  - ◆ User Stories

# System Requirements

- Formal representation of the requirements of a system, part of the system engineering body of knowledge.
- Typically leverages specific terms:
  - shall**: Requirement is contractually binding, meaning it must be implemented and verified
  - will**: Statement of fact. Not subject to verification.
  - should**: Goals, non-mandatory vision. Typically a goal which must be addressed by the design team but is not formally verified.
- Note: Do not use **must** as no one has defined how must is different from shall. Also, shall has held up in court, must has not.
- The above is based on ISO TR 10176

# System Requirements - Examples

- ◆ **Shall Requirement**

- ◆ The system shall process updates from the data source within 6 seconds from initial receipt.

- ◆ **Will Requirement (describing another system)**

- ◆ The data source will provide updates every 10 seconds in Json format.

- ◆ **Should Requirement (design requirement)**

- ◆ The system should not prevent users from carrying out other activities while it is processing updates from the data source.

- ◆ **Note:**

- ◆ *Will* and *should* requirements are typically to provide context
- ◆ Notes are common, to provide further context

# User Based Requirements

## ◆ Use Cases

- ◆ In software and systems engineering, a **use case** is a list of actions or event steps typically defining the interactions between a role (known in the Unified Modeling Language as an actor) and a system to achieve a goal. The actor can be a human or other external system.

## ◆ User Stories

- ◆ A **user story** is a tool used in **Agile** software development to capture a description of a software feature from an end-**user** perspective. The **user story** describes the type of **user**, what they want and why. A **user story** helps to create a simplified description of a requirement.



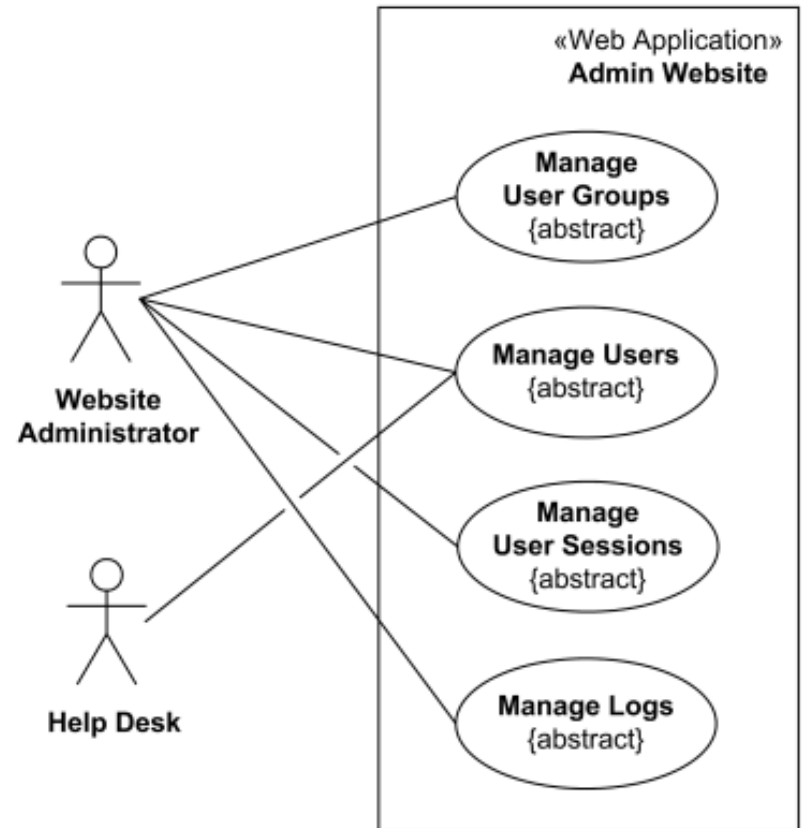
# User Based Requirements - Examples

## Use Case Example (UML)

### Website Administration

**Purpose:** Website management or administration URL use case diagrams example

**Summary:** Website administrator actor could manage user groups, users, user sessions, and logs. Help Desk staff uses a subset of functions available to the Website Administrator.



# User Based Requirements - Examples

## Use Case Example (Table Based)

Elaboration of each scenario from the UML Use Case diagram

Provides a shared understanding between:

- 💧 Customer
- 💧 Developer
- 💧 Tester

Can be used to “sell-off” development work and for verification purposes

Use Case Element	Description
Use Case Number	ID to represent your use case
Application	What system or application does this pertain to
Use Case Name	The name of your use case, keep it short and sweet
Use Case Description	Elaborate more on the name, in paragraph form.
Primary Actor	Who is the main actor that this use case represents
Precondition	What preconditions must be met before this use case can start
Trigger	What event triggers this use case
Basic Flow	The basic flow should be the events of the use case when everything is perfect; there are no errors, no exceptions. This is the "happy day scenario". The exceptions will be handled in the "Alternate Flows" section.
Alternate Flows	The most significant alternatives and exceptions

# User Based Requirements - Examples

## User Story Format

*As a <type of user>, I want <some goal> so that <some reason>.*

# User Based Requirements - Examples

As IT Security, I want Sharepoint to have a whitelist of attachment file types so that we can prevent the upload of potentially malicious files.

- ◆ IT Security – Who
- ◆ a whitelist of attachment file types – What (i.e., the feature)
- ◆ prevent the upload of potentially malicious files – Why

# User Based Requirements - Examples

As IT Security, I want Sharepoint to have a whitelist of attachment file types so that we can prevent the upload of potentially malicious files.

- ◆ Acceptance Criteria (Basic):

- ◆ The following file types are whitelisted in the Sharepoint upload tool:  
PDF, Word, Excel, PNG
- ◆ All other file types are blacklisted, specifically: EXE, BAT, CMD

- ◆ Acceptance Criteria (Given-When-Then):

- ◆ Given a file of type PDF, Word, Excel or PNG, When a Sharepoint user uploads the file Then Sharepoint should allow the file upload
- ◆ Given a file of type EXE, BAT or CMD, When a Sharepoint user uploads the file Then Sharepoint should reject the file upload

# User Based Requirements - Examples

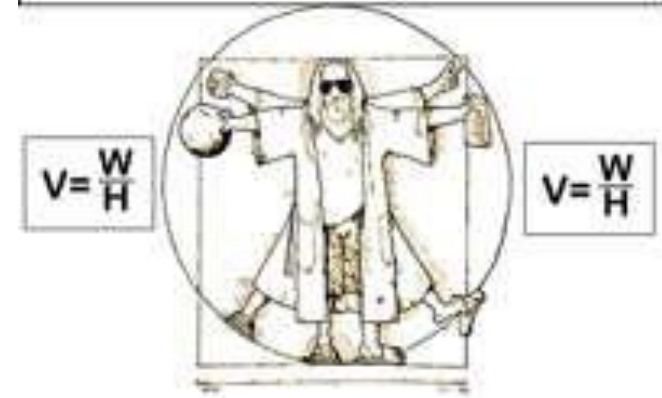
**User Stories** – Help to provide the ‘why’ to those implementing and testing the requirement.

**Dude's Law:**  $V = W / H$ ,  
where V is value,  
W is why (intent) and  
H is how (mechanics).

If (H)ow increase and (W)hy is constant, then (V)alue is reduced.

If your W is constant (you know what you expect) and you reduce H (less process) then the V increases.

**Dude's Law: Value = Why / How**



As you drive the (H)ow towards zero, which you could call leaning out your processes, (V)alue increases even if Why is constant.

<https://devjam.com/2010/08/05/dudes-law-gordon-pask-shoveler/>

# SMART Requirements

- ◆ **Specific**

- ◆ A good requirement is specific and not generic. It should not be open to mis-interpretation when read by others.

- ◆ **Measureable**

- ◆ This answers whether you will be able to verify the completion of the project. You should avoid signing up for any requirement that cannot be verified as complete.

- ◆ **Attainable (Achievable, Actionable, Appropriate)**

- ◆ This is intended to ensure that the requirement is physically able to be achieved given existing circumstances.

- ◆ **Realistic**

- ◆ Answers whether the requirement is realistic to deliver when considering other constraints of the project and requirements.

- ◆ **Time Bound (Timely, Traceable)**

- ◆ Where appropriate each requirement should be time-bound or specify by *when* or *how fast* a requirement needs to be completed or executed.



# SMART Requirements - Examples

**Specific** – Are the following requirements strong or weak?  
Why?

*The report shall display all the monthly data from the marketing department.*

*The report shall contain the following columns: Total Sales for Month, Average Retail Price, Total Units Sold, Remaining Inventory, Total Cost of Goods Sold.*

# SMART Requirements - Examples

**Measureable** – Are the following requirements strong or weak?  
Why?

*The system shall have an optimal response time for the end-user.*

*The system shall have user response times on user click-events that are 5-seconds or less during business hours of 9AM-5PM, Mountain Time, Monday-Friday.*

# SMART Requirements - Examples

**Attainable** – Are the following requirements strong or weak?  
Why?

*The monthly marketing sales report and the monthly financial statements shall both be delivered on the 1st business day of the month.*

*The monthly marketing sales report shall be delivered on the 1st business day of the month. The monthly financial statements will be delivered on the 3rd business day of the month.*

# SMART Requirements - Examples

**Realistic** – Are the following requirements strong or weak?  
Why?

*The marketing sales report shall be delivered into production on or before November 1, 2008.*

# SMART Requirements - Examples

Time Bound – Are the following requirements strong or weak?  
Why?

*The report shall be available soon after month-end close.*

*The report shall be available by noon on the first business day after the successful completion of the month-end accounting reports (insert identifying report id).*

# Requirements - General

Why do we need requirements:

- ◆ To define the project/product scope (the what)
- ◆ To have a shared understanding of the application/system (the why)
- ◆ To enable traceability to design, implementation and/or test
  - ◆ Note: Test is key here – verifying that you have built what you said you would.

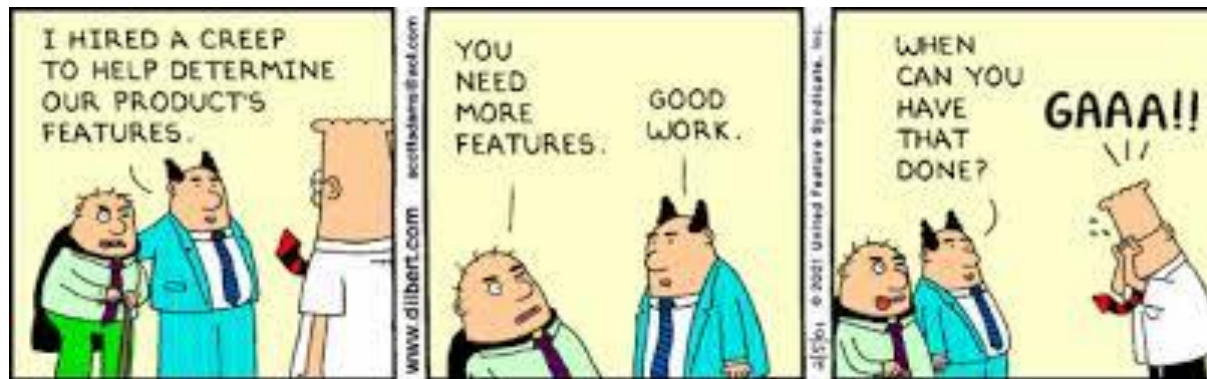
Other best practices:

- ◆ Numbering your requirements so you can easily reference them
  - ◆ Example: APP-001, UCASE-101
- ◆ Using a requirements management tool, such as DoorsNG

# Requirements Management

## Scope Creep

- ◆ We've looked at this topic a number of times already
- ◆ It's a consistent, important theme in what we do





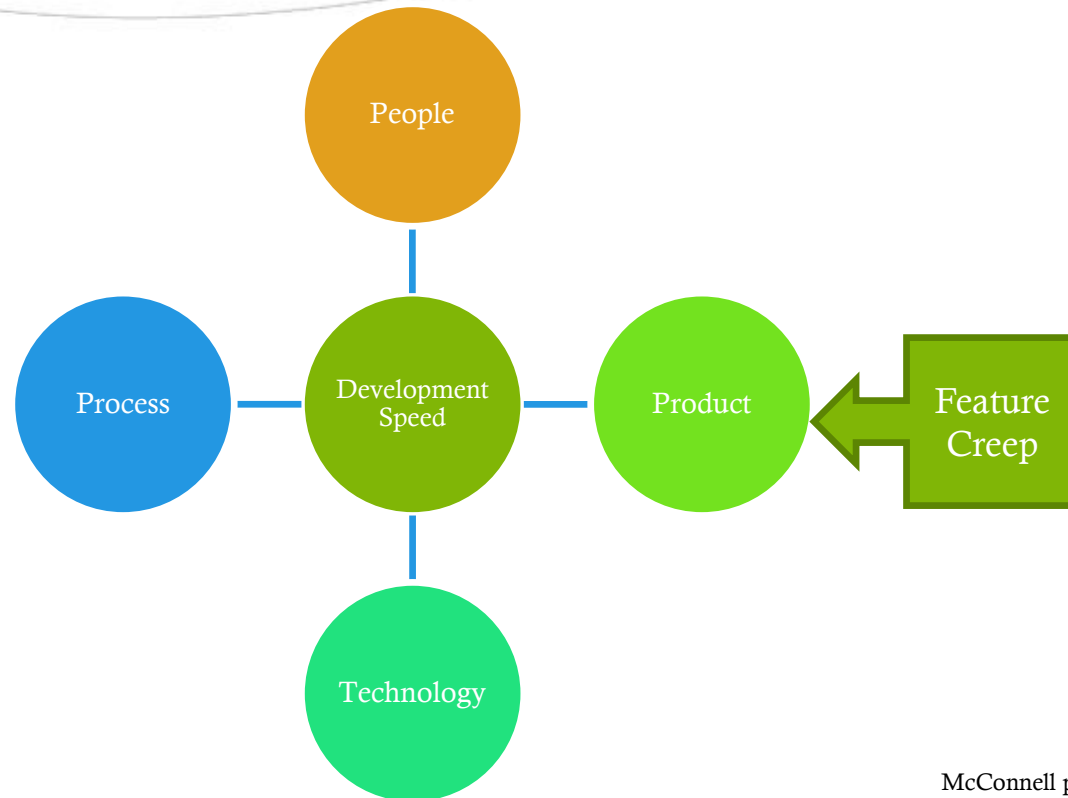
# Feature Set Control

- ◆ McConnell cites several studies that show how feature creep has been fatal to budgets, schedules and whole projects
- ◆ Project Management Institute (pmi.org) says “poor requirements management is a major cause of project failure, second only to changing organization priorities”
  - ◆ *Pulse of the Profession, 2013*
- ◆ Such a big deal that whole new professions are growing up around this area

# Four Pillars of Rapid Development



# Four Dimensions of Development Speed



McConnell p. 11

# McConnell's View

## Customer's Importance to Rapid Development

- Improved efficiency
  - Tasks take less time
- Process Dimension of Development Speed
  - Less rework
  - Reduced risk
  - Lack of friction
    - Also a “Classic Mistake”

## Customer-Oriented Practices

- Planning
- Requirements Analysis
- Design
- Construction
- Plus, Manage Customer Expectations

# Customer Oriented Practices

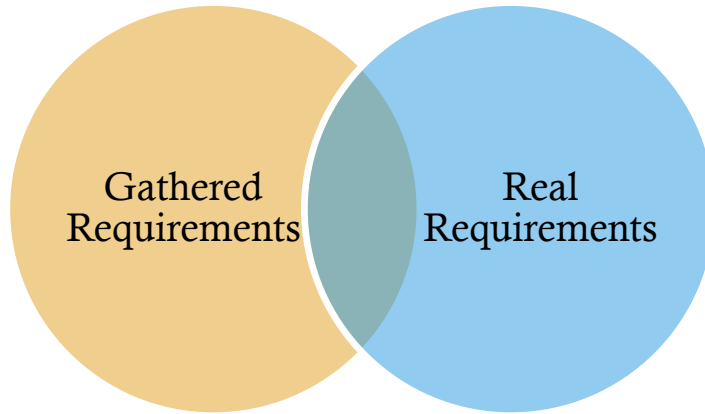
## Planning

- Select an appropriate lifecycle model that provides signs of progress and allows changed requirements
- Identify the “real” customer
- Efficient interaction - single point of contact
- Think “win-win”
- Manage risks

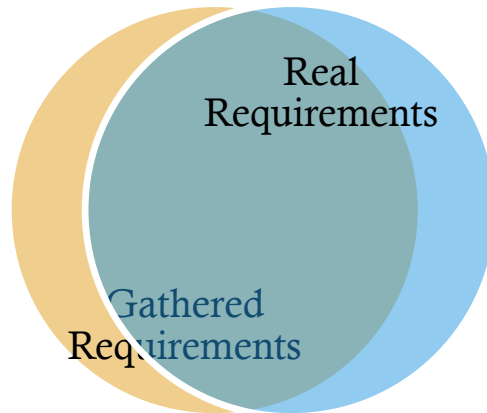
## Requirements Analysis

- Gather the “real” requirements (ask “why”)
- Involve customers closely without them writing specs
- Use requirements-elicitation practices – e.g. JAD, prototypes, etc.
- Use tools such as focus groups and surveys
- Anthropology

# Customer Oriented Practices



"Typical"  
requirements  
gathering processes



Customer-oriented  
requirements  
gathering processes

# Lifecycle Model Considerations

Different projects have different development needs – even if they all need to be developed as soon as possible

## Requirements Considerations

- How well do my customer and I understand the requirements at the beginning of the project?

## Architectural Considerations

- How well do I understand the system architecture?
- Am I likely to need to make major architectural changes midway through the project?

## Reliability Considerations

- How much reliability do I need?

## Future Version Considerations

- How much do I need to plan ahead and design ahead during this project for future versions?



# Requirements Management

When is a good time to manage requirements?



# Feature Set Control in Project Stages

## Chapter 14

### Early Project Control

- Define a feature-set consistent with schedule/budget constraints
- Minimal spec
- Requirements scrubbing
- Versioned development

### Mid-Project Control

- Controlling creeping requirements
- Change control

### Late-Project Control

- Trimming feature to meet schedule and budget
- Feature cuts

# Early Project Feature Set Control

## Minimal Specifications

### Problems with Traditional Specs

- Wasted effort
- Obsolescence
- Lack of efficacy
- Overly constrained design


### Benefits of a Minimal Spec

- Improved morale and motivation
- Opportunistic efficiency
- Less wasted effort
- Shorter requirements phase

### Risks of a Minimal Spec

- Omission of key requirements
- Unclear or impossible goals
- Gold-plating
- Unsupported parallel activities
- Attachment to specific features
- Sub for good requirements management

# Techniques for Creating Minimal Specs

- Short paper spec
  - Point-of-departure
  - User manual as spec
  - UI prototype
  - Paper storyboard
  - Vision statement
  - Product theme
- 
- Customer-Oriented Development
    - UI prototype
    - Paper storyboard
    - [Design Thinking](#)

# Minimal Specification - Today

- 💧 What is a [Minimum Viable Product](#) (MVP)?
- 💧 What do you think of this approach?
- 💧 What are the limitations, i.e. when won't it work, what needs to be in place?
- 💧 What do you do with the feedback you get on your MVP?

# Early Project Feature Set Control

## Requirements Scrubbing

### McConnell

- Eliminate requirements that are not absolutely necessary
- Simplify requirements
- Substitute cheaper options, where available

### BABOK

Business Analysis Book Of Knowledge

- Identify stakeholders
- Solution-scope management
- Baseline
- Sign-off
- Traceability

# Early Project Feature Set Control

## Versioned Development

- Plan requirements for a complete project
- Implement in phases/versions
- “Put in any hooks you’ll need...”
  - Evolutionary/Staged Delivery
- Lifecycle model selection
  - also links to Mid-Project Control
- Some other considerations
  - Iterative development
    - Lifecycle model selection
    - Minimum Viable Product (MVP)
  - Product roadmap
    - What is it?
    - Problematic for vendors
    - Why is that?



# Mid Project Feature Set Control

McConnell

## Feature Creep Control

- Sources of change
  - Market pressure
  - Customer learning
  - Developer gold-plating

## Change Control

- Allow if it enables best possible product in time available
- Allow stakeholders to assess impact and decide (i.e., Change Control Board)
- Notify other affected stakeholders of the impact
- Provide an audit trail of changes

# Mid Project Feature Set Control

## Change Control Board

In software development, a **Change Control Board** (CCB) or **Software Change Control Board** (SCCB) is a **committee** that makes decisions regarding whether or not proposed **changes** to a software project should be implemented.

*(Source: Wikipedia)*

### Features:

- Internal or External
- Meets on a regular basis (daily, weekly, monthly)
- Includes key stakeholders
  - Project/Technical Management
  - Subject Matter Experts
  - Team Leads
- Decides to implement, request more information, reject or defer a change request

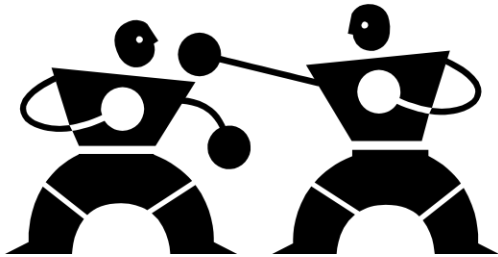
# Mid Project Feature Set Control

BABOK

- ◆ Approval of requirements may be sought at the end of a project phase or at a number of intermediate points in the business analysis process
- ◆ Any changes to requirements after baselining, if changes are permitted, involves use of a change control process and subsequent approval
  - ◆ As requirements are refined or changed as the result of new information, changes will be tracked as well
- ◆ Change-driven approaches typically do not use a formal change control process, as requirements are prioritized and selected for implementation at the beginning of each iteration and no changes to the requirements occur during an iteration

# Late Project Feature Set Control

## Feature Cuts



- The process won't be pretty
- BABOK recommends requirements prioritization
- What can be shifted to the next iteration/version?

How does Agile  
impact this process?

How does Waterfall  
impact this process?

# Summary

## Feature Set Control

- ◆ Feature Set control is critical to project success
  - ◆ Good project management practices
  - ◆ Good requirements management processes
  - ◆ Appropriate lifecycle model selection
- ◆ May ultimately need to cut scope
  - ◆ which could be just the next sprint/iteration
  - ◆ Could also be a big deal



# Project Recovery

McConnell Chapter 16



# Project Recovery

When they  
fail, what can  
be done about  
it?



# Elements of Successful Projects

## What are the Elements of Successful Projects?

- The Customer/Client accepts the solution (scope / functionality)
- Meet the Triple Constraints
  - The team completes the project within the timescale (**time**)
  - The team completes the project within the budget (**cost**)
  - The team completes the project within acceptable quality levels (**quality**)
- Deliverables are successfully transitioned into operations
- Users are satisfied
- Project is supportable in live operations
- Project made the expected business objectives
- Record made of lessons learned so that future projects will benefit from the success of this project (post mortem)

# Recognizing Projects in Trouble

## What are the Symptoms of Troubled Projects?

- Behind schedule
- Over cost
- Not producing the intended scope
  - Disagreements over scope
  - Unapproved/uncontrolled changes to requirements
- Quality problems (failed acceptance tests)
- Stalling at the 90% complete mark
- Resource issues
  - Too few; wrong kind; poor morale; turnover
- Too many things (risks) happening that throw the project off track
- Continually waiting for dependencies (internal/external)
- Problems with subcontractors
- Client satisfaction issues
- Decisions not being made
- Political infighting

# Root Causes of Failure in Projects

## What are Some of the Main Causes of Project Failure?

- Unclear or incomplete scope definition
- Unclear, unconvincing or missing business case (the why)
- Poor project planning
- Poor communication
- Lack of end user involvement
- Changing requirements
- Unclear project priorities: time, cost or scope
- Lack of executive support
- Questionable competence of resources
- Lack of resources
- Unrealistic expectations
- Poor project management

# What does McConnell Say?

- General Recovery Options
  - Cut scope
  - Increase productivity
  - Slip the schedule
  - A combo of the first three
- First Steps
  - Assess your situation
  - Think win-win
  - Prepare yourself to fix the project
  - Ask the team for input
  - Be realistic
- Dimensions of Development Speed
- Then, Timing

# Recovery: Dimensions of Dev. Speed

## People

- Restore group morale
- Clean up major personnel and leadership issues
- Add people carefully
- Focus people's time
- Allow differences
- Ensure pace

## Process (1)

- Identify/fix classic mistakes
- Fix the clearly broken parts of your development process
- Create detailed mini-milestones
- Link schedule to milestone completion
- Track progress meticulously

## Process (2)

- Record reasons for missed milestones
- Recalibrate after a short time
- Only commit to a meaningful schedule
- Manage risks painstakingly

## Product

- Stabilize requirements
- Trim the feature set
- Assess your political position
- Take out the garbage
- Reduce the number of defects – and keep them reduced

Why isn't Technology a factor?

# Other Best Practices

- ◆ Generally be open and honest with your customer/client
  - ◆ Leverage your good relationship, if possible (i.e., customer oriented practices)
- ◆ Provide your customer/client with appropriate visibility
  - ◆ If you can, make them involved in schedule/scope choices
- ◆ Provide your team with appropriate visibility
- ◆ Timing – Don't be too early or wait too long to implement your recovery plan
  - ◆ Too early – people won't believe it's needed
  - ◆ Too late – may have been failed mini-corrections, could lose your credibility for a bigger recovery effort

# Case Studies (if time)

- ◆ Update: FastIT and BigCorp
- ◆ Chapter 16 case studies
  - ◆ 16 – 1 (page 372), unsuccessful recovery
  - ◆ 16 – 2 (page 385), successful recovery



# Form Assignment 3 Groups

- ◆ Groups of 2 or 3
- ◆ People who were grouped for Assignment 2 may **not** be in the same group for Assignment 2
- ◆ Self-enroll in D2L

# Next week

- ◆ 1 hour: Teamwork and Team Structure
- ◆ 2 hours: Assignment 3
  - ◆ Must submit to D2L at the end of class

# Comp8081

end of Week 12

Donna Turner

