Software Project Management

by

Philippe Kruchten

January 2014

Philippe Kruchten, Ph.D., P.Eng., CSDP

Professor of Software Engineering
NSERC Chair in Design Engineering
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC Canada
pbk@ece.ubc.ca

Founder and president
Kruchten Engineering Services Ltd
Vancouver, BC Canada
philippe@kruchten.com

Outline

1. Introduction
2. A conceptual model of software development
3. Contexts of software development
4. Process and management frameworks & standards
5. Managing risks
6. Managing time: macro-level
7. Managing time: micro-level
8. Managing quality
9. Managing objectives and scope
10. Managing complexity
11. Managing changes
12. Managing software assets
13. Managing software products
14. Managing people: individual level
15. Managing people: team level
16. Managing external stakeholders
17. Managing the software process
18. Software development governance
19. Very large projects, programs, and project portfolios
20. Conclusion

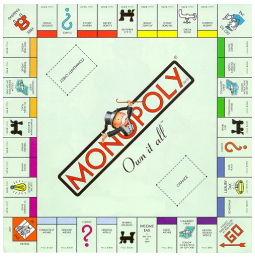
01: Introduction

Software Project Management
Philippe Kruchten

Module outline

- What is a project
- What is Project Management
- What is a software project
- What is software project management
- A brief history of project management
- The special case of *software* projects

- Software
- Project
- Management



What is a project?

- An endeavor with a defined goal, a start and an end, and some constraints.

What is a Software Project

- Software-intensive system
- Software-reliant system

- Software represents the dominant cost
- Focus here is on the *software* part, though many aspects will apply to the whole system

Software Project

- Objectives
  - Clearly defined set of goals; non conflicting…?
- Start and end points
  - not a continuous activity, like operations or support
- Uniqueness
  - One time thing (product); not a repetitive activity
  - Not production, operations
- Constraints
  - Cost, schedule, quality

Source: Futrell

What is Project Management

- Project management is the application of knowledge, skills, tools, and techniques to project activities in order to meet or exceed stakeholder needs and expectations from a project
  (Project Management Institute)

 7

Defining Software Project Management

- Software Project Management is the art of balancing competing objectives, managing risks, and overcoming constraints to successfully deliver a product which meets the needs of both customers (the payers of bills) and the users. (RUP)

 8

"Clear end point"…?

- Multiple releases
- Continuous deployment

- Resources (financial, staff)
- Difference between:
  - Internal IT support
  - Software product (or software in a system)
  - "Bespoke" software (contract with a customer)

 9

Origins of project management

- Pyramids, Roman bridges, and all that stuff.

- Henry Gantt, 1917: barcharts
- Flow line scheduling: 1930
- Line of Balance: 1940 Goodyear, US Navy
- Milestone charts, 1940
- Critical Path Method (CPM), DuPont de Nemours 1950's
- PERT, US Navy 1958

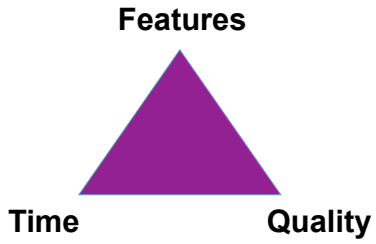
 10

Origins

- Precedence method
- Iron triangle (time, cost, output):
  - *Martin Barnes, UK, 1969*
- …
- …
- Toyota Production System
- Agile Manifesto

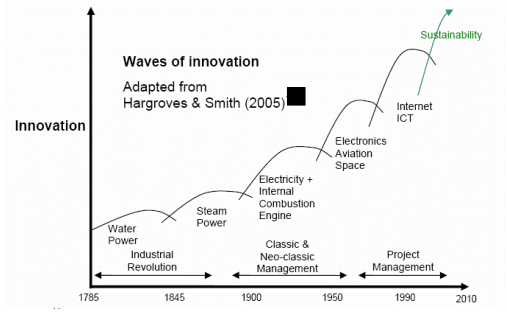
 11

A Fundamental Conundrum



 12

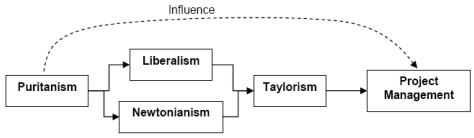
Waves of Innovations



 Source: Weaver 2007 13

Pre-classical developments

- Puritanism: the protestant work ethic
- Liberalism: capitalism, division of labour
- Newtonianism: scientific enquiry



Source: Weaver 2007

 14

Classical development

- Taylorism: scientific management
  - Frederick Taylor (1911)
  - Work breakdown structure and scheduling
- Bureaucracy, administrative management
- System theory, contingency view (1940's)
  - Multiple input, multiple outputs, complex interrelated processes
- Operations research
  - *Military tactics, supply, etc…*

 15

And then came the computer

- Mainframes

- PC (commodore, Atari, Apple, IBM)
- Microplanner
- Primavera (1983)
- Microsoft Project (1987)

- *And now everybody can schedule… yeah!*

 16

And then….

- Earned Value Management
- Program management
- Portfolio management

 17

On the bleeding edge…

- Lean production
  - And therefore "lean management"
- Critical Chain and Theory of Constraints
- Chaos theory
  - Complex adaptive system

 18

The profession of project management

- Profession:
  - Body of knowledge
  - Code of ethics
  - Education
  - Certification, etc…
  - "Belonging", -> culture?
- Project Management Institute (PMI)
  - And a handful of others: IPMA, APM (UK)..

 19

Project Management Body of Knowledge (PMBOK)

- PMBOK: Developed by the Project Management Institute (PMI) (USA)
- Initial publication in 1987; republished in 1996
- Adopted by IEEE as IEEE 1490-1998
  - *Wow, that was a bad move for software (I think)*
- NOT software specific
- A certification: Project Management Professional (PMP)
  - And now Agile project certification
- Coming in 2013: Software extension to PMBOK (SWX)

Source: PMI & IEEE

 20

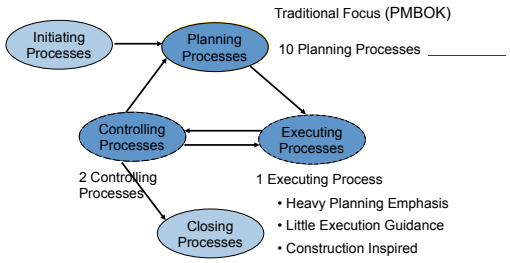
Project Management Body of Knowledge

Project Integration Management	Project Scope Management	Project Time Management
Project Plan Development Project Plan Execution Integrated Change Control	Initiation Scope Planning Scope Definition Scope Verification Scope Change Control	Activity Definition Activity Sequencing Activity Duration Estimation Schedule Development Schedule Control
Project Cost Management	**Project Quality Management**	**Project Human Resource Management**
Resource Planning Cost Estimating Cost Budgeting	Quality Planning Quality Assurance Quality Control	Organizational Planning Staff Acquisition Team Development
Project Communications Management	**Project Risk Management**	**Project Procurement Management**
Communications Panning Information Distribution Performance Reporting Administrative Closure	Risk Management Planning Risk Identification Qualitative Risk Analysis Quantitative Risk Analysis Risk Response Planning Risk Monitoring & Control	Procurement Planning Solicitation Planning Solicitation Source Selection Contract Administration Contract Closeout

Source: E. Lopes Cardozo, Empulsys

 21

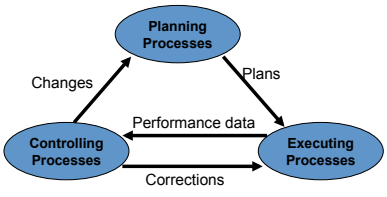
Looking into the PMBOK Philosophy



Sources: L. Koskela & Mike Griffiths

 22

Basic Loops in the PMBOK



Sources: L. Koskela & Mike Griffiths

 23

Basic Elements & Theories & Practices

- Planning processes
  - Well-understood and pushed to the extreme in some industries; WBS; Activities/tasks; Risks
  - Estimation (when? how much?)
- Executing processes
  - Task assignment
- Controlling processes
  - Thermostat model, cybernetic model
  - What to measure; assessing progress?
  - Earned value

 24

- This is great stuff, right?
- Let us apply all this to software development, shall we?

Software Project is a Different Beast

- Not a construction or assembly process
- Not an administrative process
- Exploratory: trial and error
- No (or few) pre-defined work breakdown structures (WBS)
- The "nice party" metaphor

Why is Software Different?

1. No fundamental laws of software
   - Cannot validate software on blueprints
   - Need to build and test
2. Extreme modifiability
   - End-user & customer want to exploit this
   - Constant changes
3. No manufacturing costs, no border
   - All costs are in design, no economy of scale
4. Technology churn

Summary

- This course is not on project management, but on software project management
- Software is different than other disciplines:
  - Knowledge intensive
  - Creative
  - Not repetitive
  - But software itself is pretty flexible

02: A Conceptual Model of Software development

Software Project Management
Philippe Kruchten

The Frog and the Octopus

A Conceptual Model of Software Development

fable |ˈfābəl| *(noun)*

a short story, typically with animals as characters, conveying a moral.

- a story, typically a supernatural one incorporating elements of myth and legend.
- myth and legend *: the unnatural monsters of fable.*
- a false statement or belief.

Once upon a time, a frog and an octopus,
Met on a software project, that was deep in the bush.
The frog said, "you know, all these projects are the same;
Over time we fill with our work the gap that we find
Between the burgeoning product, and our dreamed intent."

"Oh, no" objected the octopus, "they cannot be the same;
They come in all forms or shapes and sizes and colours,
And we cannot use the same tools and techniques
Like in the cobbler shop, one size does not fit all."

Outline

- Motivation
- The model
- The frog part
  - Four key entities:
    - Intent, Product, People, Work
  - Three attributes
    - Uncertainty / Risk, Time, Quality
  - External influences
  - Two more attributes
    - Value and Cost
- Using the model
- Further ideas to explore

"The purpose of science is not to analyze or describe but to make useful models of the world. A model is useful if it allows us to get use out of it."

Edward de Bono

Motivation

- Software project management can turn into a long series of recipes
- Vast spectrum of possible projects
- Two models:
  - What is common across all software projects (frog)
  - What are the factors of variability (octopus)

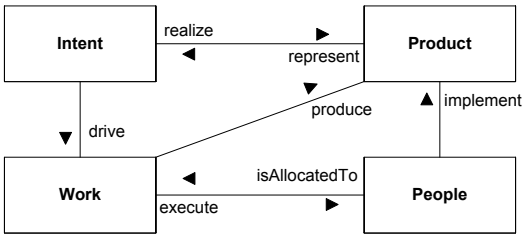
A Conceptual Model of Software Development

4 key concepts, 3 key attributes

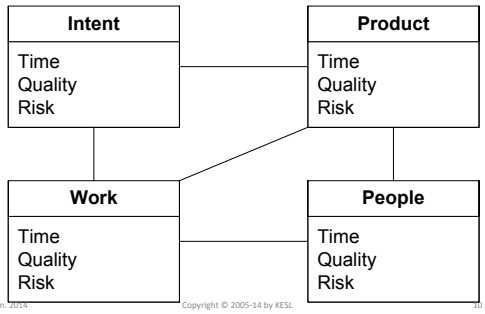
- Intent
- Product
- Work
- People

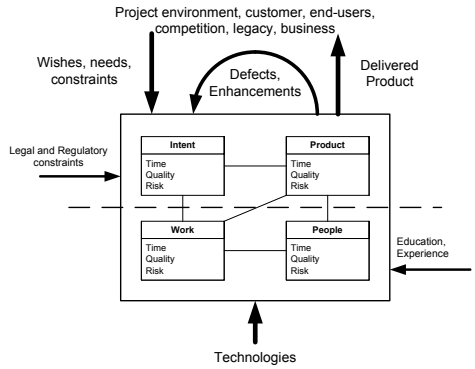
- Time
- Quality
- Risk

Intent, Work, People, Product

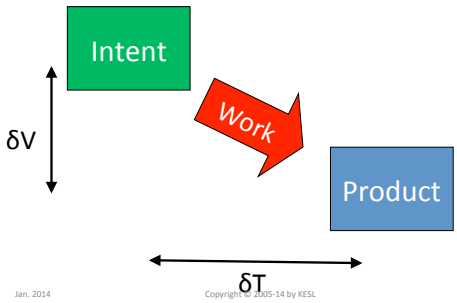


Adding Time, Quality & Risk

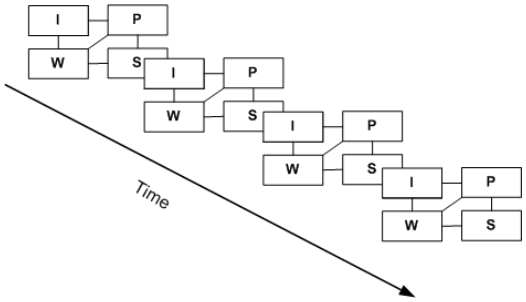




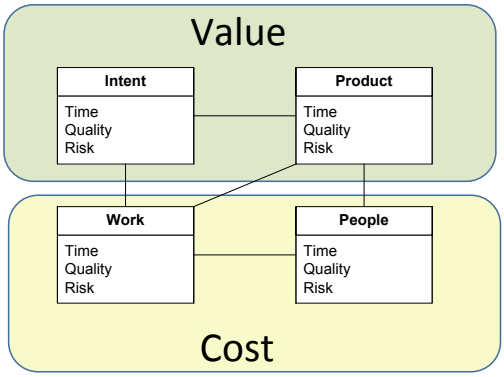
SW Dev. Project: Tension between Intent and Product



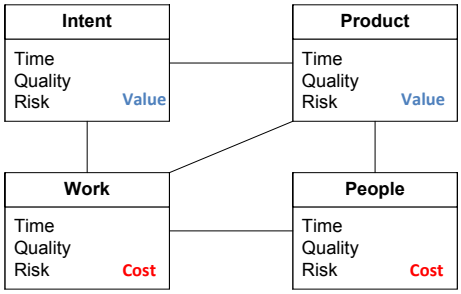
Iterations



Value



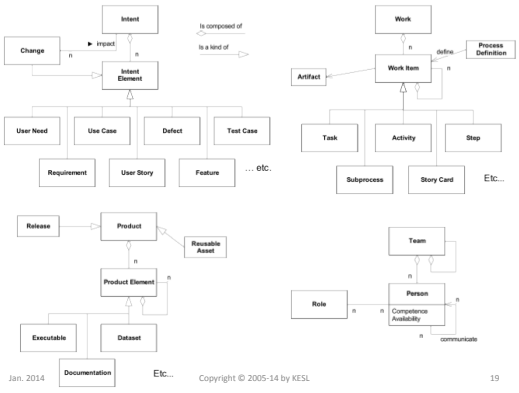
Adding Value and Cost

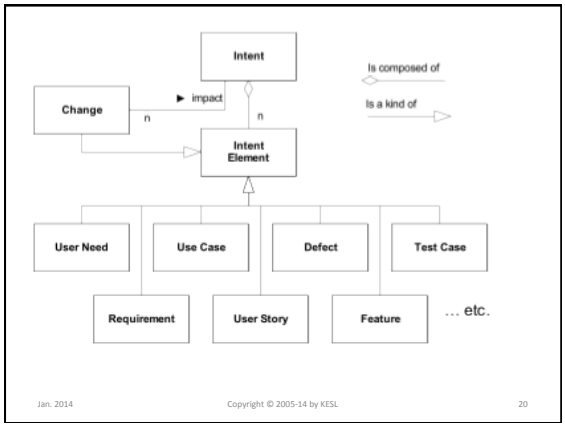


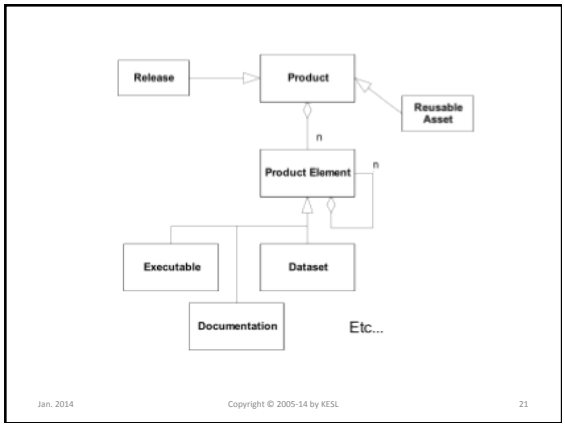
A **project** is all the **work** that **people** have to accomplish over **time** to realize in a **product** some specific **intent**, at some level of **quality**, delivering **value** to the business at a given **cost**, while resolving many **uncertainties and risk.**

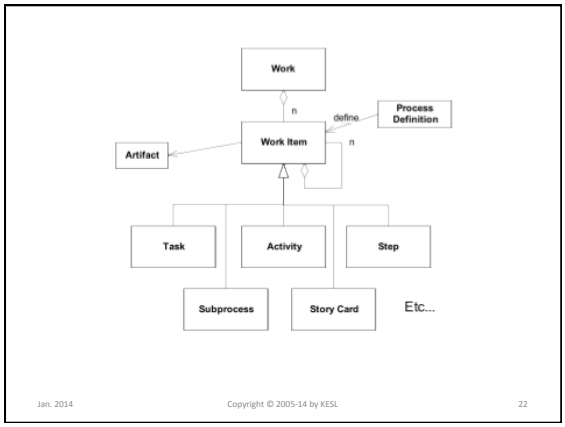
Exercise: which concept is at play?

- If Christine were to resign today, we would be in serious trouble
- The customer want us to implement another use case, regarding costing with discount.
- I know this was not planned, but we will need to move the server farm out of the basement.
- Is our documentation compliant to IEEE 1540?
- When (in which release) will we have a complete implementation of the data entry GUI?
- Should we hire a senior tester, or a beginner that we will train?
- We have over 27 “show stopper” defects in Bugzilla.
- We have to schedule a retrospective for each major release.




Intent
Is composed of
Is a kind of
Change
impact
n
n
Intent Element
User Need
Use Case
Defect
Test Case
Requirement
User Story
Feature
… etc.
Jan. 2014
Copyright © 2005-14 by KESL
20


Release
Product
Reusable Asset
n
Product Element
n
Executable
Dataset
Documentation
Etc...
Jan. 2014
Copyright © 2005-14 by KESL
21


Work
n
define
Process Definition
Artifact
Work Item
n
Task
Activity
Step
Subprocess
Story Card
Etc...
Jan. 2014
Copyright © 2005-14 by KESL
22

03: The context of software development

Software Project Management
Philippe Kruchten

Outline

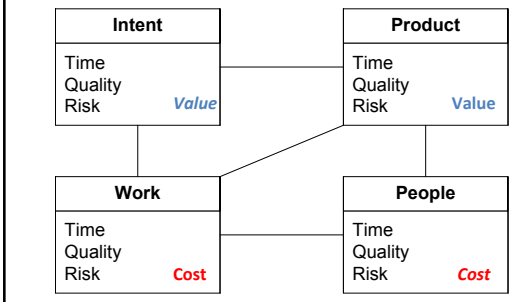
- Two models:
  - What is common
  - What is different
- One size does not fit all
- The vast spectrum of software development projects "cases"
- Sources of Variability
- The Octopus

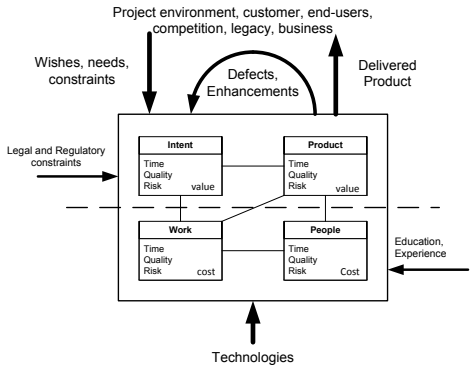
Conceptual model

- Unifying model
- Commonality
- Variability

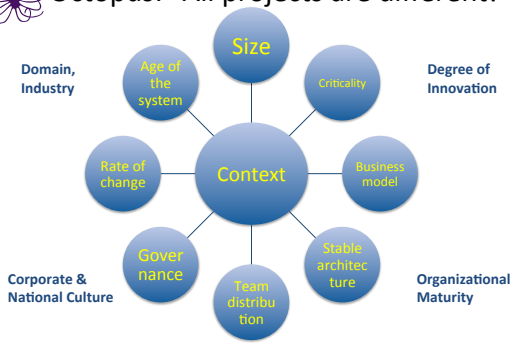
- Usage:
  - Set up the syllabus
  - Analyze and critique practices
  - Compare approaches, process models

Frog: "All projects are the same"





Octopus: "All projects are different!"

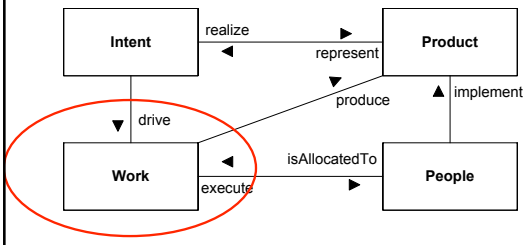


Defining Process

- Process: set of activities intended to achieve a goal
- Process to run a project
  - = software development process
  - = software engineering process
  - = software process
- Software project management is a subset of that software development process

 9

Process is defining Work



Jan. 2014 10

Importance of process

- Processes represent the collective knowledge on how to run a project
- A Project process has 2 main aspects:
  - Engineering : design test, code, technology
  - Project management
- Good project managers must understand the process
  - Both the engineering process
  - and the management process

Jan. 2014 11

Type of Software Projects

- Commercial, speculative vs. contract work
- Many instances vs. single few instances
- Internal vs. external (acquisition)
- Greenfield vs. evolution or legacy transformation
- Single project vs. program or portfolio
- Size

- Process: one size does *not* fit all…!

Jan. 2014 12

Project Context

- Environmental Conditions (organization)

Drive/constrain

- Context Attributes (software project)

Selection and adaptation

- Good Practices (actual process)

Jan. 2014 13

Environmental conditions

- Business domain
  - E-commerce
  - Manufacturing
  - Automotive
  - Aerospace
- Number of instances
  - One, A dozen, Millions, SaaS,…
- Maturity of organization
  - Small start up
  - Mid size software Dev. Co.
  - Large system integrator

+… collective experience

Jan. 2014 14

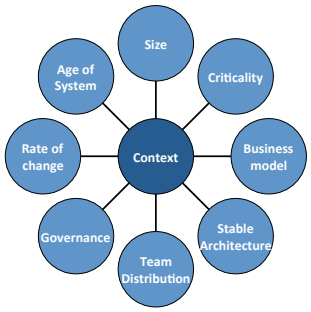
Environmental conditions (cont.)

- Level of innovation
  - New product, never been done… or
  - Old classic, just better, faster, larger, …
- Culture
  - Communication
  - Trust
  - Shared mental models
  - Education (?)

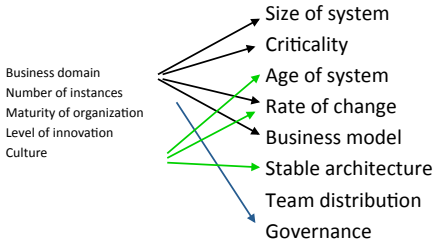
In general, environmental conditions are proper to the organization, and common to several projects

Context attributes

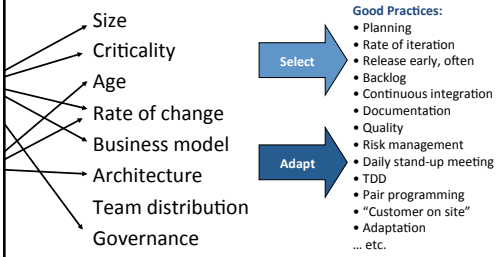
1. Size
2. Criticality
3. Age of system
4. Rate of change
5. Business model
6. Stable architecture
7. Team distribution
8. Governance



Environmental drivers ➡ Context attributes



Context attributes ➡ Practices/process



1. Size of system

- SLOC, FP
- Impacts team size, duration…

- *Driven by:* Business domain
- *Related to:* Legacy, geographic distribution, governance
- *Affects:* Iteration rate, planning, communication modalities, documentation, risk management, "customer on site", etc.

2. Criticality

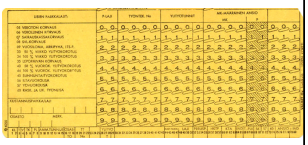
Belleville-sur-Loire, France

- "Software that kills"
  + Massive losses, damage to environment
- Demonstrably correct
- Formal methods
- Extensive testing
- Audited by external agencies

- *Driven by:* Business domain
- *Related to:* Rate of change
- *Affects:* Documentation, testing, inspection

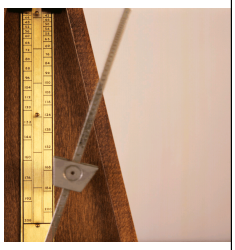
3. Age of system

Legacy evolution
Brownfield *vs.* green field development, maintenance



- *Related to:* size
- *Affects:* Testing, (lack of) documentation, architecture

4. Rate of Change



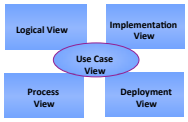
- Adaptation versus anticipation
- External & internal changes
  - Customer, competitor, technology, legislators, inside organization, turnover, team evolution, maturation, …
- *Driven by:* business domain
- *Related to:* size
- *Affects:* Iteration length, planning, adaptation,…

5. Business Model

- Commercial market
- Bespoke software
- In-house development
- Open source development
- … etc.

- *Driven by:* Business domain
- *Related to:* Governance
- *Affects:* Documentation, number of instances, "customer on site", communication, risk management, geographic distribution, *rate of iteration*

6. Architecture stability



- How much of a stable system and software architecture is in place when the project starts?

- *Driven by*: Level of innovation
- *Related to*: Size of system, age of system
- *Affects:* Rate of iteration, risk management, testing

7. Geographic Distribution of the Team

- Seems to make everything a bit harder and more susceptible to fail

- *Driven by:* Maturity of organization, culture
- *Related to:* Size, business model
- *Affects:* Communication modalities, documentation, DSM, governance, …

8. Governance

- Structural: chains of authority, responsibility and communication to empower the various actors
- Dynamic: measures, control, mechanisms, policies to enable all actors to carry out their respective responsibilities

Clay Williams, IBM, 2008

Is this "big process" coming by the back door?

The agile "sweet spot"

System Size	• 0 ..**12** … 300
Criticality	• **Simple, $ losses**, … deaths
System Age	• Exploratory, **greenfield**, legacy maintenance
Rate of change	• Low, **medium, high**
Business model	• **In house**, Open Source, ….
Stable architecture	• **Stable**, changed, new
Team distribution	• **Collocated**, …, …, offshore outsource
Governance	• **Simple rules**, …, SOX, …

Examples: analyzing context

Business context	Commercial, speculative	Contractual	Research, Open-source
# of instance	1	A few	many
Specifier Developer relationship	Internal	Internal offshore	External, outsourced
Reuse	Greenfield	Brownfield	Legacy
Dev. context	Single	Program	Portfolio
Size of dev. team	<12	12 - 20	>20

Business context	Commercial, speculative	Contractual	Research, Open-source
# of instance	1	A few	**many**
Specifier Developer relationship	**Internal**	Internal offshore	External, outsourced
Reuse	Greenfield	**Brownfield**	Legacy
Dev. context	Single	Program	Portfolio
Size of dev. team	**<12**	12 - 20	>20

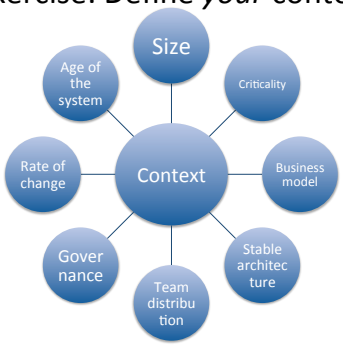
Business context	Commercial, speculative	Contractual	Research, Open-source
# of instance	**1**	A few	many
Specifier Developer relationship	**Internal**	Internal offshore	External, outsourced
Reuse	**Greenfield**	Brownfield	Legacy
Dev. context	**Single**	Program	Portfolio
Size of dev. team	**<12**	12 - 20	>20

Business context	Commercial, speculative	Contractual	Research, Open-source
# of instance	1	**A few**	many
Specifier Developer relationship	Internal	Internal offshore	**External, outsourced**
Reuse	**Greenfield**	Brownfield	Legacy
Dev. context	**Single**	Program	Portfolio
Size of dev. team	<12	12 - 20	**>20**

Business context	Commercial, speculative	Contractual	Research, Open-source
# of instance	**1**	A few	many
Specifier Developer relationship	Internal	Internal offshore	**External, outsourced**
Reuse	**Greenfield**	Brownfield	Legacy
Dev. context	Single	Program	**Portfolio**
Size of dev. team	**<12**	12 - 20	>20

Business context	Commercial, speculative	Contractual	Research, Open-source
# of instance	1	A few	many
Specifier Developer relationship	Internal	Internal offshore	External, outsourced
Reuse	Greenfield	Brownfield	Legacy
Dev. context	Single	Program	Portfolio
Size of dev. team	<12	12 - 20	>20

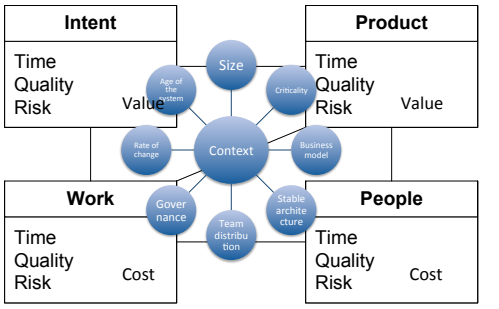
Exercise: Define *your* context



Summary

- Two models:
  1. Commonality (frog)
  2. Variability (octopus)
- Reasoning about project management
- Selection of appropriate process:
  - Lifecycle (time line)
  - Activities (work)
  - Roles (people)
  - Artifacts, workproducts (product, intent, *quality*)

Commonality + Variability



A **project** is all the **work** that **people** have to accomplish over **time** to realize in a **product** some specific **intent**, at some level of **quality**, delivering **value** to the business at a given **cost**, while resolving many **uncertainties and risk.**

All aspects of software projects are affected by **context**: size, criticality, team distribution, pre-existence of an architecture, governance, business model, which will guide with practices will actually perform best, within a certain domain and culture.

04: Standards and Frameworks

Software Project Management
Philippe Kruchten

Module outline

- Project management: models, framework, processes
- Software development process
- Examples of processes
  - RUP, DSDM, MSF, agile approaches
- Project management 1-2-3
- Software development plan

Jargon…

- Model
- Framework
- Process
- Process model
- Process framework

Project Management Theory

- Management theory & Production theory
- Transformation view
  - 1900-1950, Taylor and Ford

  - Time in motion
  - Decomposition
  - Local optimization
- Value view
  - 1950-1975, Drucker & Porter

  - Holistic view
  - Value creation
  - Value chains
- Constraints view
  - 1975 – now, Senge & Goldratt

  - Holistic view
  - Constraints focus
  - Downward serving

Source: L. Koskela & Mike Griffiths

Stages in Software Project Control

1. Chaos
   - Controls: minimal
   - Mantra: Just do it!
   - Lifecycle: undefined
2. Prescriptive Control
   - Controls: conformance to plan
   - Mantra: Plan the work and work the plan
   - Lifecycle: Waterfall & Task-based (or WBS)
3. Adaptive control
   - Controls: conformance to acceptable results
   - Mantra: Embrace change
   - Lifecycle: iterative, feature-driven

 Source: Jim Highsmith

Process

- Process: set of activities intended to achieve a goal
- Process to run a project
  = software development process
  = software engineering process
  = software process
- Software project management is a subset of that software development process

Importance of Process

- Processes represent the collective knowledge on how to run a project
- A Project process has 2 main aspects:
  - Engineering : design test, code, technology
  - Project management
- Good project managers must understand the process
  - Both the engineering process
  - and the management process

 9

Management & Engineering

- Planning, coordinating, leading, controlling
- Focused on people, teams, products & work

- Analyzing, designing, building, testing
- Focused on the quality of the technical solution

 10

Examples of Processes

- Rational Unified Process (RUP)
- Microsoft Solution Framework (MSF)
- Dynamic System Development Method (DSDM)
- eXtreme programming (XP)
- …

 11

Examples of Process Frameworks

- Capability Maturity Model (CMM)
  - A process assessment framework
- IEEE 1074: Standard for Developing Software Lifecycle Processes
- ISO/IEC 12207: Software Lifecycle Processes
- Project Management Body of Knowledge (PMBOK)
- ISO 9000 (?)

 12

Process basics

- Lifecycle
  - Phases, milestone
- Workproducts, artifacts
  - concrete "things" delivered or internal
- Activities, task
  - things to do, recipes on how to do it
- Workflows
  - meaningful sequences of activities
- Roles
  - skills, competencies, responsibilities

 13

Capability Maturity Model

- Developed in the early 1980's by the Software Engineering Institute (SEI)
- Framework for the assessment of software engineering process
- A reference
- A ladder

Source: Jalote

 14

CMM – Key Process Areas for SPM

CMM Level 3
- Integrated Software Management
- Intergroup Communication
- Peer Reviews

CMM Level 4
- Quantitative Process Management
- Software Quality Management

CMM Level 5
- Process Change Management

Source: Jalote

 15

Project Management Body of Knowledge

- PMBOK: Developed by the Project Management Institute (PMI) (USA)
- Initial publication in 1987; republished in 1996
- Adopted by IEEE as IEEE 1490-1998

- Main PMBOK NOT software specific
- 2013: Software Extension
- A certification: Project Management Professional (PMP)

Source: PMI & IEEE

 16

PMBOK: Knowledge Areas

4. Project Integration management
5. Project Scope management
6. Project Time management
7. Project Cost management
8. Project Quality management
9. Project Human resources management
10. Project Communication management
11. Project Risk management
12. Project Procurement management

Looks like MSF, no?

 17

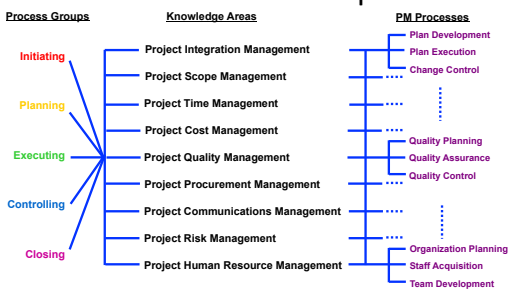
Project Management Body of Knowledge

Project Integration Management	Project Scope Management	Project Time Management
Project Plan Development Project Plan Execution Integrated Change Control	Initiation Scope Planning Scope Definition Scope Verification Scope Change Control	Activity Definition Activity Sequencing Activity Duration Estimation Schedule Development Schedule Control
Project Cost Management	**Project Quality Management**	**Project Human Resource Management**
Resource Planning Cost Estimating Cost Budgeting	Quality Planning Quality Assurance Quality Control	Organizational Planning Staff Acquisition Team Development
Project Communications Management	**Project Risk Management**	**Project Procurement Management**
Communications Panning Information Distribution Performance Reporting Administrative Closure	Risk Management Planning Risk Identification Qualitative Risk Analysis Quantitative Risk Analysis Risk Response Planning Risk Monitoring & Control	Procurement Planning Solicitation Planning Solicitation Source Selection Contract Administration Contract Closeout

Source: E. Lopes Cardozo, Empulsys

 18

PMI's PMBOK Concepts



Source: Bill Cottrell, IBM

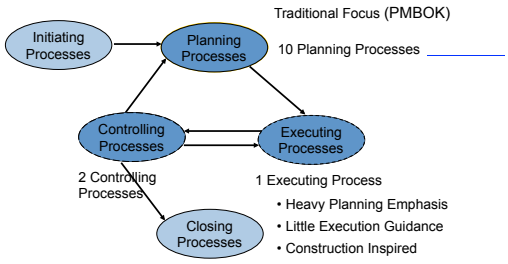
 19

PMI PMBOK: Focus over Time

Source: Bill Cottrell, IBM

 20

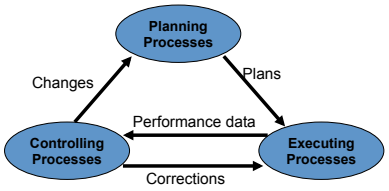
Looking into the PMBOK Philosophy



Sources: L. Koskela & Mike Griffiths

 21

Basic Loops in the PMBOK



Sources: L. Koskela & Mike Griffiths

 22

Basic Elements & Theories & Practices

- Planning processes
  - Well-understood and pushed to the extreme in some industries; WBS; Activities/tasks; Risks
  - Estimation (when? how much?)
- Executing processes
  - Task assignment
- Controlling processes
  - Thermostat model, cybernetic model
  - What to measure; assessing progress?
  - Earned value

 23

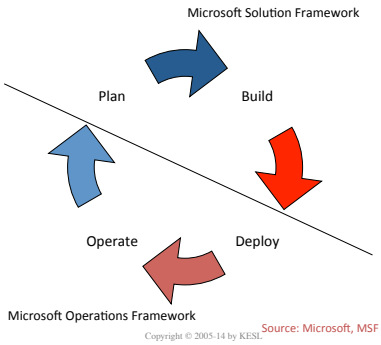
Microsoft Solution Framework

- Derived form an internal process (MS Office group)
- Version 3
  - Six whitepapers and some 40+ templates
  - Training material
  - Available on www.microsoft.com/msf
- Now V4 on Visual Team System (2006)
  - Cf. MSDN
  - MSF for Agile Software Development…

Source: Microsoft

 24

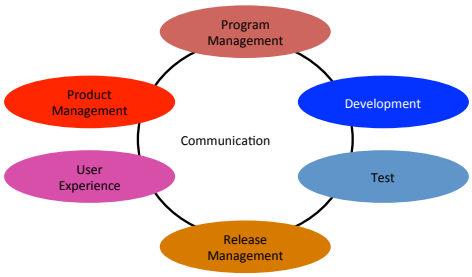
MSF-MOF



Source: Microsoft, MSF v3

 25

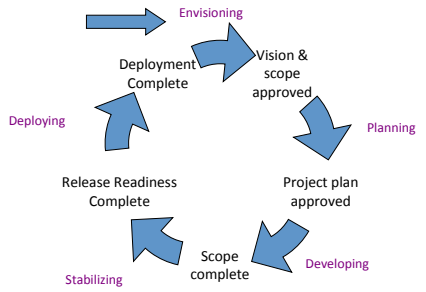
MSF Team Model



Source: Microsoft, MSF v3

 26

MSF - Lifecycle



 28

Source: Microsoft, MSF v3

MSF - Project Management Disciplines

- Integration management
- Scope management
- Time management
- Cost management
- Communication management
- Human resources management
- Procurement management
- Risk management
- Quality management

Source: Microsoft, MSF v3

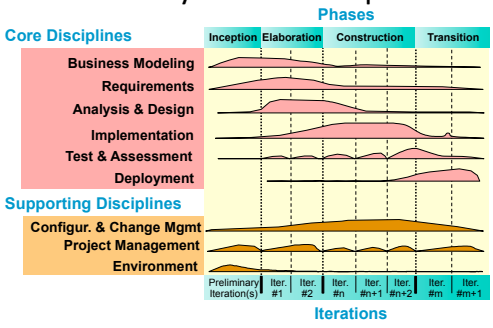
 29

Rational Unified Process

- Developed by Rational Software in 1995-2003
- A product
- HTML-based
- Configurable to a wide range of situations
- Extensible process framework
- Supported by tools

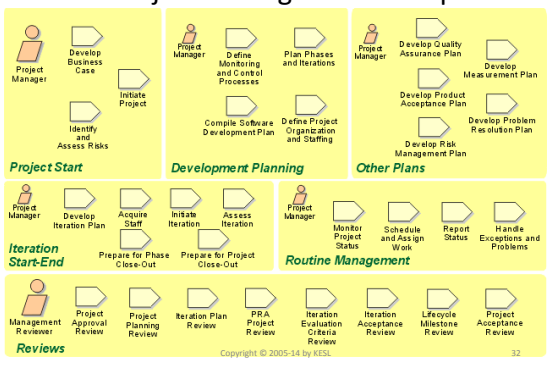
 30

RUP Lifecycle and Disciplines

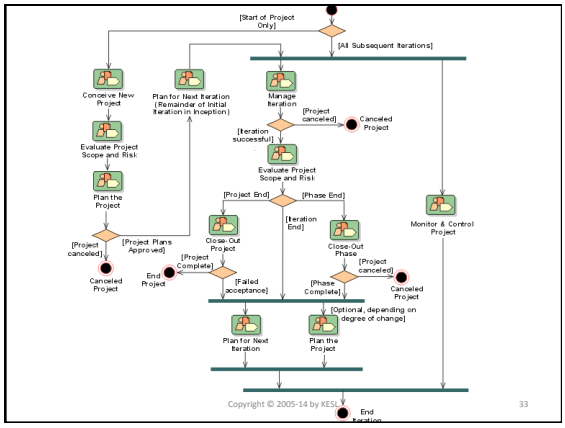


 31

RUP Project Management Discipline

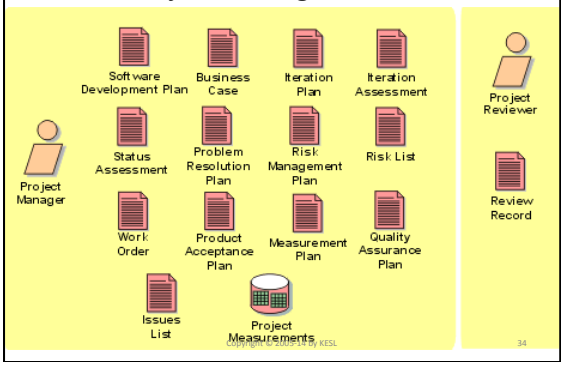


 32



 33

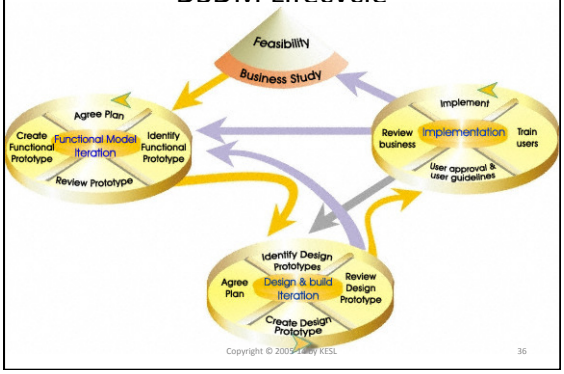
RUP: Project Management Artifacts



Dynamic System Development Method

- DSDM Consortium
- Product, licensed by DSDM Consortium
- Born in the UK beginning of the 90's
- www.dsdm.org

DSDM Lifecycle



Counterpoint to planning: Emergence

- Agile processes:
  The requirements, the design, the process and therefore the product will gradually emerge as the project proceed.
- Very little planning, only rough sketches and envelopes at first.
- Immediate feedback allows driving the project (very short Deming cycles; few artifacts).
- Practices to support this paradigm: Test first, customer on site, Pair Programming, Scrums, Planning game, etc…

Agile Methods

- Agile Alliance
- Late 1990's
- Agile Manifesto
- Kent Beck, Martin Fowler, Jim Highsmith, Alistair Cockburn, Ron Jeffries…
- XP, Scrum, Lean Development (LD), Crystal, Adaptive Software Development, etc… RUP? DSDM?
- See http://www.agilealliance.org

Agility

- A definition
  - Agility is the ability to both create and respond to change in order to profit in a turbulent business environment.

Jim Highsmith (2002)

- Characteristics
  - Iterative and incremental
  - Small release
  - Collocation
  - Release plan/ feature backlog
  - Iteration plan/task backlog

Sanjiv Augustine (2004)

Agile Values: the Agile Manifesto

We have come to value:

- Individuals and interactions *over* process and tools,
- Working software *over* comprehensive documents,
- Customer collaboration *over* contract negotiation,
- Responding to change *over* following a plan.

That is, while there is value in the items on the right, we value the items on the left more

Source: http://www.agilemanifesto.org/

 41

Agile Principles

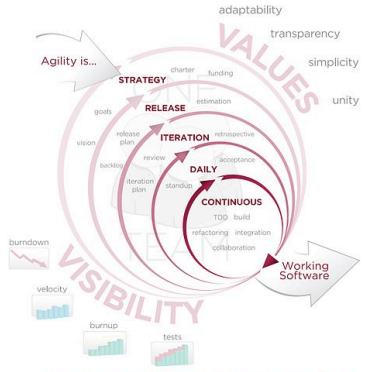
- Customer satisfaction
- Change is OK
- Deliver working software frequently
- Business people and developers must work together daily
- Motivated individuals; right environment; trust
- Face-to-face communication is preferred
- Sustainable development

 42

Agile Principles (cont.)

- Continuous attention to technical excellence
- Simplicity
- Emergence of architecture, requirements and design
- Self-organizing teams
- Self-reflection to become more effective

 43



 44

Named Agile Methods

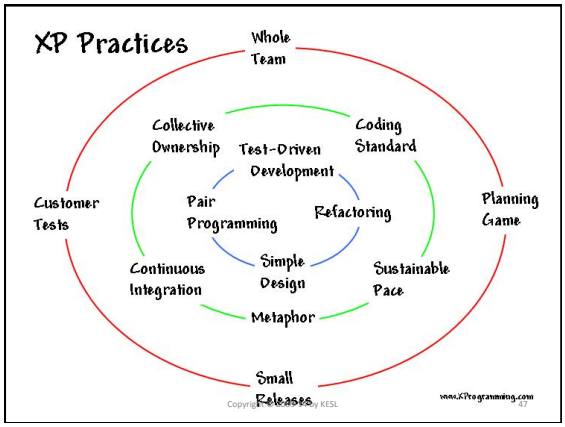
- XP = eXtreme Programming (K. Beck)
- SCRUM (K. Schwaber, J. Sutherland)
- Adaptive development process (J. Highsmith)
- Lean Software Development (M.&T. Poppendieck)
- Crystal (A. Cockburn)
- Feature Driven Development (S. Palmer)
- Agile Unified Process (S. Ambler)
- etc., etc…

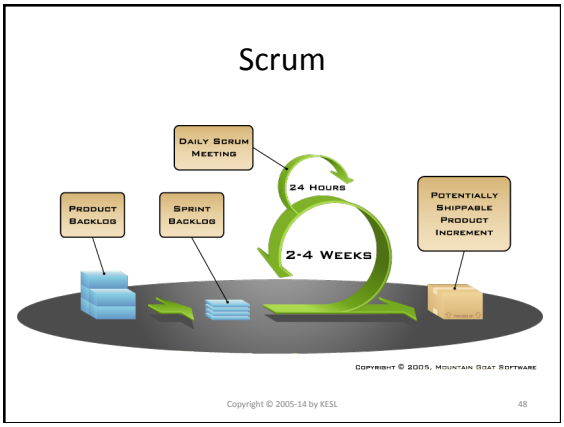
 45

eXtreme Programming (XP)

- Kent Beck, Ward Cunningham, Ron Jeffries
- Values:
  - Communication
  - Simplicity – simplest product that satisfy needs
  - Feedback – obtain and value feedback from all stakeholders
  - Courage – prepared to make hard decisions
- Some thirteen core practices

 46



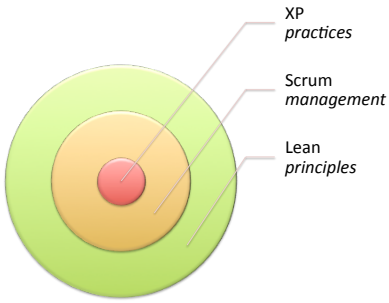


Lean Principles (for Software Development)

- Eliminate waste
- Amplify learning
- Decide as late as possible
- Deliver as fast as possible
- Empower the team
- Build integrity in
- See the whole

 49

Different methods for different issues



 50

Getting at the Essence of Agility

- Software development is a knowledge activity
  - Not production, manufacturing, administration…
- The "machines" are humans
- Dealing with uncertainty, unknowns, fear, distrust…
- Feedback loop ->
  - reflect on business, requirements, risks, process, people, technology
- Communication and collaboration ->
  - Building trust

 51

Key principles

- Feedback loop ->
  - reflect on business, requirements, risks, process, people, technology
- Communication and collaboration ->
  - Building trust

 52

Agile sweet spot

System Size	• 0 ..**12** … 300
Criticality	• **Simple, $ losses**, … deaths
System Age	• Exploratory, **greenfield**, legacy maintenance
Rate of change	• Low, **medium, high**
Business model	• **In house**, Open Source, ….
Stable architecture	• **Stable**, changed, new
Team distribution	• **Collocated**, …, …, offshore outsource
Governance	• **Simple rules**, …, SOX, …

 53

Of process, maps, and plans

"If you do not know where you are going, you will probably end up somewhere else."

(Laurence J. Peter)

"Would you tell me please which way I ought to go from here?"
"That depends a good deal on where you want to get to," said the cat.
"I don't much care where—," said Alice.
"Then it does not matter which way you go," said the cat.

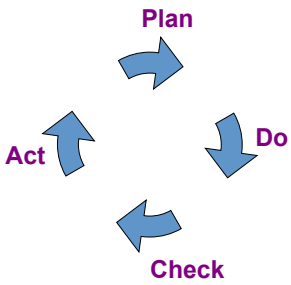
(Lewis Carroll)

 54

Project Management 101: PDCA

- Deming Cycle

After W. Edwards Deming



- Deming called it Shewhart Cycle

55

Deming Cycle

- Plan the short-term objective
  - Determine the time-frame
  - Decide what will be needed
  - Decide who is doing what
- Do what the plan said
  - Collect data
  - Design studies or other stuff
  - Train people …

 56

Deming Cycle (cont.)

- Check to see how the plan was carried out
  - Compare data collected to plan
  - If plan not carried out, then do it
  - Look for lessons for the future
  - Discuss adjustments
  - Determine course of action and changes
- Act on the recommendation of the team
  - Implement fixes, adjustments
  - Inform others of needed changes
  - Improve communication

 57

Kaoru Ishikawa's improvements (1985)

- Plan
  1. Determine goals and targets
  2. Determine methods for reaching these goals
- Do
  3. Education and training
  4. Implement the work
- Check
  5. Check the effect of implementation
- Act
  6. Take any appropriate action

 58

Multiple time horizons

- PDCA at the hour level / individual level
- PDCA at the day or week level for an individual or a small team
- PDCA at the phase level
- PDCA at the project level
- etc…

 59

Multiple entities to apply PDCA to

- Time
- Product
- People
- Other resources

Also
- PDSA = Plan Do Study Act

 60

The road ahead

- For each of the important elements:
  - Time, resource, requirements, etc…
- …we will look at:
  - how to represent them
  - how to PLAN for them
  - what tools and techniques are useful
  - what practices exists in various methods and processes

 61

The Software Development Plan (simple)

- At minimal:
  - Organization and responsibilities
  - Schedule
  - Resources: staff and budget
  - Product overview

 62

Software Development Plan – Full (1)

- Project Overview
  - Purpose, scope, objectives
  - Assumptions, constraints
  - Key deliverables
- Project organization
  - Organizational structure
  - External interfaces
  - Roles and responsibilities

 63

Software Development Plan – Full (2)

- Management process
  - Estimates
  - Project plan
    - Phases
    - Iterations (if any)
    - Schedule
    - Resources
    - Budget

 64

Software Development Plan – Full (3)

- Project monitoring and control
  - Requirement management plan
  - Schedule, control, budget control
  - Quality control plan
  - Reporting plan
  - Measurement plan
- Risk management plan
- Close-out plan
- Technical process plans
  - Process, methods tools techniques
  - Infrastructure plan
  - Product acceptance plan

Software Development Plan – Full (4)

- Supporting Process plans
  - Configuration management plan
  - Evaluation plan
  - Documentation plan
  - Quality assurance plan
  - Problem resolution plan
  - Subcontractor management plan
  - Process improvement plan
  - Communication plan

Tailoring is the key

- Use only the plans that make sense in your context
- First step is to tailor the outline
- Be very clear about what is know and even more about is NOT known
- Be very clear about assumptions, and label them as such

Assessment, monitoring, control, …

- Reports
- Status assessment
- Minutes of reviews and other meetings
- Databases of:
  - requirements
  - issues
  - risks
  - defects

To cover the CA of PDCA (Deming cycle)

Summary

- Software ● Project ● Management
- Process:
  - activities, artifacts=workproducts, workflow, roles
  - techniques, tools, templates, guidelines
- Project:
  - bounded in time
  - unique set of goals & constraints
- Software project management = a (sub) process of software development process

Summary (2)

- Standards and Assessment frameworks:
  - PMBOK
  - SW-CMM (and ISO15504)
  - ISO/IEC 12207
  - IEEE 1074
- Examples of Software Development Processes
  - RUP
  - DSDM
  - MSF
  - Agile: XP, SCRUM, Lean, etc.

Summary (3)

- Deming cycle: PDCA
- Software Development Plan
  - a composite of several plans
  - need tailoring to suit project
  - IEEE std 1058:1998 + others
- Report and Status assessments
- SPM? It's not just about "being the boss"

 71

05: Managing Risks and Uncertainty

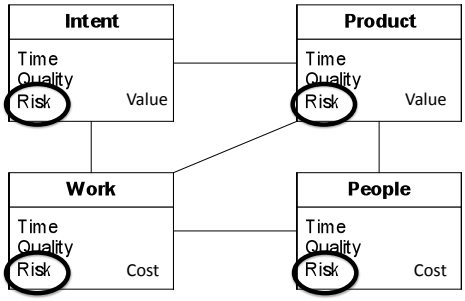
Software Project Management

Philippe Kruchten

Module Outline

- Failure & success
- Uncertainty & risks
- Risk exposure
- Sources of risk in software projects
- Basic risk management strategies
- Iterative lifecycle, as risk mitigation
- Standards & tools

Risk and uncertainty



Success

- Success is meeting the entire set of all requirements and constraints held as project expectations by those in power. (RUP)
- Success vs. failure
  - degrees of success or failure

Uncertainty

- A state of the project that may have many different outcomes, answers, values… currently unknown
- Each outcome may have a probability of occurrence
  - There is a 60% chance that the number of users will be more than 1,000, and a 10% chance that we'll have more than 10,000, and about 0% chance that we'll have more than 1 billion
- Software projects tend to have more more uncertainties than other type of projects.

Risk Defined

- Risk: the possibility of a loss or injury (Webster)
- Risk is the possibility of suffering loss (SEI)
- A risk is an ongoing or impending concern that has a significant probability of adversely affecting the success of major milestones (RUP)

Risk Defined (cont.)

- Risk: The likelihood of an event, hazard, threat, or situation occurring and its undesirable consequences; a potential problem (IEEE Std 1540).
- A risk is whatever may stand in our way to success, and is currently unknown or uncertain.

Success and Failure: Chaos Report 2003

- Success: 34% (was 16% in 1994)
  - On time, on budget, with expected functionality
- Failure: 16% (was 31% in 1994)
  - Cancelled somewhere along the lifecycle, etc…
- "Challenged" projects: 51%
  - cost and schedule overrun, reduced functionality,
  - "restarts" is often the cause

Source: Standish Group

Risk Exposure

Risk exposure (R)
= Probability (R) x Impact (R)

Example

Risk	Probability	Impact	Exposure
Late by a week	10%	$187,000	$18,700
Show-stopper bug in User-Interface	30%	$200,000	$60,000
John leaves	2%	$500,000	$10,000
… etc.	…	…	….
		Total exposure:	$88,700

Risk Management

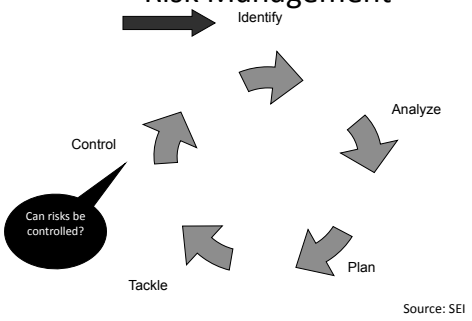
"The readiness is all."
- Hamlet V:ii:215

- Risk Assessment
  - Risk identification
  - Risk analysis
  - Risk prioritization
- Risk Control
  - Risk monitoring ("top 10 risks")
  - Risk planning: avoidance, transfer, reduction
  - Risk resolution
    - prototypes, simulations, analyses, expertise, …

Direct or Indirect Risks

- Direct risk: project has a large degree of control
  - Examples:
    - Wrong workload estimates
    - Poor quality
- Indirect risk: project has little or no degree control
  - Examples:
    - Loss of personnel
    - New competitor on the market

Risk Management



Source: SEI

Risk Identification

- Produce and maintain a list of risks specific to the project
- Start from know lists and taxonomies:
  - SEI, Caper Jones's book
  - Brainstorming sessions:
    - scenarios about what can go wrong
    - no finger pointing, don't shoot the messenger!
    - no attempt to fix anything

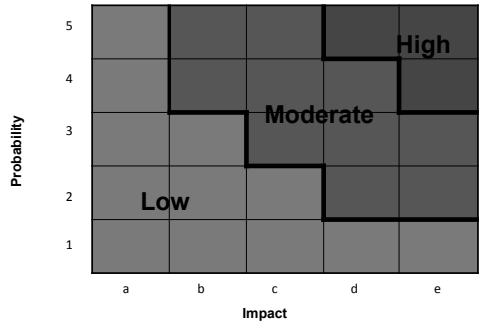
Examples of "Unsatisfactory Outcome"

- Budget and cost overrun
- Product with the wrong functionality or incomplete
- User-interface shortfalls
- Performance shortfalls
- Lack of reliability
- Software impossible to evolve or maintain
- Shortcomings: Management or Technical?

Risk Analysis

- What is affected, what is the cause?
- One or many risks?
- Determine Exposure
- Quantitative ($$$) ? or qualitative?
  - make it simple and effective
  - Impact: low, medium, high (or 1 to 5)
  - Probability: a percentage (or 1 to 5)

Alternative: Using a simple matrix



Prioritize Risks

- You can't tackle everything!
- Put aside the ones you will have to live with
- Bring out a list of the "top ten"
- Make it visible to all involved (external stakeholders and upper management included)

Example of a Starting List

1. Goals not defined properly
2. Goals defined, but changes out of control
3. No proper planning
4. No leadership
5. OK plan, but lack resources to match
6. No contingency in plan
7. Expectations not properly managed
8. Progress not monitored properly
9. No reporting
10. Naïve view of fixing problems (e.g.: "add more resources")

Example of Actual Risks

Risk	Prob	Imp.	Exp	Actions	Warning signs
Burnout	3	3	9		
Scope creep	1	3	3		
Bad mngmt	2	3	6		
People leave	3	3	9		
Late subcontract	2	2	4		
Charles leaves	1	3	3		
… etc.					

Source: O'Connell 2001

Risk Treatment

What do we do with the risks?

- Risk acceptance: live with it!
- Risk avoidance: reorganize to eliminate the risk
- Risk contingency: Develop a "plan B"
- Risk mitigation: reduce the probability or the impact
- Risk transfer: push the risk onto another stakeholder

Example of actual risks (continued)

Risk	P	I	E	Actions	Warning signs
Burnout	3	3	9	??	absences
Scope Creep	1	3	3		
Bad mngmt	2	3	6		delays
People leave	3	3	9	incentives	resumes
Late subcontrac	2	2	4	Xfer >John	delays
Charles leaves	1	3	3	incentive	resignation
Infeasibility	1	3	3		

Risk Management: A Continuous Activity

- New risks pop up
- Impact & likelihood vary
- Update, then sort the list
  - tip: Use an Excel spreadsheet
- Make risk a global concern
  - Publish weekly "top ten"
  - Do something about them (where possible)
- Encourage risk identification
  - No "instant punishment" on the discoverer

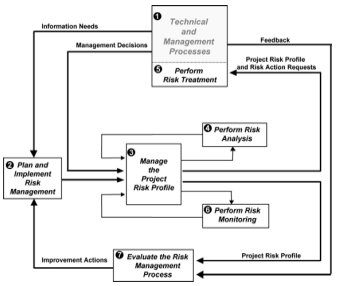
Risk Management: State of Practice

- 20% of organization claim doing it
- 10% are really doing more than risk identification and then problem management
- Issue: is the payoff worth the effort?

Standard on Risk Management

- IEEE standard 1540: 2001
  Leads to a rather heavy processes
- ISO guide 73:2009 Risk Management-Vocabulary
- ISO 31000:2009 Principles and Guidelines for Risk Management

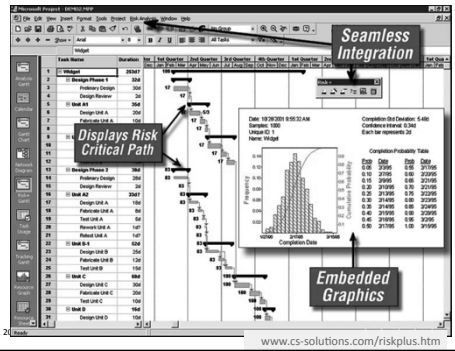
IEEE 1540



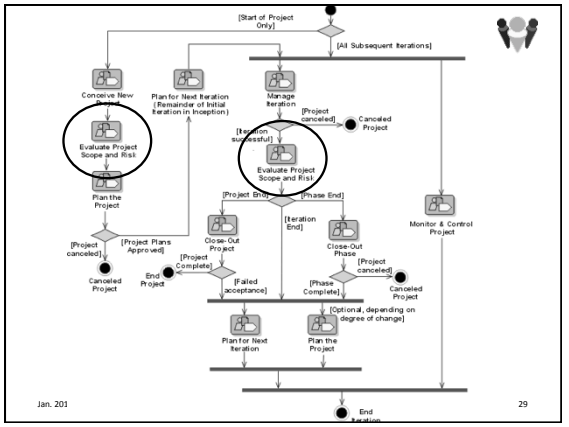
Risk Management Tools

- Spreadsheet (e.g., Excel)
- CS Solutions has a MS Project add-on for schedule-related risks
  – www.cs-solutions.com/riskplus.htm

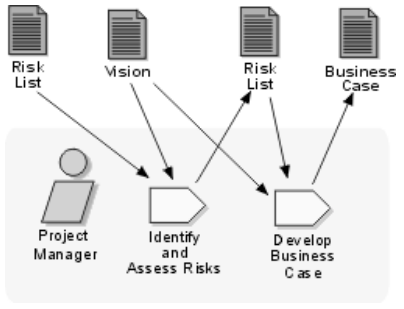
CS Solutions: Risk+ 2.0



www.cs-solutions.com/riskplus.htm



Evaluate Scope and Risks



Activity: Evaluate Risk

Steps:

- Identify Potential Risks
- Analyze and Prioritize Risks
- Identify Risk Avoidance Strategies
- Identify Risk Mitigation Strategies
- Identify Risk Contingency Strategies
- Revisit Risks during the Iteration
- Revisit Risks at the End of an Iteration

Summary: Risks

- Develop and maintain throughout the project a list of risk
  - Define a simple rule for sorting them
- Re-assess weekly risk exposure
- Publicize widely top ten direct risks
- Make sure something is being done about the top ten risks (at least)

  Quote: If you don't actively attack the risks, they will actively attack you. (Tom Gilb)

More on Risk: Threats & Opportunities

- Risk: uncertain event or situation that affects project *negatively*
- What about opportunities? Uncertain events that affect the project *positively*?
- Risk list = threats + opportunities ?

Taking risk?

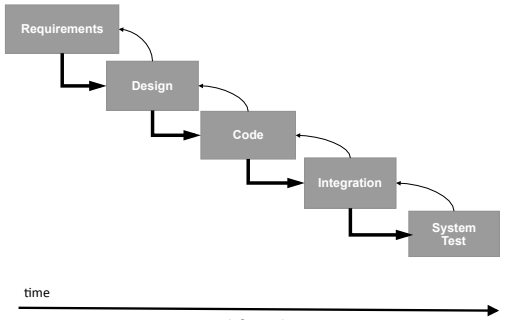
- Tolerance threshold
- Allow small failures
- "Better safe than sorry"

  or
- "No pain, no gain"

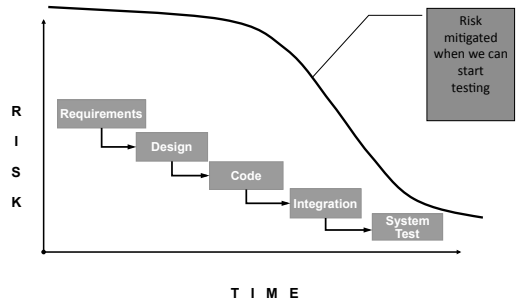
Module Outline

- Failure, success and risks
- Risk management
- Waterfall lifecycle
- Iterative lifecycle

Waterfall Lifecycle



Risks in the Waterfall Lifecycle



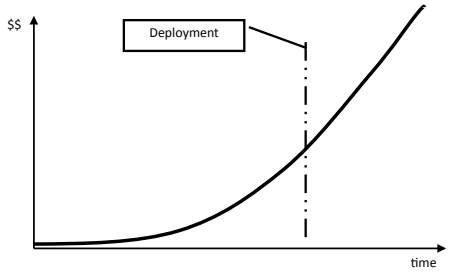
Pros and Cons of the Waterfall:

- Pros
  - Used in other disciplines
  - Well-established
  - Rigorous and rational
  - Minimize waste
- Cons
  - Needs very mature organizations and process
  - Relies on analysis of intermediate artifacts
  - Assumes stable environment and stable requirements
  - Leaves little room for learning
  - and its "feedback loop" too limited (or too long)

More Cons

- An initial design will likely be flawed with respect to its key requirements
- 50% of time and effort spent in Final integration and test
- Late discovery of design defects results in costly over-runs and/or project cancellation
- The time and money spent implementing a faulty design are not recoverable

Cost of fixing problem



Boehm

Limits of the Waterfall Lifecycle

Can't do well:

- Projects with short time-frame
- Projects with lots of unknowns
  - requirements
  - technology
- Projects with lots of "new" stuff
  - People, language, domain, etc..
- Large project….

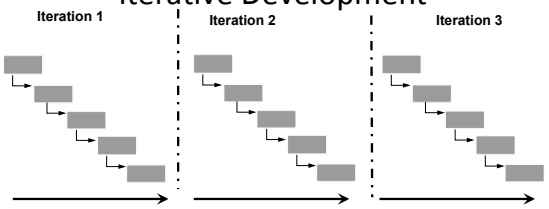
Iterative Development

Quote: "Plan to throw one away; you will anyhow."

(Fred Brooks, the Mythical Man Month, 1975)

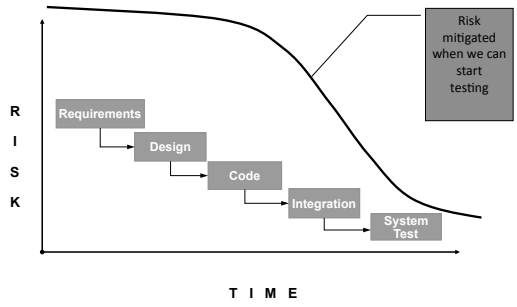
- Design a little, build a little, test a little
- Supported by efficient software development environments
- "Several waterfall one after another" ???

Iterative Development

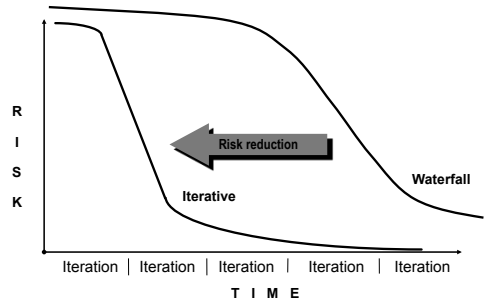


- Each iteration produces an executable release
- Each iteration includes integration and test
- Earliest iterations address greatest risks

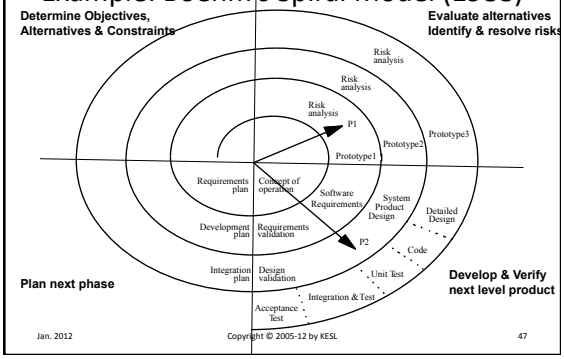
Risks in the Waterfall Lifecycle



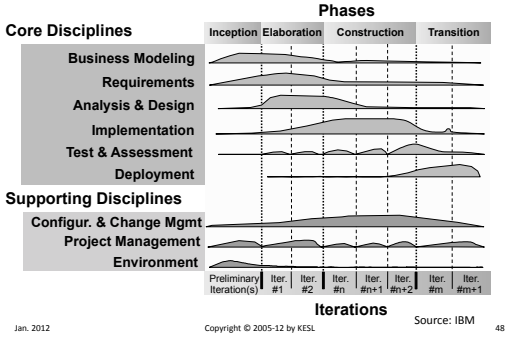
Risks in the Iterative Lifecycle



Example: Boehm's Spiral Model (1988)



Example: Rational Unified Process

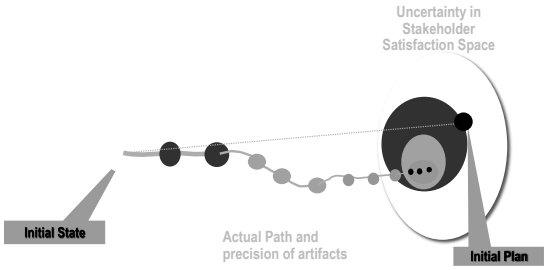


Source: IBM

More Examples

- DSDM
- All Agile Methods
  - XP
  - Crystal
  - Lean Development
- Iterative development acknowledged by major standards

Iterations allow tactical adjustments

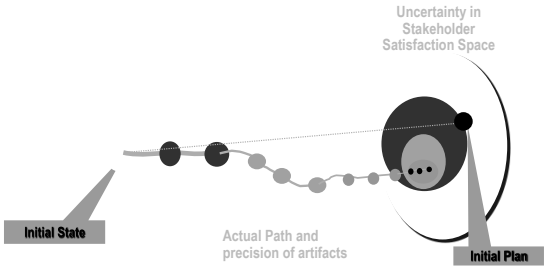


Source: W. Royce, IBM

Impact on Software Project Management

- More work for the manager
  - more planning
- Best way to do active Risk Management
- More predictability in outcome
  - Including earlier opportunity for cancellation
- Greater quality (?)
- Tighter feedback loop
  - requirements, design, process, people, ….

Iterations allow tactical adjustments

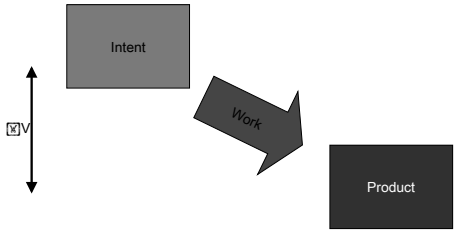


Source: W. Royce, IBM

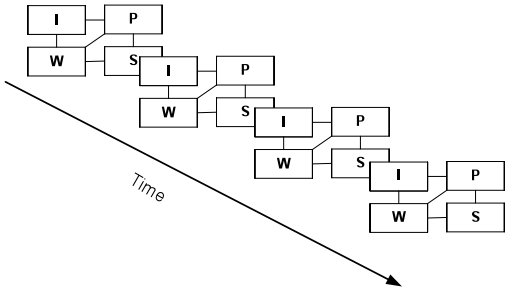
Summary: Iterative Lifecycle

- Traditional Waterfall Lifecycle not too successful with software-intensive projects
- Iterative Development emerged as an alternative in the late 1980's
  - facilitates risk management
  - provides tighter, shorter feedback loop
  - allows for tactical adjustments:
    - requirements
    - engineering (design, technologies, tool support)
    - process, people, budget, etc…

Project: tension between Intent & Product



Project = sequence of [I, W, S, P]

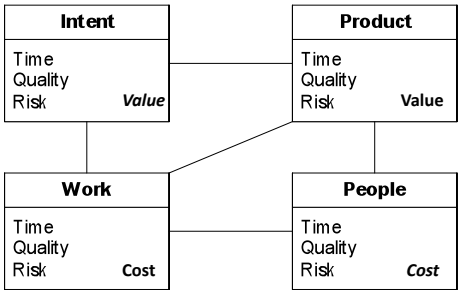


Summary: Risks

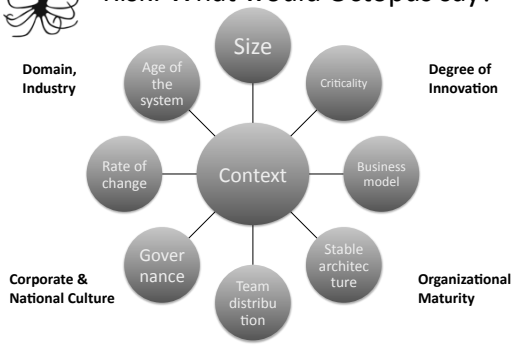
- Develop and maintain throughout the project a list of risk
  - Define a simple rule for sorting them
- Re-assess weekly risk exposure
- Publicize widely top ten direct risks
- Make sure something is being done about the top ten risks (at least)

  Quote: If you don't actively attack the risks, they will actively attack you. (Tom Gilb)

Risks: What would Frog say?



Risk: What would Octopus say?



06: Managing time (1)

Software Project Management

Philippe Kruchten

Module outline

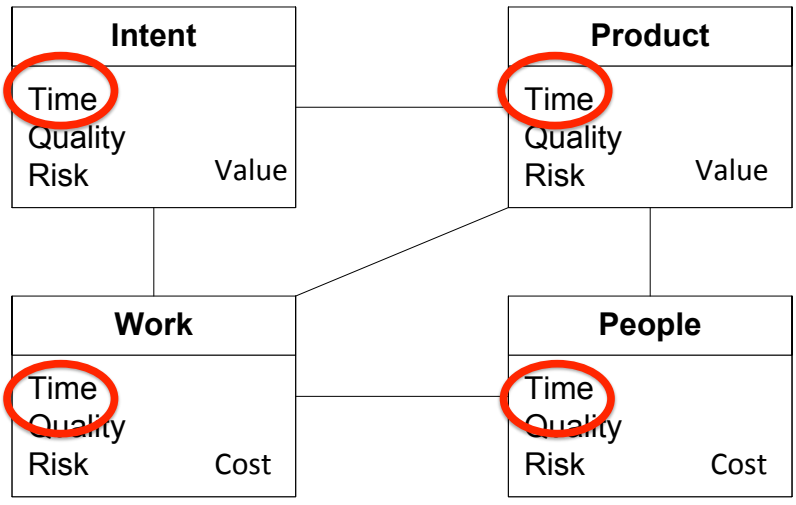
- Effort & Schedule
- Estimation – the black magic
  - Function Points and Use case points
  - COCOMO, SLIM, etc.
  - Wideband Delphi, planning poker
- Time line
  - Phases and Iterations
  - Milestones

Effort and Schedule

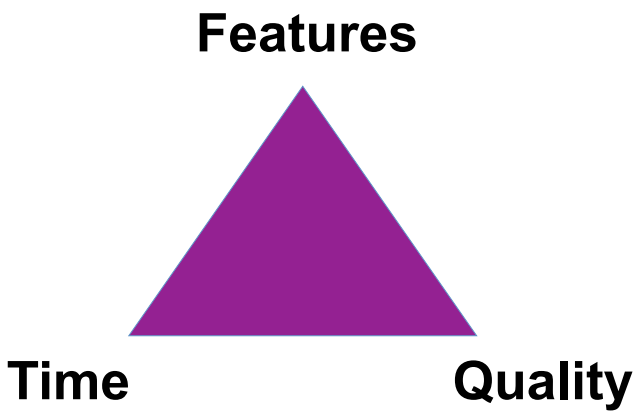
Two questions you will have to answer as a software project manager:

- When will we be done?
  - Schedule
- How much will it cost?
  - Effort

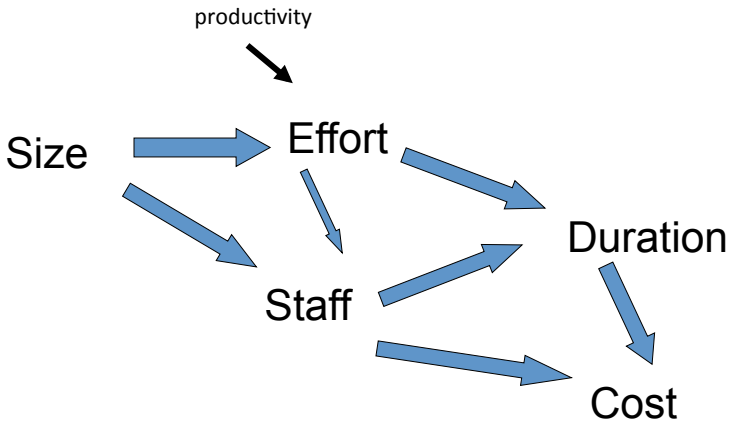
Time



Our Fundamental Conundrum



Entangled Variables



Size, First, then Effort…

- How big is that mountain?
- Units of measure:
  - Functional size: function points, and variants
  - Code size: Lines of code
- Effort: staff-month
- Top-down approaches: analytical
  - Example: COCOMO
- Bottom-up approach: aggregations

… then Schedule

$Duration_{months} = Effort_{staff*month} / Resource_{staff}$

- Cannot trade people and months
  - 7 months with 8 people
    is not the same as
  - 8 month with 7 people
- Brook's Law:
  - Adding people to a late project make it later

Estimation

- Estimation of Size, Effort, Schedule is the most difficult area of *software* development management.
- We know so very little about size estimation, it is almost embarrassing.
- Black Magic …

Estimation

- Estimate *vs.* measure
- Estimate *vs.* commitment

Estimation

Two main approaches:

- Estimation by analogy
- Estimation by proxy

Example

- By analogy: how long will it take?
  - On average it took us 24 person-months to develop a new "BLA" so far (16 data points)
  - This is like a normal "BLA", with some niceties, so let us say 28 person-months
- By proxy: how high is this building?
  - Using the shadow…?

Small experiment: How good an estimator are you?

1. Year Elvis Presley was born
2. Length of the Nile river
3. Latitude of Shanghai
4. Year Gutenberg press started to operate
5. Volume of water in the Great lakes
6. Distance Vancouver – St. John's (NFL)
7. Year Alexander the Great was born
8. Temperature of the surface of the sun

Source Lines Of Code (SLOC)

- Used for many years as a poor substitute for Functional Size
- KSLOC = kilo-SLOC = 1000 lines of code
- KDSI = Kilo Delivered Source Instructions
- No comments, no test data
  - Line in Fortran and Cobol
  - Semi-colons in more recent languages: C, Java, etc.

SLOC: Pros and Cons

+ Easy to measure, unambiguous
+ Simple indicator to monitor progress
+ We have productivity estimates data
  – per language and type of application
- Accounting for rework is hard
- More difficult to use because of new technologies
  – reuse, COTS, model-driven design, patterns
  – Gui builders, application generators

Productivity

- SLOC / Staff-month
- Not coding speed !!!
- For the whole project and all activities together, including those not related to code, such as Project Management.
- Any guesses?

Productivity (SLOC/Staff-Month)

Automation	245 [120..440}
Command and control	225 [95..330]
Data Processing	330 [165 .. 500]
Software tools	260 [143 .. 610]
Military, spatial	100 [45 .. 300]
Scientific	195 [130 .. 360]
Telecommunications	250 [175 .. 440]
Simulations, trainers	224 [143.. 780]
Web	275 [190 .. 975]
Over 500 projects	[45 .. 975]

  Source: Reifer 2002

Functional size

- A size of software derived from quantifying the functional user requirements (FUR)
- FUR examples:
  - Data movements
  - Data manipulations

ISO 14143:1997 – Software measurement - Functional Size – Definition of concepts

FURs

- Data in
- Storage
- Transformation
- Retrieval
- Data out

Function Points

- Started by Allan Albrecht (IBM) in 1979
- Consistent measure of software size
  - independent from technology used
- IFPUG = International Function Point Users' Group
  - www.ifpug.org
- FPA: Function Point Analysis
- Counting Practice Manual, v 4.1 US$75.00
- Certification Program

FP

- Weighted sum of Input, Output, Enquiries, Files…

		simple	medium	complex	
# input	X_1	3	4	6	$C_1 X_1$
# output	X_2	4	5	7	$C_2 X_2$
# enquiry	X_3	3	4	6	$C_3 X_3$
# file	X_4	7	10	15	$C_4 X_4$
# ext i/f	X_5	5	7	10	$C_5 X_5$
					$\sum C_i X_i$

From FP to SLOC

- Cobol: 106 sloc/fp
- Ada: 71 sloc/fp
- C: 150 sloc/fp
- Assembler: 320 sloc/fp
- etc…

Full Function Points

- COSMIC: Common Software Measurement International Consortium
  - Alain Abran (ETS, UQAM), Charles Symons, etc.
- www.cosmicon.org
- Cosmic-FFP Measurement Manual, v 2, 1999
  - Free from UQAM
  - ISO standard 19761
- Open, simpler, more widely applicable
  - MIS *and* real-time

Functional Size Unit

- 1 data movement = 1 Cosmic Functional Size Unit (cfsu)
- 4 types of functional units:
  - Entry, Exit, Read, Write
  - No data manipulation
- Example: Create new employee
  - 1 entry
  - 1 write
  - 1 exit: confirmation or error

  Total: 3 cfsu

Functional Size Unit

- Example: Control temperature a regular interval
  - 1 entry (clock tick)
  - 1 entry (temperature reading)
  - 1 read (target temperature range)
  - 1 exit (on or off command to the heater)

  Total: 4 cfsu

Function Points: Summary

- Need rather detailed requirements specs, and somewhat an embryo of a design (architecture)
- Difficult to master, in spite of the apparent simplicity of the technique
- Less dependent on technology, language, etc.
- Not widely used
- Variants: Feature points, Use Case points

Other approaches to sizing

- Case-Based Reasoning
- In-house database of completed projects
- Expert opinion
- Wideband Delphi

- Build to size
  - Fix the effort, and build whatever fits inside this envelope

Wideband Delphi

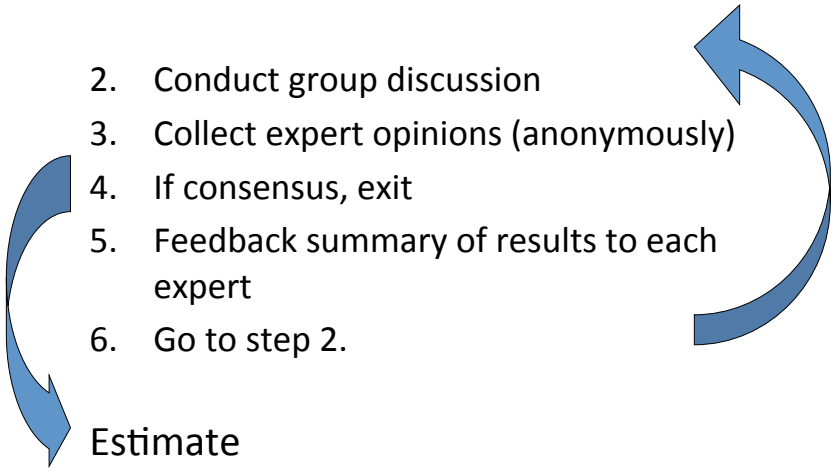
- Delphi: Rand Corp. 1960 (??)
- Expert opinions, feedback, iteration, until consensus is reached.

- Difference
  - Delphi: no group discussion
  - Wideband Delphi: group discussion to speed up the consensus, to understand the variance

Steps in Wideband Delphi

1. Present a problem to panel of experts
2. Conduct group discussion
3. Collect expert opinions (anonymously)
4. If consensus, exit
5. Feedback summary of results to each expert
6. Go to step 2.

Estimate



Evaluation sheet: Example

Project: Operating System *Date* : 6/6/2000

Range of estimates after the 3rd *round*
Your estimate *25 PM*
Median estimate *45 PM*

Your estimate for the next round: 35 *PM*
Rationale for your estimate :
Looks like a standard process control operating system. Our people have had lots of experience with such systems. They should have no trouble with this one. I am increasing my estimate to account for the new DMA channel mentioned by one of the estimators.

Evolution of Estimates

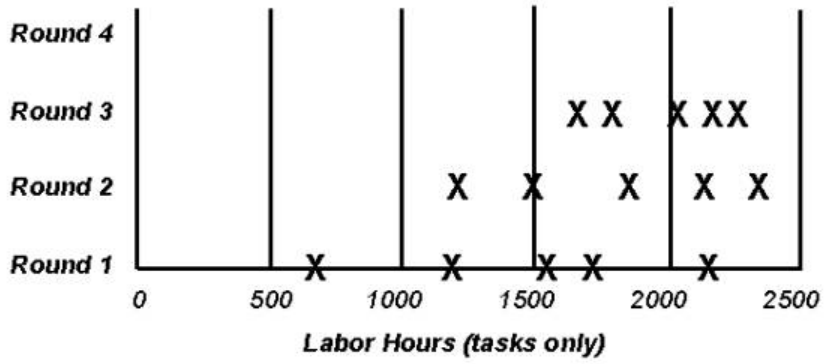


Figure 4. Estimation chart showing three rounds from a Wideband Delphi session.

Wideband Delphi: Pros and Cons

Pros

- Easy and inexpensive
- Educative value as a side effect

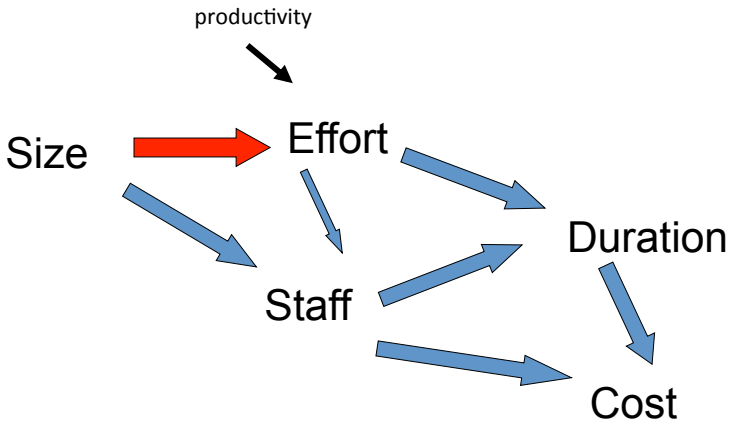
Cons

- May reach a consensus on an incorrect estimate
- Not very repeatable
- False sense of confidence

Agile Practice: Planning poker

- Grenning 2002
- Not very different than Wideband Delphi
- 0, 1, 2, 3, 5, 8, 13, 20, 40, and 100
  - Why not other numbers?
- Process
  - Read description
  - Bid
  - Discuss discrepancies (2 min)
  - Re-bid

From Size to Effort



From Size to Effort

- E = f(S) ?
- Models
  - COCOMO
  - COCOMO II
  - SLIM
  - Price-S

Parametric tools: COCOMO

- COnstructive COst MOdel
- Barry Boehm, TRW, 1981
- A regression model
  - identify some variables
  - regression on a database of known projects
  - find useful relationships on these variables
  - use them to make predictions for other projects
  - grow the database and refine the model

COCOMO Modes = Types of Projects

1. Organic
   - payroll, inventory, scientific
   - small team, little innovations required
   - few constraints
2. Semidetached
   - Compilers, database systems, OS, etc…
   - medium size
3. Embedded
   - real-time constraints, innovation, complex interfaces

COCOMO: 3 levels of sophistications

- Basic COCOMO
- Intermediate COCOMO
  - Bring in some cost drivers
- Detailed COCOMO
  - Program decomposed into major components
  - Process decomposed in phases

Basic COCOMO Effort Formula

$$\text{Effort (E)} = a \times (\text{Size})^b$$

where

- *a* and *b* are constants derived from regression analysis
- Size in KSLOC
- E in staff-month (19 days/month)

Basic COCOMO Effort

- Organic:

  $E = 2.4 \times (\text{Size})^{1.05}$

- Semi-detached:

  $E = 3.0 \times (\text{Size})^{1.12}$

- Embedded:

  $E = 3.6 \times (\text{Size})^{1.20}$

Basic COCOMO Effort

- Project of 200 KLOC
  - That's a large project
- Organic:
  $E = 2.4 \times (200)^{1.05} = 626$ staff-months
- Semidetached:
  $E = 3.0 \times (200)^{1.12} = 1{,}133$ staff-months
- Embedded:
  $E = 3.6 \times (200)^{1.20} = 2{,}077$ staff-months

Basic COCOMO: Time to develop

- Organic:
  - $T_{dev} = 2.5\ (E)^{0.38}$ months
- Semidetached:
  - $T_{dev} = 2.5\ (E)^{0.35}$ months
- Embedded:
  - $T_{dev} = 2.5\ (E)^{0.32}$ months

Basic COCOMO: another example

- 7.5 KSLOC, simple (organic)

Effort: $E = 2.4 \times (7.5)^{1.05} = 20$ staff-months

Duration: $T_{dev} = 2.5(E)^{0.38} = 8$ months

Average staff: $N = E / T_{dev} = 2.5$ staff aver.

Productivity: $P = S / E = 375$ LOC/staff-month

Intermediate COCOMO: Cost Drivers

- Boehm analyzed the source of variance around the basic predictions of his model
- Found 15 variables that affect the effort

- New formula:

$$\text{Effort } (E) = a \times (\text{Size})^b \times C$$

- C = Effort Adjustment Factor (EAF)
  - C = C1 x C2 x… x Cn

Cost Drivers

- Product related
  - RELY – required reliability
  - DATA – size of database
  - CPLX – Complexity, constraints
- Computer related
  - TIME – execution time constraints
  - STOR – main storage constraint
  - VIRT – Virtual machine volatility
  - TURN – Computer turnaround time

Cost Drivers (cont.)

- Project related
  - ACAP – Analyst capability
  - AEXP – Application experience
  - PCAP – Programmer Capability
  - VEXP – Virtual Machine Experience
  - LEXP – Programming language experience
- Personnel related
  - MODP – use of modern techniques
  - TOOL – use of modern software tools
  - SCED – development schedule pressure

Cost Drivers Ratings

Cost Driver Rating

	Very Low	Low	Nominal	High	Very High	Extra High
RELY	.75	.88	1.0	1.15	1.40	
DATA		.94	1.0	1.08	1.16	1.65
CPLX	.70	.85	1.0	1.15	1.30	1.65
TIME			1.0	1.11	1.30	1.66
Etc.	…	…	…	…	…	…

COCOMO Cost drivers

Cost Drivers	*Very Low*	*Low*	*Nominal*	*High*	*Very High*	*Extra High*
Product Attributes						
RELY Required software reliability	.75	.88	1.00	1.15	1.40	
DATA Data base size		.94	1.00	1.08	1.16	
CPLX Product complexity	.70	.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME Execution time constraint			1.00	1.11	1.30	1.66
STOR Main storage constraint			1.00	1.06	1.21	1.56
VIRT Virtual machine volatility*		.87	1.00	1.15	1.30	
TURN Computer turnaround time		.87	1.00	1.07	1.15	
Personnel Attributes						
ACAP Analyst capability	1.46	1.19	1.00	.86	.71	
AEXP Applications experience	1.29	1.13	1.00	.91	.82	
PCAP Programmer capability	1.42	1.17	1.00	.86	.70	
VEXP Virtual machine experience*	1.21	1.10	1.00	.90		
LEXP Programming language experience	1.14	1.07	1.00	.95		
Project Attributes						
MODP Use of modern programming practices	1.24	1.10	1.00	.91	.82	
TOOL Use of software tools	1.24	1.10	1.00	.91	.83	
SCED Required development schedule	1.23	1.08	1.00	1.04	1.10	

Intermediate COCOMO Example

- 10 KSLOC, embedded software
- Cost drivers all nominal except:
  - DATA: not much data, Low -> 0.94
  - TIME: some constraints, High -> 1.11
  - ACAP : good analysts, High -> 0.86
  - PCAP: Good programmers, High -> 0.86
  - VEXP: low -> 1.10
  - etc…
- EAF = 1.17

Intermediate COCOMO Example (cont.)

- 10 KSLOC embedded software
  - $E = 2.8\ (10)^{1.20} = 44$ staff-months
- EAF = 1.17
  - E = 44 x 1.17 = 51 staff-months

Intermediate COCOMO Example (2)

- With all factors at nominal, a project is estimated to be 44 staff-months.
- You hire more capable staff, both analysts and developers, and the ratings goes from 1.0 to 0.86, but staff cost increases from $5,000 to $6,000 per staff-month.
- Adjustments: 44 x 0.86 x 0.86 = 32.6 sm
- Cost differential:
  - $220K – $195K = $24.4 (improvement: 11%)

COCOMO Pros and Cons

Pros

- Versatile and Repeatable
- Can be backfitted from real programs
- Easy to use

Cons

- Somewhat outdated calibration
- Assume waterfall-like development
- Focuses only on development

Aside: Diseconomy of scale

- Effort = A x $Size^B$
- B > 1: diseconomies of scale
- B = 1: linear
- B < 1: economies of scale

COCOMO II: 1999

Main objectives of COCOMO II:

- To develop a software cost and schedule estimation model tuned to the life cycle practices of the 1990's and 2000's
- To develop software cost database and tool support capabilities for continuous model improvement

COCOMO II: 3 models

- The Application Composition Model
  - Good for projects built using rapid application development tools (GUI-builders etc)
- The Early Design Model
  - This model can get rough estimates before the entire architecture has been decided
- The Post-Architecture Model
  - Most detailed model, used after overall architecture has been decided on

COCOMO II differences

- The exponent value b in the effort equation is replaced with a variable value based on five scale factors rather then constants
- Size of project can be listed as object points, function points or source lines of code (SLOC).
- EAF is calculated from seventeen cost drivers better suited for today's methods, COCOMO81 had fifteen
- A breakage rating has been added to address volatility of system

COCOMO 2: Parameters & Potential Impacts

Complexity Parameters	Impact
Size (new, reuse, integration)	Variable
Operational complexity	138%
Timing constraints	63%
Reliability	54%
Storage constraints	46%
Data base size/scope	42%
Reusability	31%

Process Parameters	Impact
Documentation match to needs	52%
Process maturity	43%
Schedule constraints	43%
Risk resolution (Architecture-first)	39%
Precedentedness	33%
Development flexibility	26%
Team cohesion	29%

Team Parameters	Impact
Analyst capability	100%
Programmer capability	76%
Applications experience	51%
Personnel continuity	51%
Language/Tool experience	43%
Platform experience	40%

Tools Parameters	Impact
Multi-site development	53%
Use of software tools	50%
Platform volatility	49%

"Impact" means the range of potential impact between the lowest and highest settings for this parameter

Jan. 2013 60

SLIM

- Peter Norden (IBM, 1960s)
- Defined the Norden-Rayleigh Function
- Lawrence Putnam (QSM, 1970) used regression to derive a predictor: Software Lifecycle Management (SLIM)

Jan. 2013 61

SLIM: Putnam Equation

$$S = C \times K^{1/3} \times T_d^{\ 4/3}$$

where
- S = size in SLOC
- C = an environmental factor
- K = total life-time effort
- T_d = delivery time constraint in years
- development effort: B = 0.39 K

- Compute your C from previous projects
  - 2000=poor, 8000=good, 11000=excellent
  - real-time: 1500

SLIM: example

- Project of 200 KLOC, with a C of 4000

- Total lifetime effort:
  $K = (1/T^4)(S/C)^3$
  $K = (1/T^4)(200000/4000)^3$
  $K = (1/T^4)(50)^3$
  K = 7812 staff-month (in 2 years)
- Development effort
  B = 0.39 K = 3100 sm; staff : 130 persons

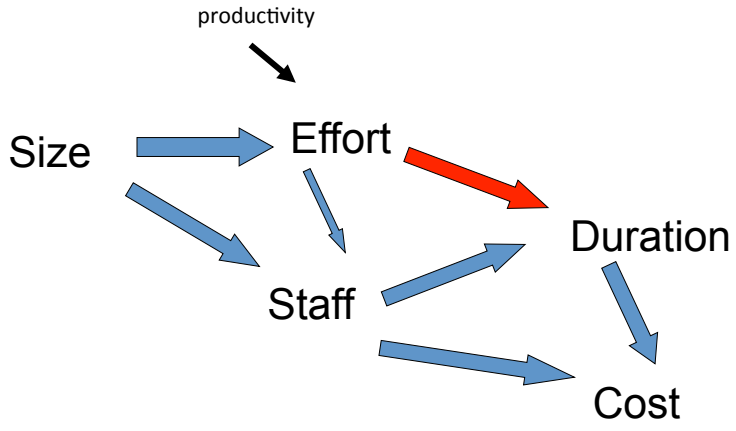
SLIM: Pros and Cons

- Pros
  - Complete tool by QSM is very sophisticated
  - Lots of know-how embedded in the tool
  - Allows calibration with local data
  - Covers complete project
- Cons
  - Works only on large systems
  - Size must be known
  - Estimates very sensitive to technology factor
  - Estimates very sensitive to time factor

McConnell's Table

	System Product		Business Product		Shrink Wrap Product	
KSLOC	Dur	Eff	Dur	Eff	Dur	Eff
10	10	48	6	9	7	15
25	15	140	9	27	10	44
50	20	360	11	71	14	115
100	26	820	15	160	18	270
200	35	1900	20	370	24	610
400	47	4200	27	840	32	1400

From Effort to Duration

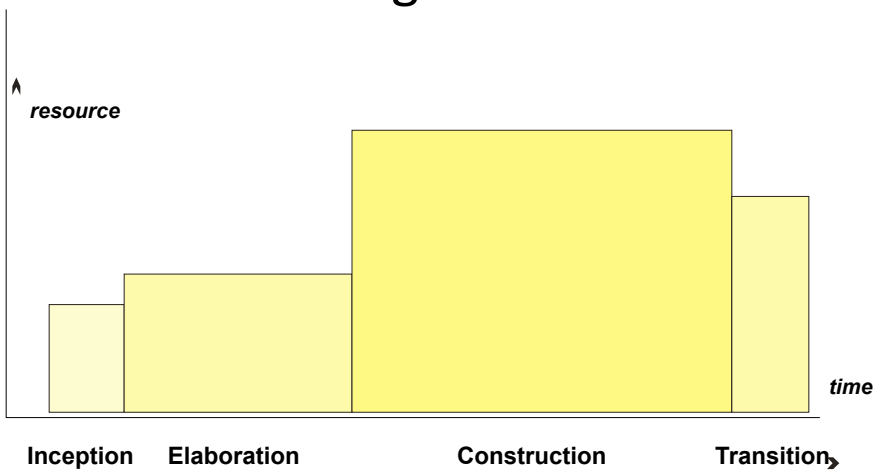


From Effort to Duration: Staffing

- Typical, greenfield development
- First cycle
- Some risks and no predefined architecture

	Inception	Elaboration	Construction	Transition
Effort	5%	20%	65%	10%
Duration	10%	30%	50%	10%

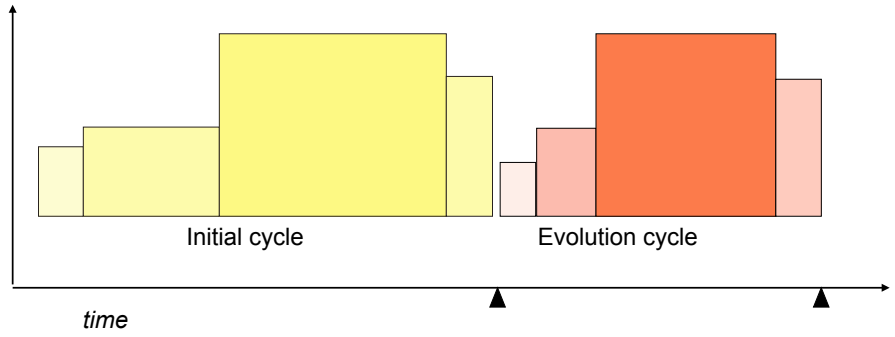
Staffing Profile



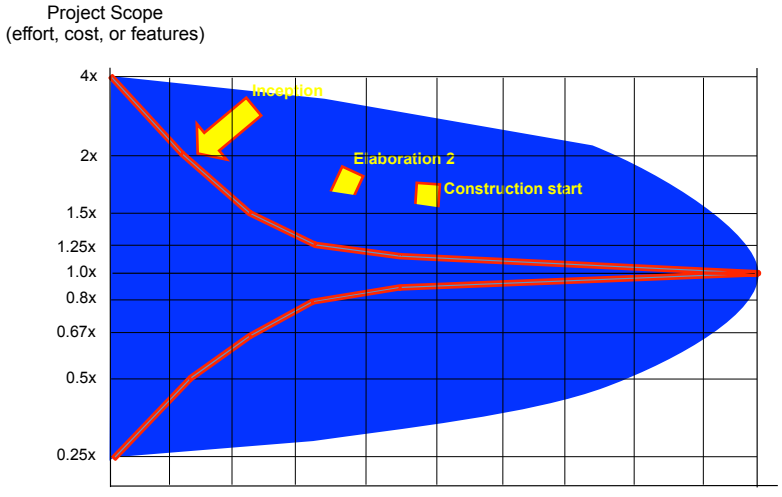
Variations

- Many risk, lost of unknowns
  - Longer elaboration
- No risks, architecture in place
  - Shorter elaboration
- Commercial product
  - Longer transition
- Certification issues
  - Very long transition
- Initial funding issue
  - Long inception

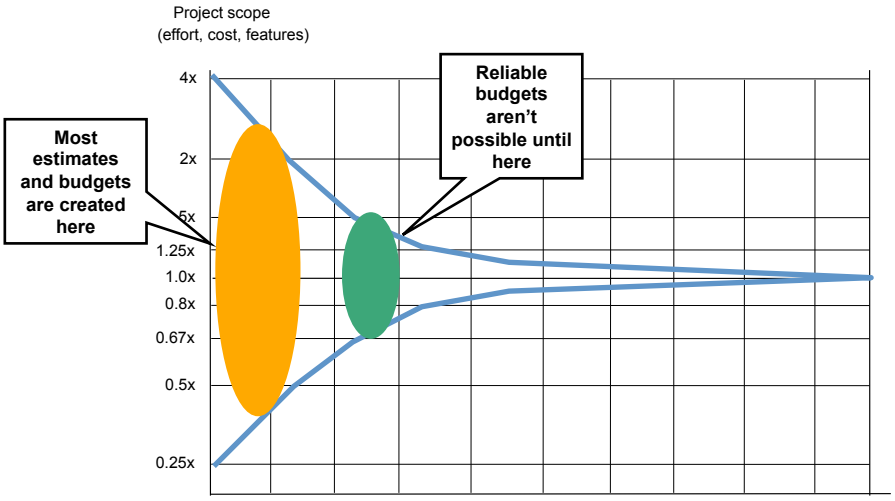
Evolution Cycle



Cone of Uncertainty and Estimate Refinement

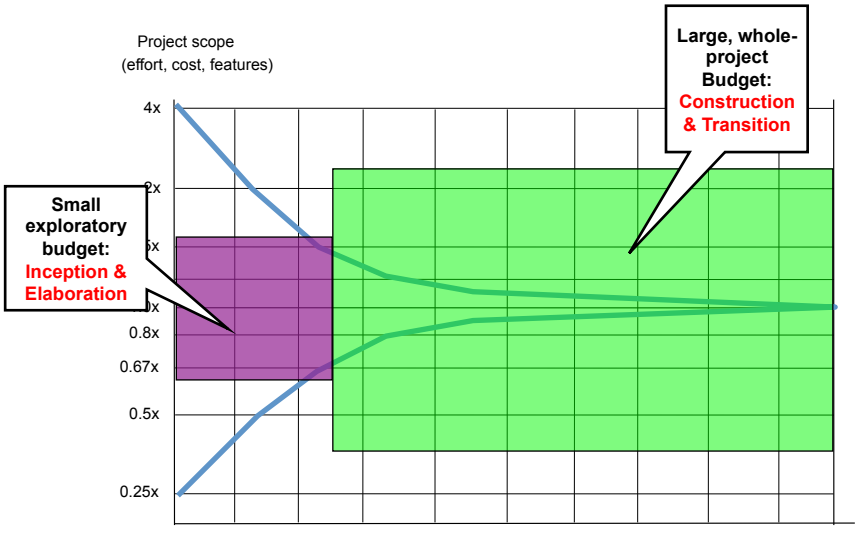


When to define budget?



Jan. 2013 2

Two-Phase Budgeting



Jan. 2013 3

Benefits of Two-Phase Budgeting

- Delays commitment until time when a commitment can be meaningful
- Forces activities that should occur upstream actually to occur upstream
  - Requirements, technical planning, quality planning, risk mitigation, architecture validation, etc.
- Helps set realistic expectations for all project stakeholders
- Improves coordination with non-software groups
- Improves execution by putting plans on more informed basis

Iteration Planning

- One iteration
  - Results in one executable version
  - Mitigates some risk
  - Implement ssome “stuff”
  - Builds up on previous iterations
  - Allow some feedback
    - Product
    - Process
    - People

Duration of an Iteration

- Duration driven by
  + size of organization
  + size of project
  - familiarity with the process, maturity
  - technical simplicity

For large projects, I use the rule of thumb:

$$W = \sqrt{(S/1000)}$$

  - W in weeks
  - S in SLOC

Examples

- 5 people, 5 weeks:
  - 1 week iterations
- 10 people, 6 to 12 weeks
  - 2 week iteration
- 130 people, 5 years
  - 3 to 6 month iterations

Iterations: How Many?

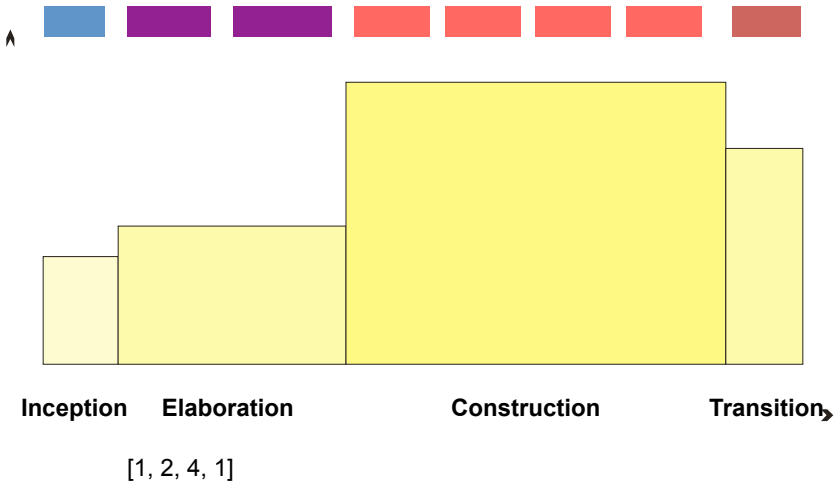
- 6 plus or minus 3

Inception : 0 .. 1
Elaboration : 1 .. 3
Construction : 1 .. 3
Transition : 1 .. 2

Iterations: How Many?

- Very efficient organizations:
  - Short iterations
  - as many as 15 in a single project
- Do not confuse "builds" with iterations
  - Nightly build, weekly build
  - Iteration involves more of the lifecycle activities

Phases and Iterations



[1, 2, 4, 1]

Capturing the Information

- Estimation hypothesis
- Estimations

- Duration =>
- Major milestones

- Iteration length =>
- Minor milestones

Capturing the Information

- Schedule
- Goes into Software Development Plan
  - 1.1.2 assumption and constraint
  - 1.1.4 Schedule and budget summary
  - 5.1.1 Estimation plan
  - 5.1.2 Staffing plan
  - 5.2 work plan
  - etc…

Worked out Estimation Example

- Assumptions:
  - Small Project,
  - Business support,
  - Few technical Risk,
  - Greenfield development,
  - Distributed system (internet based)
- Start from Use Cases
- Using Use Case Point

Unadjusted Use-Case Points UUCP

- 1 Complex Use Case
- 5 Medium Use Cases
- 6 Simple Use Cases

- Weight:
  - Complex :15
  - Medium: 10
  - Simple: 5
- UUCP = 1 x 15 + 5 x 10 + 6 x 5 = 95

Adjust for Technical Complexity

- Distributed 2 x 5
- Responsive 1 x 2
- Simple 1 x 0
- etc…

- TCF = 0.6 + (0.01 x TFactor)

- TCF = 0.6 + (0.01 x 32) = 0.92

Adjustment for Environment

- Familiar with internet 4 x 1.5
- Familiar with the application 4 x 0.5
- Stable requirements: 2 x 2
- etc…

- EF = 1.4 + (-0.03 x EFactor)

- EF = 1.4 - 0.03 x 25 = 0.65

Adjusted Use-Case Points

- UCP = UUCP x TCF x EF

- UCP = 95 x 0.92 x 0.65 = 57

Effort

- 20 to 28 staff-hour / UCP
  - Here some local data helps!
- 20 * 57 = 1140 staff-hour
- = 163 staff-day
- = 8.5 staff-month
- Average Staff
  - S = √8.5 = 3

Schedule & Cost

- 3 + months…
- as 7 iterations of 2 weeks each:
  - [1, 2, 3, 1]
- Staff:
  - 2 for inception and elaboration,
  - 3 for construction and transition
- Cost:

  1 x 2 x 2 + 2 x 2 x 2 + 3 x 3 x 2 + 1 x 3 x 2

  = 4 + 8 + 18 + 6 = 36

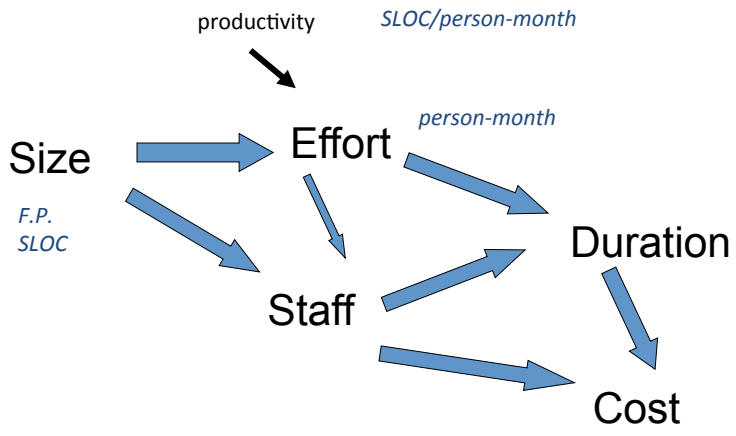
  = 9+ staff month

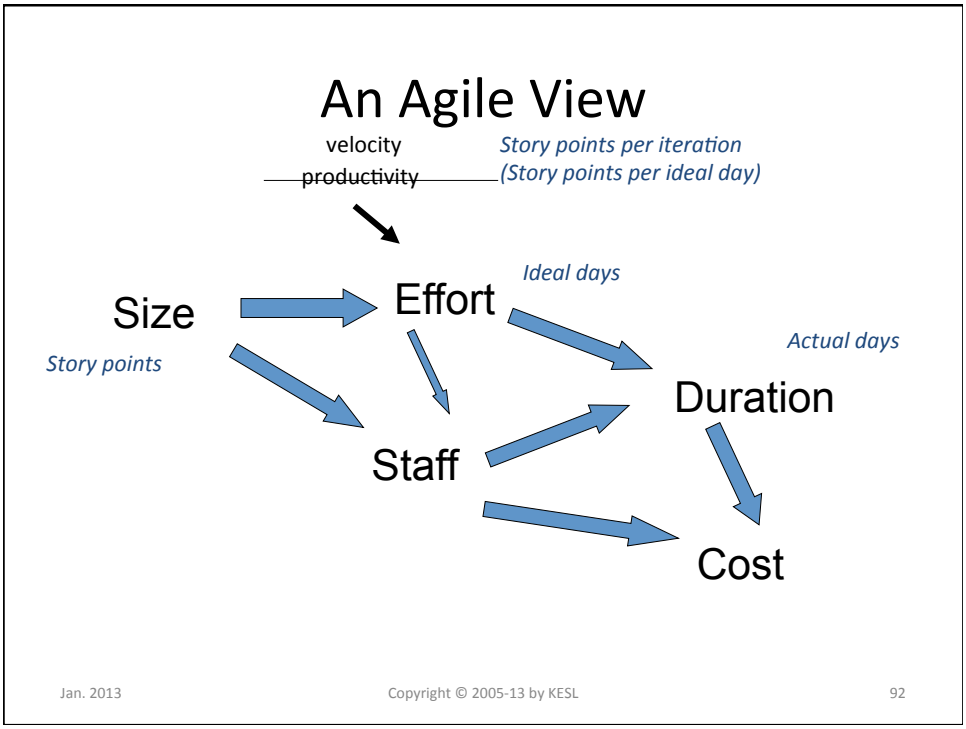
  = 10 sm x $6,000 = $60,000

Ooops! Issue?

- Using McConnell's table
  - Business system
  - 9 staff-month goes with 6 months duration
  - and 10 KSLOC
- Risk?
  - Are we over-optimistic?
  - What make us think we can pull it in 3 months?
  - Is 10 KSLOC realistic, or totally out of this world?
  - Can we compare with another similar project?

The classical View





Summary

- Estimation of size is hard
  - SLOC, FP, FFP, UCP (etc.)
  - Wideband Delphi, and similar "wet finger" techniques
- From Size to Effort & Duration
  - Models: COCOMO 1, COCOMO 2, SLIM
  - Cost drivers: up and down, technical and non-technical
- Use multiple approaches when possible
- Estimate repetitively
  - as soon as more information is known
  - minimum, once per iteration
- Gather and keep data on past projects

07: Managing time (2)

Software Project Management
Philippe Kruchten

Module outline

- Schedule
- Techniques
  - Graphs
  - Critical path
- Other similar techniques
- Applicability in Software development

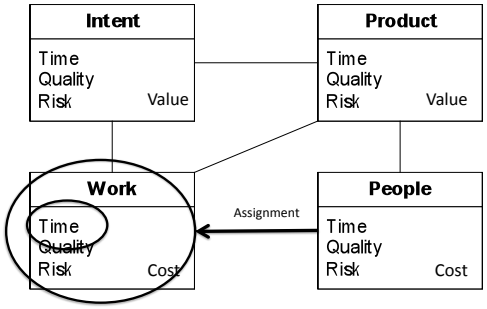
"Time is *not* money; with money you can put it on the table and you can see it, and if you leave it, it may even accumulate - whereas with time, you can't see it or touch it. It expires at a regular and consistent rate whether you use it or not."

Keith A. Pickavance (2008)

Acknowledgment

Most material is from Prof. Anthony Finkelstein.
– Used with his permission

Schedule: Work over time + Resource utilization



Resource Scheduling

- Graphical networks and associated calculation techniques assist effective thinking by providing a step-by-step routine for coordinating work assignments and resource utilisation with project objectives
- Control criteria for the evaluation of work progress are established and the most economical means for correcting delays are diagnosed

Network Scheduling

- A network depicts the sequence of activities necessary to complete a project
- Segments of a project are represented by lines connected together to show the interrelationship of operations and resources
- When a duration is associated with each segment, the model shows the time distribution of the total project and its operations, this information can be used to coordinate the application of resources

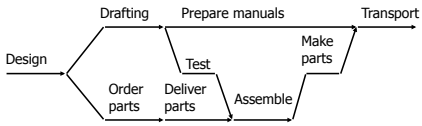
Techniques

- 2 techniques for network scheduling
  - Critical Path Method (CPM)
  - Project Evaluation and Review Technique (PERT)
- Path of critical activities that control the projects duration :
  - Critical Path Scheduling (CPS)
  - CPS is a management control tool for defining, integrating and analysing what must be done to complete a project economically and on time

Fundamentals

- Regardless of the name or the graphical conventions the fundamentals are the same
  - determine a list of necessary activities
  - establish a restriction list that sets the order of activity accomplishment
  - combine the two lists to construct a network

Arrow Diagram



shows operations required to produce custom made machine

1.Activity List

- Break the project down into its component operations to form a complete list of essential activities
  - Without a valid list all subsequent steps are meaningless!
  - An activity is a time-consuming task with a distinct beginning and endpoint
  - Some easily identified characteristic should be associated with each start and finish point

Tips on How to Construct an Activity List

- The way in which activities are defined may be influenced by planning purposes
  - for example if certain types of skilled people are in short supply activities requiring the limited resources should be separated from other operations
- The method or systems development process you are using should help you identify a “first-cut” activity list
  - for example the waterfall model identifies activities (and some sequencing)

2. Checking the Activity List

- A network is a composite picture of an entire undertaking, the activity list needs to be reviewed by suppliers, cooperating departments, subcontractors and anybody whose work impinges on the project

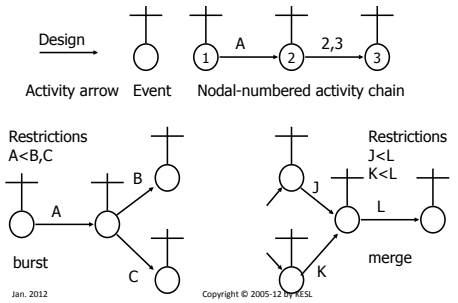
3. Restriction List

- Establishes the precedence of activities
- Uses the rough sequence generally arising from the activity list
- Each activity bracketed by the activities which must immediately precede it, the *prerequisite*, and the activity that must immediately follow it, the *postrequisite*

Activity and Restriction Lists Example

Description	Symbol	Prerequisite	Postrequisite	Restriction List
Design	A		Drafting, Order parts	A<B,C
Order parts	B	Design	Deliver parts	B<D
Drafting	C	Design	Prepare manuals, Make parts	C<E,F
Deliver parts	D	Order parts	Assemble	D<G
Prepare manuals	E	Drafting	Transport	E<I
Make parts	F	Drafting	Assemble	F<G
Assemble	G	Deliver parts, Make parts	Test	G<H
Test	H	Assemble	Transport	H<I

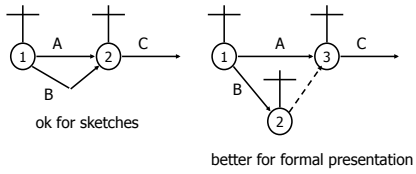
Network Conventions



Dummy

- A dashed line arrow is used in a network to show the dependency of one activity on another
- It is called a *dummy* activity and has all the restrictive properties of regular activities except that it takes zero time
- There are two types of dummies:
  - logic dummies which represent constraint relationships between nodes
  - artificial dummies which assist in numbering and uniquely identifying nodes

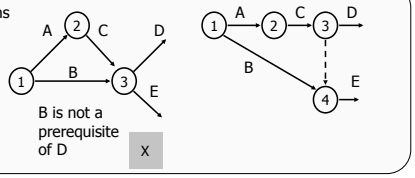
Artificial Dummies



Common Errors I

SPILLOVER

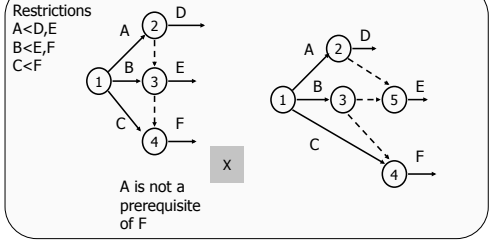
Restrictions
A<C
B<E
C<D,E



Common Errors II

CASCADE

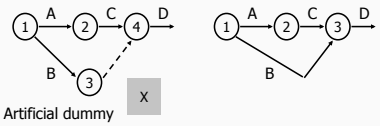
Restrictions
A<D,E
B<E,F
C<F



Common Errors III

UNNECESSARY

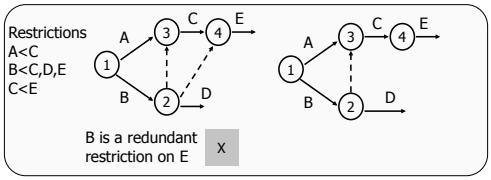
Restrictions
A<C
B<D
C<D



Common Errors IV

REDUNDANT

Restrictions
A<C
B<C,D,E
C<E



Critical Path

- The key to network scheduling is the critical path, the chain of activities that determines the total project duration
- Two approaches are available for estimating activity duration
  - a deterministic approach relies on a single most likely time estimate (this is probably sufficient for software projects)
  - a statistical approach uses a range of possible activity times to determine a single duration

Most Likely Time Estimates

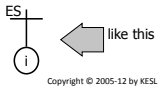
- An estimate of the elapsed time required to accomplish an activity is called the *activity duration*
- An activity duration may be expressed in months, weeks or days, it should not reflect uncontrollable contingencies, estimates for activity duration MUST be derived from objective sources, once activity durations have been identified it may be necessary to revise the network
- The activity duration should be entered besides the appropriate arrow in the network

Boundary Time Calculations

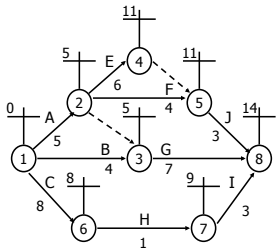
- *earliest start (ES)* the earliest time an activity can begin when all preceding activities are completed as rapidly as possible
- *latest start (LS)* the latest time an activity can be initiated without delaying the minimum project completion time
- *latest finish (LF)* the LS added to the duration (D)
- *total float (TF)* the amount of surplus time allowed in scheduling activities to avoid any interference with any activity on the critical path, the slack between the earliest and latest start times (LS — ES = TF)

Calculating ES

- Make a forward pass through the network adding each activity duration in turn to the ES of the prerequisite activity, dummies are treated exactly the same as other activities
- When a merge is encountered the largest ES + D of the merging activities is the limiting ES for all activities bursting from the event, each limiting ES is recorded on the left bar of the event markers

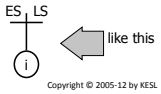


Earliest Start Example



Calculating LS

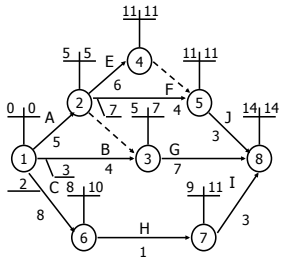
- Basically the reverse of that for ES, make a backward pass through the network subtracting activity durations from the limiting LS at an event
- The limiting LS, the smallest one at a burst event, is entered on the right bar of the cross
- Subsequent LS's are calculated by subtracting activity durations from the LS on the next node cross



Calculating LS (Continued)

- The main difference between LS and ES calculations is that each activity from a common event can have a different LS while all activities starting from the same event have the same ES, to deal with this a *shelf* is added to each activity in a burst that has a larger LF value than the limiting one
- The initial step in LS calculations is to make the right bar of the last cross in the network agree with the left bar, successive subtractions of activity durations from each limiting event should eventually lead to a zero LS for the first network node

Latest Start Example

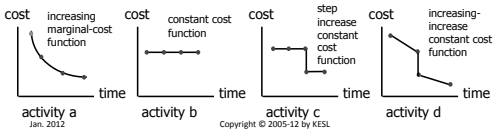


each event with the same entries on the bar is critical

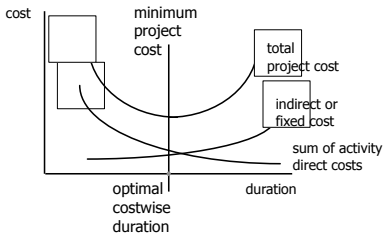
a line connecting these events along activities responsible for the event times is the critical path

Time-Cost Trade-offs

- An initial network often reveals that a project will take longer than anticipated
- The critical path exposes the group of activities from which cuts should be made to shorten the project but it does not indicate which cuts will be least expensive



Project Cost Patterns



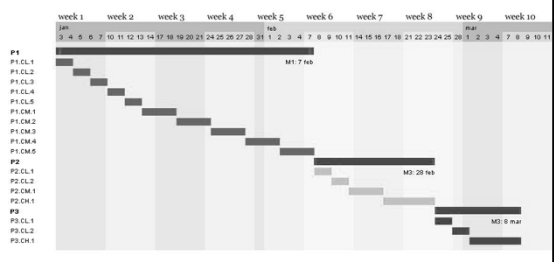
Summary: Scheduling

- Project planning and scheduling are essential skills for the software engineer.
- Resource scheduling is a core issue. Critical path scheduling is a simple technique to achieve this, there are many software tools to support it.
- A schedule is only any use if it is realistic and maintained up to date as the project proceeds.

Other techniques

- Gantt chart
- PERT
- Technique can be enhanced with probabilities and risks

Gantt chart

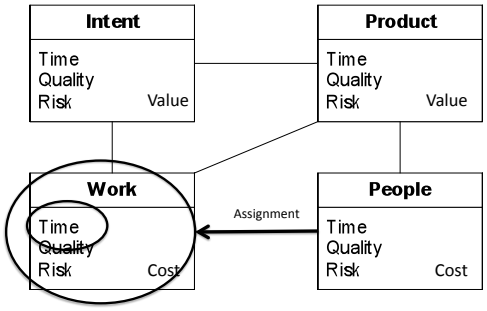


Scheduling in Software development

- Much uncertainty
- Dependencies discovered late
- New tasks discovered late
- Big effort to keep data up-to-date
- OK at a very high level (release planning)

- Needed however when software is a component of a larger system for indentifying dependencies and integration points, and slippage
- Needed for programs and portfolio management

Schedule: Work over time + Resource utilization



Exercise: Kitchen renovation

- Estimate the effort
  - Planning poker.
- Establish dependencies
- Built the schedule
- Find the total duration
- Find the critical path

Exercise: Kitchen renovation

- Install a new hardwood floor.
- Remove, sand, re-paint the cabinets.
- Replace tile countertop with granite.
- Repaint entire kitchen.
- Lay shelf paper.
- Install recessed lighting.
- Install a built-in refrigerator.
- Replace existing oven with a new one.
- Run a water line to existing island and add a sink.
- Replace existing simple window with a bay window.

08: Managing Quality

Software Project Management

Philippe Kruchten

Module outline

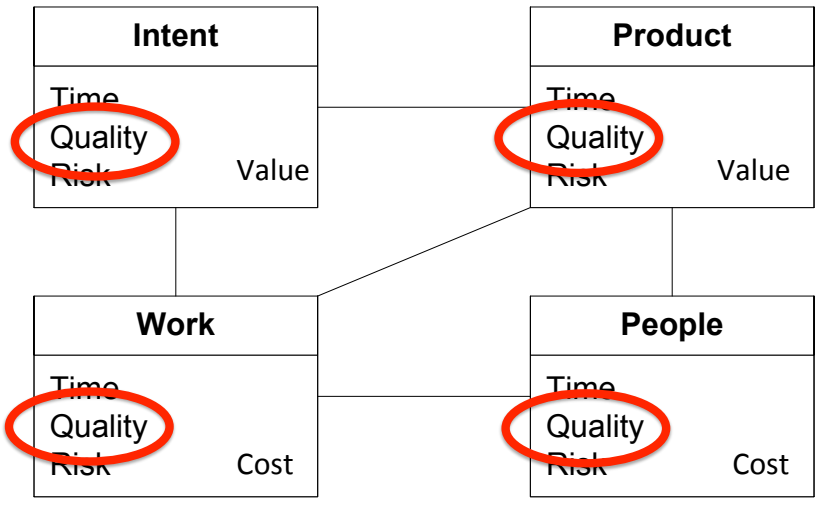
- What is software quality?
- Quality management
  - PDCA
- Defects, the measure on non-quality
- Measurement
  - GQM, ami
- Quality standards
  - IEEE, ISO, others

The Third Question

1. When will we be done? *(schedule)*
2. How much will it cost to get there? *(resource)*

3. Will they like it? *(quality)*
   - Will it be good enough that they will pay good money for it?
   - Will they buy more from us in the future?
   - Will they say good things about us….

Quality



Key issues

- Defining what is quality
- Quality, yes, but at what cost?
  "If quality is good, more quality must be better."
- Who is doing it? When?
- When do we stop? How do I know that we have "enough" quality?

Quality Defined

Traditionally:

- Quality of design, or construction
  - Good Craftmanship, good materials, etc…
- Quality of conformance
  - Norms, standards, codes
- Is this all?
- User satisfaction = compliant product + good quality + delivery within budget and schedule

GEQ: Good Enough Quality

- Not so bad
  "We're not dead yet"
  and as long as customers do not sue us….
- Positive infallibility
  "Anything we do is good"
  We do not even entertain negative thoughts…
- Righteous exhaustion
  "Perfection or bust"
  It is the effort that counts…

Source: James Bach

GEQ: Good Enough Quality (cont.)

- Customer is always right
  "Customers seem to like it"
  So, why bother more….
- Defined process
  "We use a good process"
  If the quality is not there, it cannot be our fault
- Static requirements
  "we satisfy all the written requirements."

Source: James Bach

GEQ: Good Enough Quality (cont.)

- Accountability
  "We fulfill our promises"
  Quality defined by contract.
- Advocacy
  "We made every reasonable effort."
- Dynamic tradeoff
  "We weigh many factors."

Source: James Bach

When failure is not an option

- Certain type of systems: safety-critical systems, which may endanger human life, certain types of failures are not an option.
  - Medical
  - Aviation, space
  - Transportation
  - Nuclear
  - Weapons

Quotes

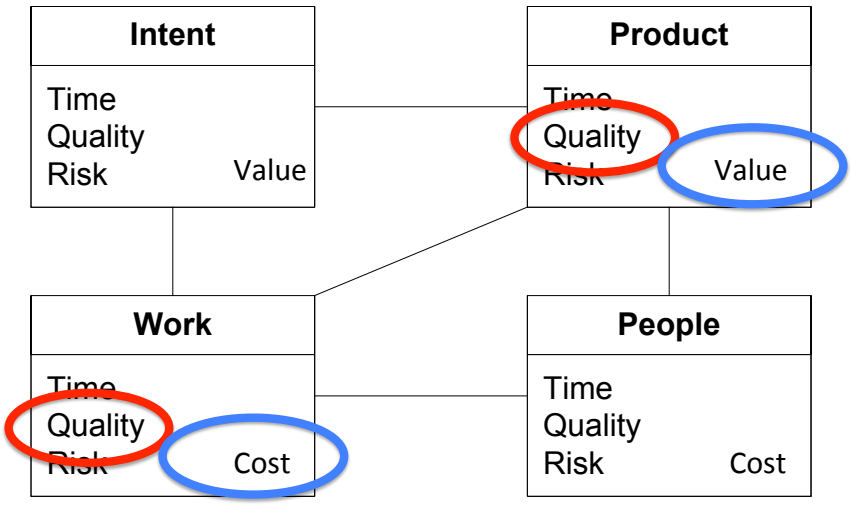
- At the leading edge, there are no subjects, no objects, only the track of Quality ahead, and if you have no formal way of evaluating, no way of acknowledging this Quality, then the train has no way of knowing where to go. Pirsig, 1977

- And what is good, Phaedrus,
  And what is not good—
  Need we ask anyone to tell us these things?
  Plato 370 BC

More quotes

- I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of meager and unsatisfactory kind. (Lord Kelvin, 1900)
- The difficulty in defining quality is to translate future needs into measurable characteristics, so that a product can be designed and turned out to give satisfaction at a price the user will pay. (Shewhart).

Quality = Value, at a Cost



Jan. 2013 13

Quality: Definitions from IEEE 610

Narrow view

- "The degree to which a system, component, or process meets specified requirements."
- "The degree to which a system, component, or process meets customer or user needs or expectations."

Broader view

Jan. 2013 15

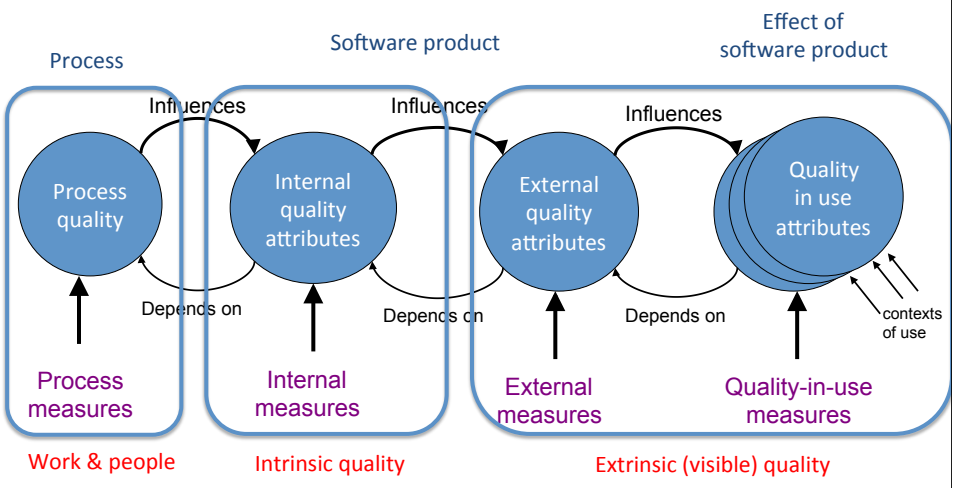
Taxonomy of Quality Attributes

- related to Product operation
  - Correctness Does it do what I want?
  - Reliability Does it do it all of the time? (see Robustness)
  - Efficiency Does it do it fast and cheap? (performance)
  - Usability Can we all run it, operate it?
  - Integrity Is it secure?
- related to Product revision
  - Maintainability can I fix it?
  - Testability can I test it? (=Verifiability)
  - Flexibility can I change it?
- related to Product transition
  - Portability can I use it somewhere else (CPU, OS)?
  - Reusability can I scavenge the software for product Z?
  - Interoperability will it work with X, Y, Z?

Alternate taxonomy: Grady's F URPS

- F Functionality
- U Usability
- R Reliability
- P Performance
- S Supportability
- yet another classification in Std: ISO 9126

The ISO 9126 Quality Model



ISO 9126 Quality model: internal & external

FRUEMP	
Functionality	Does it satisfy the users' needs?
Reliability	Can the software maintain its level of performance ?
Usability	How easy is it to use?
Efficiency	Relates to the physical resources used during execution?
Maintainability	Relates to the effort needed to make changes to the software?
Portability	How easily can it be moved to a new environment?

ISO 9126: Functionality

- Suitability
- Accuracy
- Interoperability
- Security

- *Functionality compliance*

ISO 9126: Reliability

- Maturity
- Fault-tolerance
- Recoverability

- *Reliability compliance*

ISO 9126: Usability

- Understandability
- Learnability
- Operability
- Attractiveness

- *Usability compliance*

ISO 9126: Efficiency

- Time behaviour
- Resource utilisation

- *Efficiency compliance*

ISO 9126: Portability

- Adaptability
- Installability
- Co-existence
- Replaceability

- *Portability compliance*

ISO 9126: Maintainability

- Analysability
- Changeability
- Stability
- Testability

- *Maintainability compliance*

ISO 9126: Quality in use

For specified users in a specific context:

- Effectiveness
- Productivity
- Safety
- Satisfaction

Example 1: recoverability metrics

- Availability: how available is the system for use during a specific period of time
  - X = To / (To + Tr) 0<=X<=1, closest to 1 is best
    - To = operation time, Tr = time to repair
  - Y = A1 / A2
    - A1 = # of cases of successful software use
    - A2 = number of cases of attempted use
- Mean down time:
  - Z = T / N, 0 < Z, the smaller the better
    - T = total down time, N number of observed breakdowns
- *Mean recovery time,*
- *Restorability, restartability,*
- *Restore effectiveness, etc. …*

Example 2: External understandability

- ….
- Demonstration effectiveness: What proportion of functions can the user operate successfully after a demonstration or tutorial?
  - Method: user observation, possibly with video camera
  - X = A / B
    - A number of functions operated successfully
    - B = number of demos/tutorial accessed
- Ease of learning how to perform a task in use: how long does the user take to learn how to operate a function efficiently?

More definitions: QC and QA

- Quality Control (QC)
  - The means by which we are going to evaluate (qualitatively or quantitatively) the Quality
  - reviews, test, surveys, statistical control, etc.
- Quality Assurance (QA)
  - QA provides management with the necessary data to be informed about product quality, to gain insight and confidence that quality goals will be met
  - Including identification of problems affecting quality

Cost of Quality

- Prevention costs
  - Quality planning
  - Formal reviews
  - Test equipments
  - Training
- Appraisal costs
  - Testing
- Internal failure cost
  - Rework
  - Repair

Cost of Quality

- External failure costs
  - Complaint resolution
  - Product return and replacement
  - Help line support
  - Warranty work
  - Loss of future sales ….

Measurement

- How do we measure quality?
- Easier to measure non-quality!
  - Problems, defects
  - Non conformance (to specs or standards)
  - Delays, cost overrun
- Return on investment?
- Customer satisfaction?

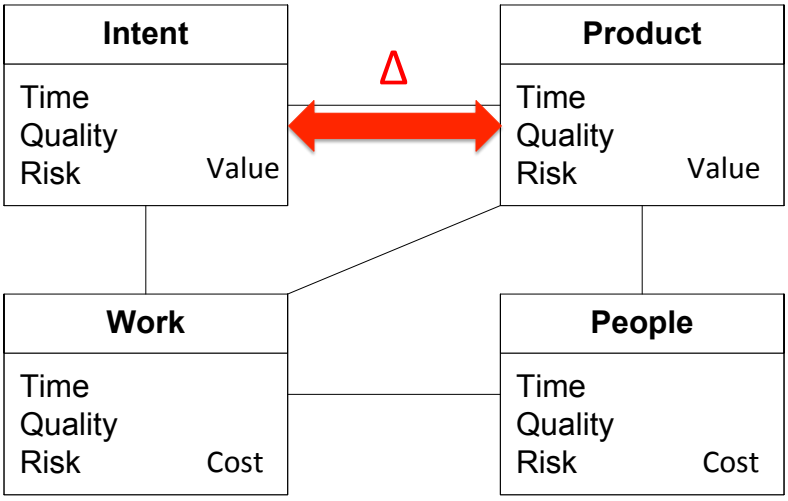
Conformance to Standards (Compliance)

- Not an assurance of success
- But it does help in some context
- Standards have captured some good practice, some common wisdom.
- They provide more objectives ways to assess a situation
  - especially when multiple parties are involved

Key concept: Defect

- A defect is something that causes the software to behave in a manner that is inconsistent with the requirements or the needs of the user (or customer).
- An anomaly, or flaw, in a delivered work product. Examples include : omissions and imperfections found during early lifecycle phases and symptoms of faults contained in software sufficiently mature for test or operation. A defect can be any kind of issue you want tracked and resolved

Defects: the gap between Intent and Product



Defects

- We can either:

1. Avoid inserting defects in the system

2. Remove the defects we have (nevertheless) inserted

Avoid inserting defects in the system

- Excellent process
- Automation of tedious or error prone tasks
- Highly educated people
- Very sophisticated tools

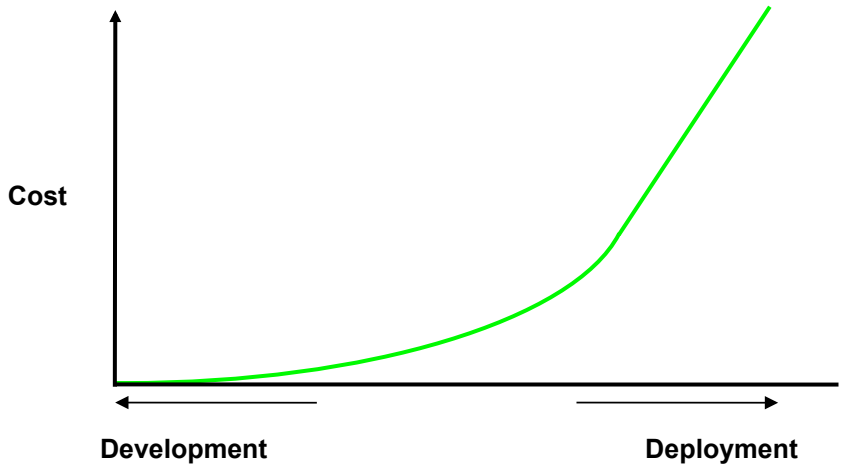
- Simple systems (?)

- Example: Cleanroom process (Harlan Mills)

Removing defects from the system

- The earlier we remove them, the better, and the cheaper.
- Reviews, inspections, audits, ….
- Testing, debugging

Cost of fixing defects



Quantitative vs. procedural approach

- We can ask: "Have we done all the right things?"
- Or we can estimate how many defects there are left
  - but can we?
  - There are models for software reliability
  - But they are not very reliable ….!

Defect Removal Efficiency

- Percentage of existing total defects detected by a certain class of quality control activities, in some part of the lifecycle.

Quality Management PDCA

- Plan
- Do
- Check
- Act

- at least once per iteration

Quality Management: Planning (P)

- Setting quality objectives
- On the Product:
  - Selecting key measures for specific quality attributes
    - Performance, etc.
    - Code quality
    - Customer Satisfaction
  - Monitoring defects:
    - Quantity of remaining defects
    - Type of defects
- on the Process:
  - Productivity (velocity)
  - Test coverage
  - Defect removal efficiency
  - Schedule
  - Resource consumption

D = Do

- Do reviews
- Conduct test campaigns, customer surveys, …
- Iterative development helps
  - You can organize multiple waves of testing
  - See trends

C = Check

- Gather information on effectiveness of
  - reviews
  - tests
- Compare quality data collected with objectives
  - Identify discrepancies
  - investigate cause of discrepancy

A = Act

- Improve process
  - Guidelines: programming, design, UI
  - Checklists
- Tooling
- Training
- Replanning or resetting expectations

Managing defects

- Problem reports
- Defects
- Defects attributes:
  - severity
  - cause (and likelihood)
  - consequence
  - *workaround*

Severity

- A simple scale is sufficient

1: major defect ("show stopper")
  - System crashes
  - Key functions not usable

2: serious user annoyance
  - some functions not usable

3: Minor annoyance

4: Request for enhancement

Severity (cont.)

- Criteria must be refined for each project
- Context is different
- Sensitivity is different

- Clear definitions avoid lots of late arguing when things get tough.

- IEEE 1044 (?) "anomalies"

Defects Must Be Actively Managed!

- Use a simple database
- Spreadsheet for small projects

- Bugzilla
- Rational ClearQuest
- Jira
- Trak
- Microsoft DDTS
- etc….

Monitoring defects

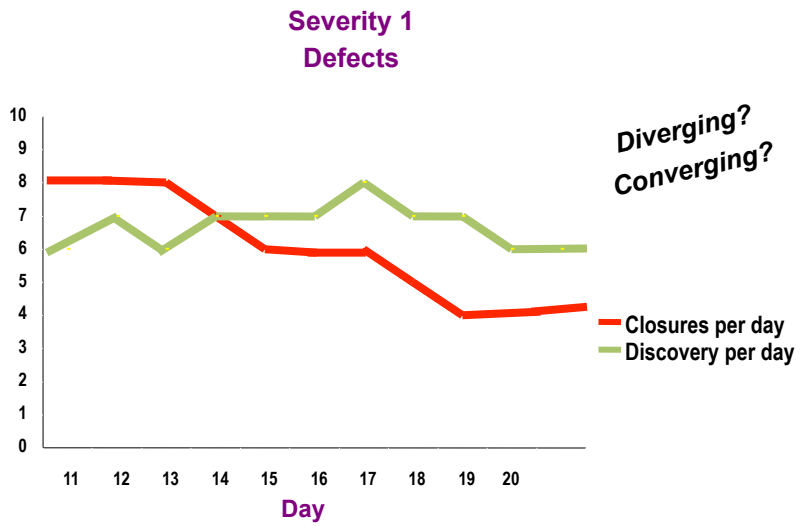
- Track weekly at least:

1. Defects found (by severity)
2. Defects closed (by severity)

- and trend (rate over time)

- Additionally
  - Defect density for major subsystems
  - Instability of some subsystems

Tracking Defects, day after day



Jan. 2013 53

ODC

Opener Section (These attributes are usually available when the defect is opened.)			Closer Section (These attributes are usually available when the defect is fixed.)				
Defect Removal Activities	Triggers	Impact	Target	Defect Type	Qualifer	Age	Source
Design Rev, Code Inspection, Unit test, Function Test, System Test.	1. Design Conformance 2. Logic/ Flow 3. Backward Compatibility 4. Lateral Compatibility 5. Concurrency 6. Internal Document 7. Language Dependency 8. Side Effect 9. Rare Situations 10. Simple Path 11. Complex Path 12. Coverage 13. Variation 14. Sequencing 15. Interaction 16. Workload/Stress 17. Recovery/Exception 18. Startup/Restart 19. Hardware Configuration 20. Software Configuration 21. Blocked Test (previously Normal Mode)	1. Installability 2. Serviceability 3. Standards 4. Integrity/Security 5. Migration 6. Reliability 7. Performance 8. Documentation 9. Requirements 10. Maintenance 11. Usability 12. Accessibility 13. Capability	Design/Code	1. Assign/Init 2. Checking 3. Alg/Method 4. Func/Class/Object 5. Timing/Serial 6. Interface/O-O Messages 7. Relationship	1. Missing 2. Incorrect 3. Extraneous	1. Base 2. New 3. Rewritten 4. ReFixed	1. Developed In-House 2. Reused From Library 3. Outsourced 4. Ported

Jan. 2013 54

Defect Removal Activities	Triggers	Impact
Design Rev, Code Inspection, Unit test, Function Test, System Test.	1. Design Conformance 2. Logic/ Flow 3. Backward Compatibility 4. Lateral Compatibility 5. Concurrency 6. Internal Document 7. Language Dependency 8. Side Effect 9. Rare Situations 10. Simple Path 11. Complex Path 12. Coverage 13. Variation 14. Sequencing 15. Interaction 16. Workload/Stress 17. Recovery/Exception 18. Startup/Restart 19. Hardware Configuration 20. Software Configuration 21. Blocked Test (previously Normal Mode)	1. Installability 2. Serviceability 3. Standards 4. Integrity/Security 5. Migration 6. Reliability 7. Performance 8. Documentation 9. Requirements 10. Maintenance 11. Usability 12. Accessibility 13. Capability

Jan. 2013 55

Target	Defect Type	Qualifer	Age	Source
Design/Code	1. Assign/Init 2. Checking 3. Alg/Method 4. Func/Class/Object 5. Timing/Serial 6. Interface/O-O Messages 7. Relationship	1. Missing 2. Incorrect 3. Extraneous	1. Base 2. New 3. Rewritten 4. ReFixed	1. Developed In-House 2. Reused From Library 3. Outsourced 4. Ported

Jan. 2013 56

Defects Database: What to track? Why?

- Exercise:
  - what attributes should we track for a defect and why?

Am I ready to "ship"?

- When?

  Absolute level of quality?
  Efficiency of defect finding/fixing is getting low?

- Estimating residual defects:
  - Trend (rate of discovery and correction)
  - Defect density: defects / KSLOC
  - Defect pooling
  - Defect seeding
- Warning: only works with large systems (statistical effects)

Technique: Defect Density

- OrcaSys V1: 100 KSLOC
  - 650 defect before release
  - 50 after release
- OrcaSys V2: added 50K SLOC
  - 400 defect before release
  - 75 after release
- OrcaSys V3: added another 100KSLOC
  - detected 600 defect so far
  - Quality objective is 95%
  - are you ready to ship?

Source: S.McConnell

Technique: Defect Density (cont)

- OrcaSys V1: 100 KSLOC
  - 700 defect s=> density = 7 defect / KSLOC
- OrcaSys V2: added 50K SLOC
  - 475 defects => density >9.5 defect/KSLOC
- OrcaSys V3:
  - with a defect density of 7 to 10 you are looking at a total of 700 to 1000 defects in version 3.
  - To find 95% you are aiming at 650 to 950 defects before release
  - you have found only 600 … still some ways to go.

Source: S.McConnell

Technique: Defect Pooling

- Create 2 pools of defects (A and B)

$Defects_{Unique} = Defects_A + Defects_B - Defects_{A\&B}$

$Defects_{Total} = (Defects_A \times Defects_B) / Defects_{A\&B}$

Technique: Defect Pooling (cont.)

- OrcaSys V3:
  - Pool A: entered by people whose initial is A-K
  - = 400 defects
  - Pool B: entered by people whose initial is L-Z
  - = 350 defects
  - Common defects (exist in A and B): 150
- Unique defects: 400 + 350 – 150 = 600
- Estimate of total defects:
  - (400 x 350) / 150 = 933

Technique: Defect Seeding

- Infect the software with a sample of representative defects
- Check the defects found and the proportion of seeded versus indigenous

$IndigenousDefects_{Total}$ =
($SeededDefects_{Planted}$ / $SeededDefects_{Found}$) *
$IndigenousDefects_{Found}$

Technique: Defect Seeding (cont.)

- OrcaSys V3:
  - You plant 50
  - 31 of the 50 have been found on top of 600 indigenous defect

You are probably looking at
50 / 31 * 600 = 967 defect total

Combine techniques

- OrcaSys V3:
  - seeding: 967
  - Pooling: 933
  - defect density: 650 - 950
  - trend: 800
- Precision is not important: with 600 so far you are not ready to deliver.

Technique: Review

- Requirements review
- Design reviews
- Code reviews
- How to organize a review?
- How effective are reviews?
- Alternatives?
- Reviews, inspections, walkthroughs, audits

Technique: Review (cont.)

- Fagan inspections
  - Moderator
  - Two inspectors or readers
  - Authors
- Identify potential faults
- Use some checklist

Reviews?

- Integral part of CMM level 3 and up
- Effectiveness not clear
- Drawback:
  - Time-consuming: meeting
  - Shallow
  - Morale
- Advantages:
  - Morale
  - Mentoring effect

Technique: Pair Programming

- How to organize it?
- Advantages
- Drawbacks
- Effectiveness and efficiency still being debated

Other things to know about SW Quality

- Surveys
- Measurement
- GQM = Goal Question Metric
- TQM = Total Quality Management
- Kaizen, etc.
- ISO 9000 Quality Standards
- Six Sigma movement

More quotes

- I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of meager and unsatisfactory kind. (Lord Kelvin, 1883)
- The difficulty in defining quality is to translate future needs into measurable characteristics, so that a product can be designed and turned out to give satisfaction at a price the user will pay. (Shewhart).

Measurement

- Halstead, and the "software science" (1970's)
  - Code complexity
- McCabe and the Cyclomatic number:
  - C = e – n + p
- Later: OO measures
- Many things to measure, but little net results on quality
  - Automate is not the answer

Goal Question Metric approach

- Vic Basili, U. of Maryland, 1985-1992
- A fresh wind of sanity
- Asking "why?"
  - Why do we measure?
  - How do we relate measures to quality or other useful goals?

Effective measurement

- Focused on specific goals
- Applied to all life-cycle product, processes and resources
- Interpreted based on characterization and understanding of the organizational context, environment and goals

GQM

- Goal: Conceptual level
  - Object of measurement (product, process, resources) point of view, quality attribute
- Question: Operational Level
  - Characterize the object relative to a quality attribute
  - relate it to the goal
- Metric: Quantitative level
  - set of data associated with the question

GQM : Example

- Goal:
  Improve the timeliness of change request processing from the Project manager viewpoint
  - Purpose: improve
  - Issue: timeliness
  - Object: change request (process)
  - Viewpoint: project manager

GQM Example (cont.)

- Question 1: What is the current change request speed
- Metrics 1:
  - average cycle time
  - standard deviation
  - % above upper limit
- Question 2: Is the performance improving?
- Metrics 2:
  - current average cycle time/ baseline average cycle time
  - Subjective rating of manager's satisfaction

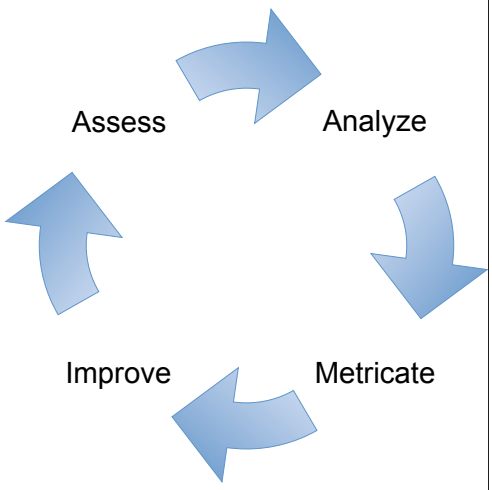
GQM: Example

- More questions:
  - Is the change management process documented?
  - Is the actual change process the one that is documented?

The ami approach

- Kevin Pulford, Annie Kuntzmann-Combelles, et al.
- European Esprit project AMI

- Expand on the original GQM
- Detailed step-by-step process



Sounds like Deming again….

The ami approach

- Assess
  1. assess your environment
  2. define primary management goals
     - knowledge goals
     - change or achievement goals
  3. check the goals against the assessment

  – Inputs: plans, business goals, audits, previous measurements
  – Support: assessment framework (CMM, SPICE, ISO 9000)

The ami approach (cont.)

- Analyze
  4. Break down management goals into subgoals
     - identify entities and attributes
  5. check consistency of resulting goal-tree
  6. Identify metrics from subgoals
     - qualitative, quantitative?
     - Something that can be measured: # of defect, # lines, of code, etc…

  – Input: goals
  – Support: templates, procedures, tools

The ami approach (cont.)

- Metricate
  7. Write and validate a measurement plan
     Objectives
     Metrics definition, collection procedure
     Primitive metric, metric
     Responsibilities, time scales
  8. Collect primitive data
  9. Verify primary data

  – Input: goals, questions, metrics,
  – Support: templates, procedures, tools

The ami approach (cont.)

- Improve
  10. Distribute, analyze and review measurement data
  11. Validate the metrics
  12. Relate data to the goals and implementation actions

  – Inputs: Measurement plan and collected data
  – Support: guidelines for presenting data

Metric definition template

- Name
- Definition
  – If not primitive, how calculated
- Goals
  – Goals and questions that are related to this metric
- Analysis procedure
  – Range, threshold, calibration, standards
  – Models and tools
- Responsibility
  – Who collects, aggregate, analyze, report

Example:

- OrcaSys V1 is not very good…large number of problems reported by the registrar's office.
- Goal 1: gain better understanding of product quality
- Goal 2: improve quality as perceived by user
- Subgoal1: understand how the different part of the system affect quality
- SG2. Gain a measure of the quality of delivered software
- SG3: understand how the different part of the development process affect quality

Example (cont.)

- Metric 3.1: Customer problems traceable to a problem in the requirements specification
- Metric 3.2: same to Design
- metric 3.3: …
- ….
- Metric 3.6 number of defects detected reviews

Example (cont.)

- M6: Error distribution
- Definition: Shows how the defects reported by the customer are attributed to various parts of the process
- Goal: associated to subgoal 2: understand where errors are introduced
- Analysis: metric extracted from Defect database, using attribute: root-cause {req, analysis, design, code, test, doc, other}

Example (cont.)

- OrcaSys V2, week 23

	Severity 1	Severity 2	Severity 3
Req	2	18	34
Analysis	0	2	12
design	6	23	56
Code	4	97	156
Test	0	2	4

Example (cont.)

- OrcaSys V2, week 23

	total	percentage	Year ago
Req	54	9.0%	12.6 %
Analysis	14	2.3 %	8.2 %
Design	85	14.2 %	10.5 %
Code	257	42.8 %	34.1 %
Test	4	<1 %	3.2 %
…	…	…	

Presenting data

- Tables
- Histograms
- Pie charts
- Scatter plots

- Trends (over time) are often more important than absolute values

What to measure? (in general)

- Management indicators
  - Work and progress (over time)
  - Budget cost and expenditures (over time)
  - Staffing and turnover (over time)
- Quality
  - Change traffic and stability (over time)
  - Breakage and modularity (av. breakage per change)
  - Rework and adaptability (av. rework per change)
  - MTBF and maturity (defect rate over time)

Walker Royce, 1999

RUP: try to focus on:

- Modularity:
  - Scrap and rework are localized
- Adaptability:
  - Rework (effort for resolution)
- Maturity
  - Defect frequency, over time
- Maintainability:
  - productivity in maintenance relative to development

Source: IBM 2003 & Walker Royce 1999

Collected data (primitive metrics)

- Total SLOC
- Total effort
- Configured SLOC
- Software change orders (SCO) of different level
- Open rework: B=SLOC broken by SCO
- Closed rework: F=cumulative fixed SLOC
- Rework effort: E
- Usage time

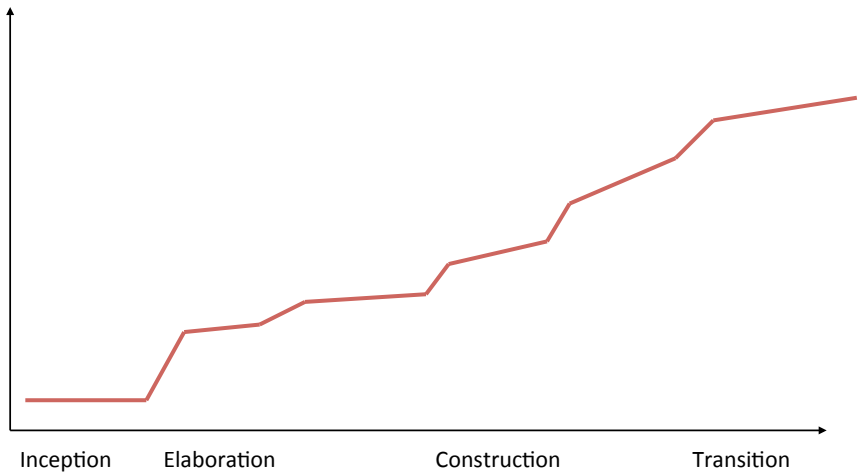
Software Change Order vs. Defect

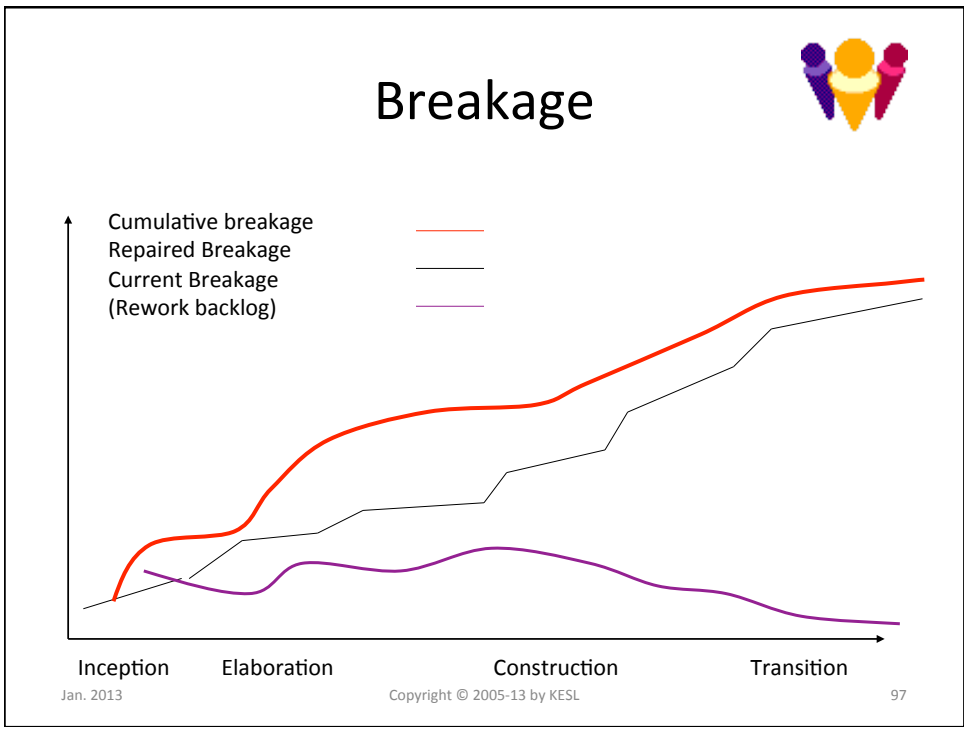
- Software change order
  0. Critical
  1. Important, non critical
  2. Minor and improvements
  3. Change in requirements, new features

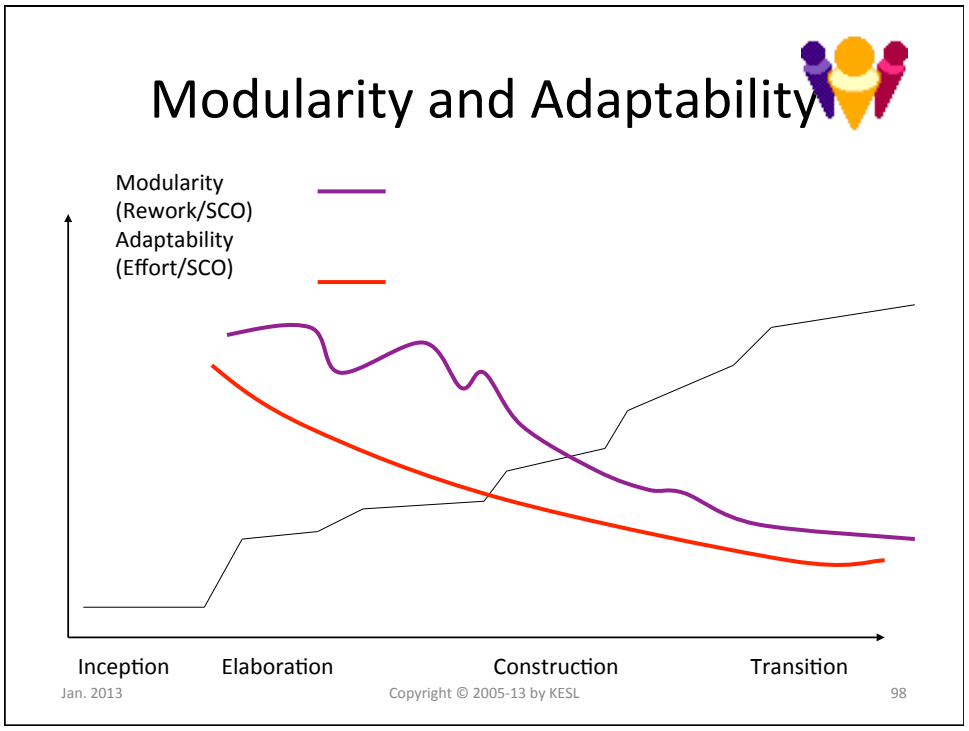
Metrics

- scrap ratio: B/ SLOCt
- Rework ratio: B/Effort
- Modularity: B/N (average breakage per SCO)
- Adaptability: E/N (average effort per SCO)
- Maturity: UT/(SCO0+SCO1) = meantime between defect
- Maintainability: (scrap ratio)/(rework ratio)

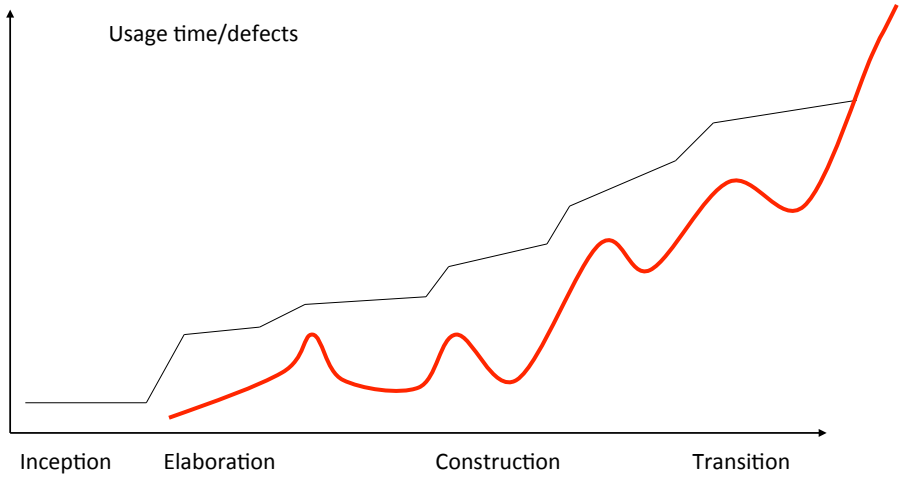
Total SLOC




Breakage
Cumulative breakage
Repaired Breakage
Current Breakage
(Rework backlog)
Inception
Elaboration
Construction
Transition
Jan. 2013
Copyright © 2005-13 by KESL
97


Modularity and Adaptability
Modularity
(Rework/SCO)
Adaptability
(Effort/SCO)
Inception
Elaboration
Construction
Transition
Jan. 2013
Copyright © 2005-13 by KESL
98

Maturity



(Too?) Many Standards on SW Quality

- IEEE 730: Software Quality Assurance Plan
  - IEEE 730.1 Guide for SQA Planning
- IEEE 982: Measures for reliable software
- IEEE 1044: Classification of anomalies
- IEEE 1061: Software Quality Metrics Methodology
- ISO 9000 family
- And now : ISO25000 series

IEEE 730: Software Quality Assurance Plan

1. Purpose
2. References
3. Management
4. Documentation
5. Standards, practices, conventions, metrics
6. Reviews and audits
7. Test
8. Problem reporting and corrective actions
9. Tools, techniques, methodologies
10. Code control
11. Media control
12. Supplier control
13. Record collection
14. Training
15. Risk management

Most relevant is ISO 9000-3

- Guidelines for Applying ISO 9001 1994 to Computer Software
  - Use ISO 9000-3 if you develop, supply, install, or maintain computer software
  - Paraphrases 9001 (but hard to read ☹)
- Main messages of 9000-3:
  - Manage quality proactively
  - Develop a quality system and document it
    - Plans, procedures, reviews, metrics, ….
  - Collect and manage data and information relative to the application of the quality system

Old and current fads and buzzwords

- TQM: Total Quality Management
  - 1950's, Deming (again)
- QFD : Quality Function Deployment
  - House of Quality
  - Originated in Japan: Yoji Akao, circa 1972
  - not software specific
- Six Sigma
  - Manufacturing
  - Applicability to software highly dubious

TQM Philosophy, in 14 steps

1. Create constancy of purpose for improvement of product and service. Constancy of purpose requires innovation, investment in research and education, continuous improvement of product and service, maintenance of equipment, furniture and fixtures, and new aids to production.
2. Adopt the new philosophy. Management must undergo a transformation and begin to believe in quality products and services.
3. Cease dependence on mass inspection. Inspect products and services only enough to be able to identify ways to improve the process.
4. End the practice of awarding business on price tag alone. The lowest priced goods are not always the highest quality; choose a supplier based on its record of improvement and then make a long-term commitment to it.

TQM

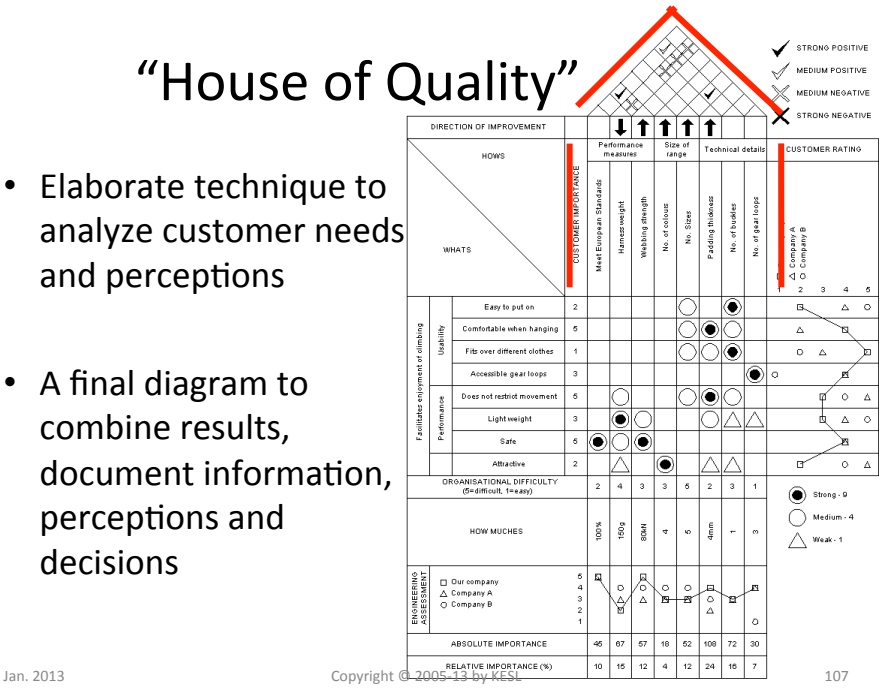
5. Improve constantly and forever the system of product and service. Improvement is not a one-time effort; management is responsible for leading the organization into the practice of continual improvement in quality and productivity.
6. Institute training and retraining. Workers need to know how to do their jobs correctly even if they need to learn new skills.
7. Institute leadership. Leadership is the job of management. Managers have the responsibility to discover the barriers that prevent staff from taking pride in what they do. The staff will know what those barriers are.
8. Drive out fear. People often fear reprisal if they "make waves" at work. Managers need to create an environment where workers can express concerns with confidence.
9. Break down barriers between staff areas. Managers should promote teamwork by helping staff in different areas/departments work together. Fostering interrelationships among departments encourages higher quality decision-making.

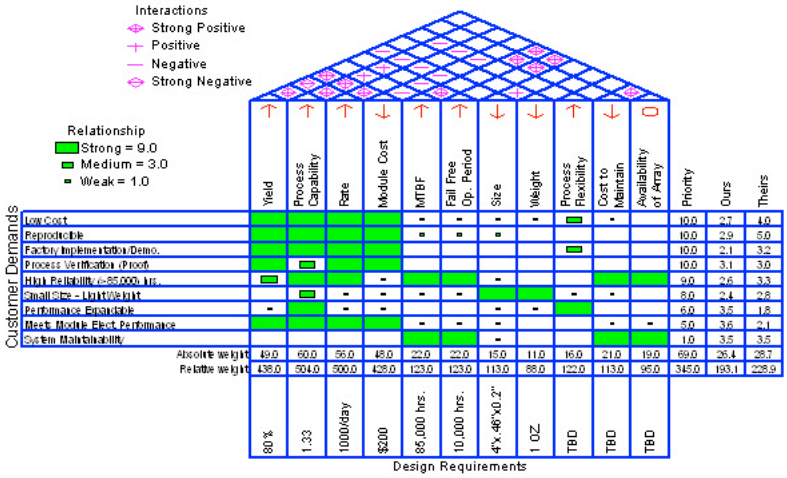
TQM

10. Eliminate slogans, exhortations, and targets for the workforce. Using slogans alone, without an investigation into the processes of the workplace, can be offensive to workers because they imply that a better job could be done. Managers need to learn real ways of motivating people in their organizations.
11. Eliminate numerical quotas. Quotas impede quality more than any other working condition; they leave no room for improvement. Workers need the flexibility to give customers the level of service they need.
12. Remove barriers to pride of workmanship. Give workers respect and feedback about how they are doing their jobs.
13. Institute a vigorous program of education and retraining. With continuous improvement, job descriptions will change. As a result, employees need to be educated and retrained so they will be successful at new job responsibilities.
14. Take action to accomplish the transformation. Management must work as a team to carry out the previous 13 steps.

"House of Quality"

- Elaborate technique to analyze customer needs and perceptions
- A final diagram to combine results, document information, perceptions and decisions



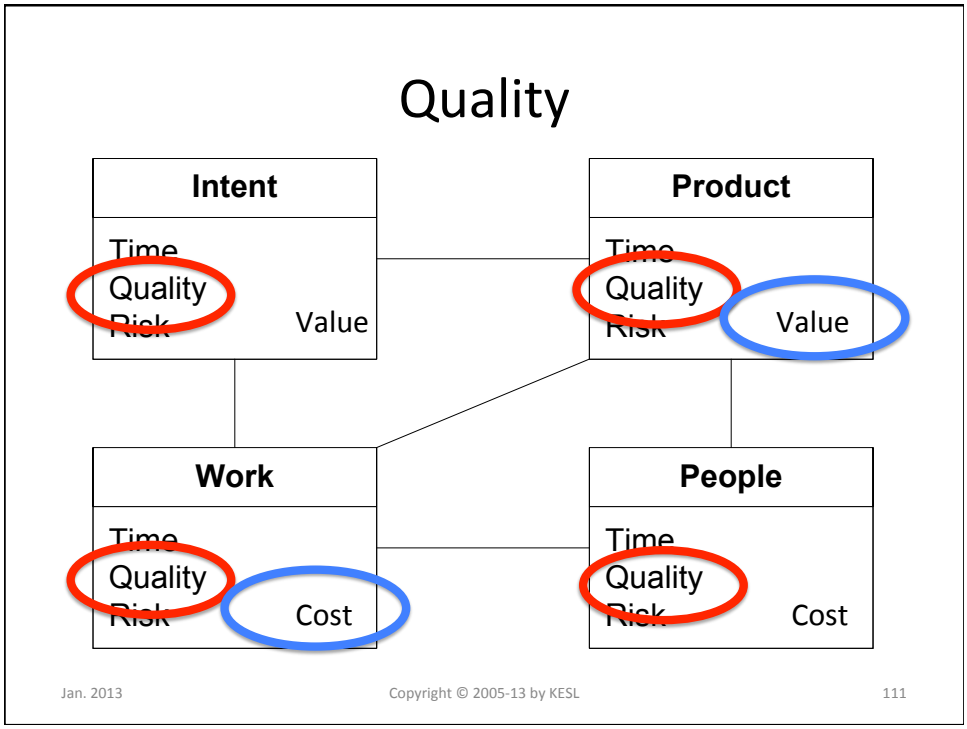


Summary

- Software quality
  - Many dimensions & attributes
  - What is "good enough"?
- Measurements
  - many proposed metrics, few are easy to correlate to quality
  - "software science" not here yet
- Goal Question Metric:
  - a top down approach
- Many standards:
  - IEEE 730
  - ISO 9000-3

Orthogonal Defect Classification (ODC)

- IBM, circa 1992, Ram Chillarege
- ODC captures and turns semantic information in the software defect stream into a measurement on the process.


Quality
Intent
Time
Quality
Risk
Value
Product
Time
Quality
Risk
Value
Work
Time
Quality
Risk
Cost
People
Time
Quality
Risk
Cost
Jan. 2013
Copyright © 2005-13 by KESL
111

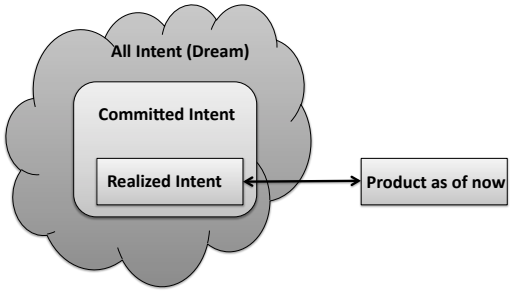
09: Managing Objectives and Scope

Software Project Management
Philippe Kruchten

Module outline

- Scope
- Requirement management
- Release and iteration planning
- Cost and value
- Prioritization
- Tracking progress

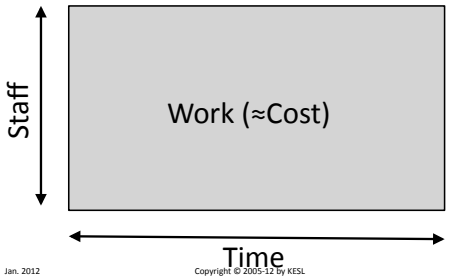
Intent under Control



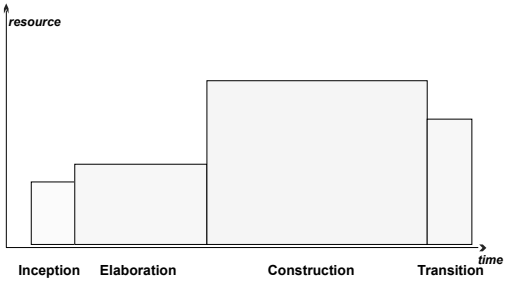
Time-box

  DSDM 1995

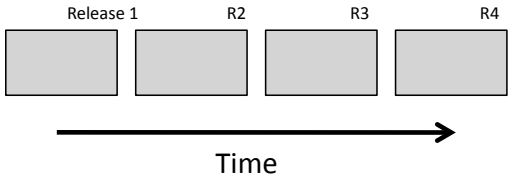
Time-box



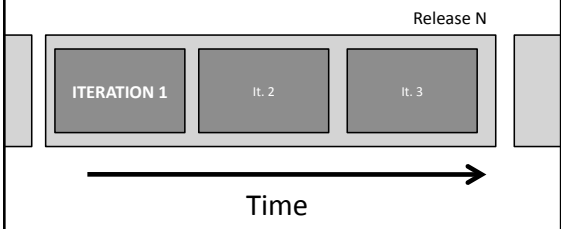
More realistically



Time-boxes: Releases

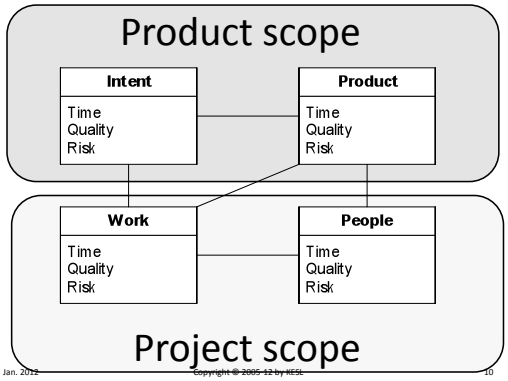


Time-boxes: Iterations (sprints)



Scope

- Product scope
  - Features and functions that characterize the product
- Project scope
  - Work that must be done in order to deliver the product



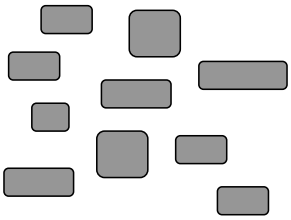
Scope Management

- Let us do all the work required, and only the work required.
- In large projects, it is easy to get lost in the details, to get side-tracked, to confuse the immediate short-term objectives with the long term ones.

Scope management

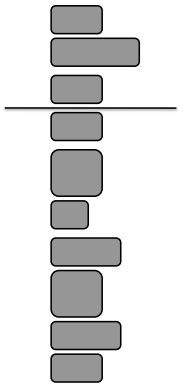
- Intersection of:
  - Project management
  - Requirements management
- Provides a way to decide: are we going to miss the objectives?

Features

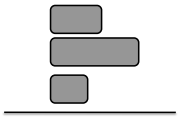


Intent

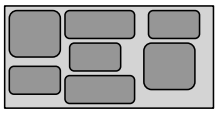
Features



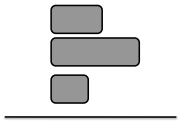
Features



Rn

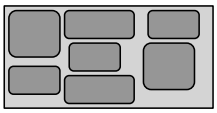


Features

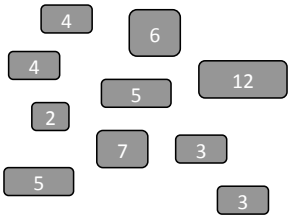


?

Rn



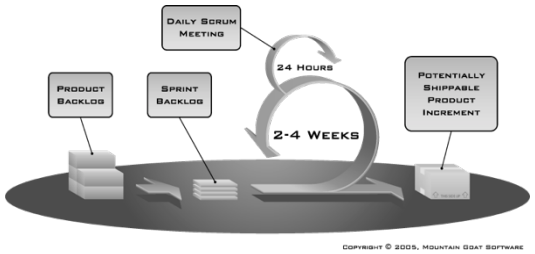
Features & Value



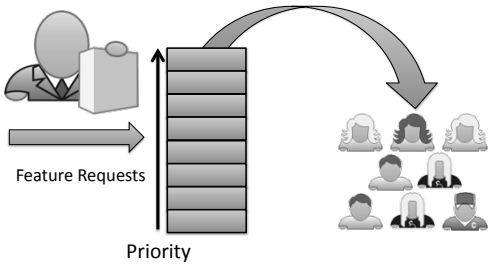
Maximizing value

Highest value first
Ignore time

Backlog

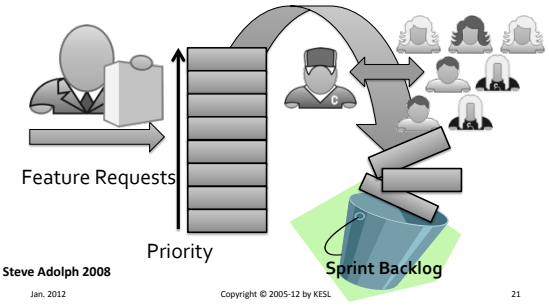


The Backlog



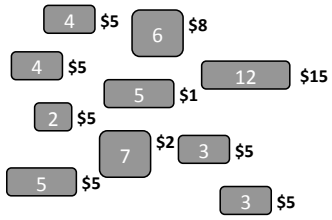
Steve Adolph 2008

Sprint Planning



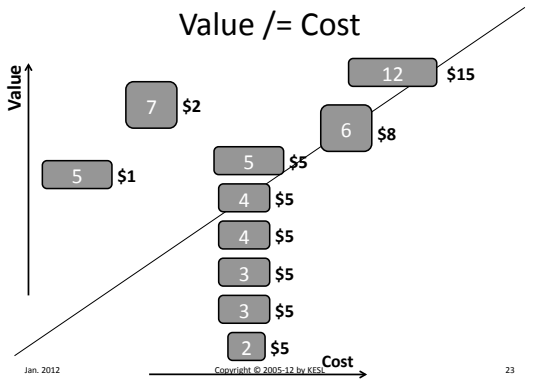
Steve Adolph 2008

Value = Cost?



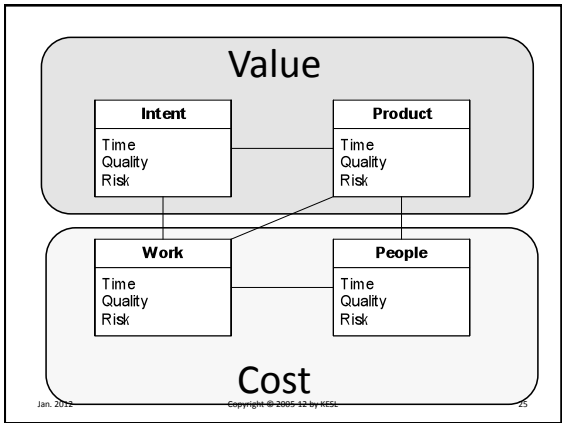
Only for simplest cases

Value /= Cost



Value and Cost

- Value: to the business (the users, the customers, the public, etc.)
- Cost: to design, develop, manufacture, deploy, maintain
- Simple system, stable architecture, many small features:
  - Statistically value aligns to cost
- Large, complex, novel systems ?



Efficiency *vs.* Effectiveness

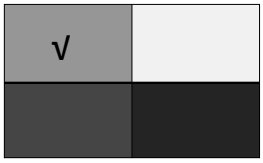
Efficiency

- relationship between the output in terms of goods, services or other results and the resources used to produce them

Effectiveness

- relationship between the intended impact and the actual impact of an activity

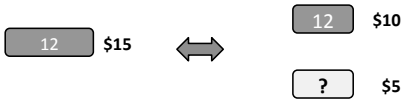
Cost **Value**



What colour is your backlog?

(so far)

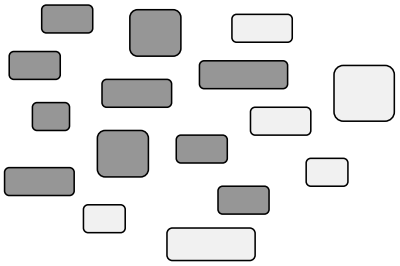
Invisible Features



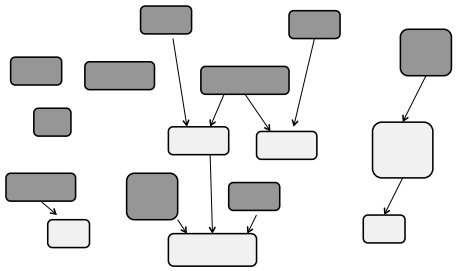
Invisible Features

- Architecture
- Infrastructure
- Common elements
- Framework
- Libraries
- Reuse
- DSL
- *Product line*

Features

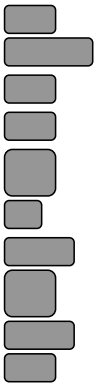


Dependencies



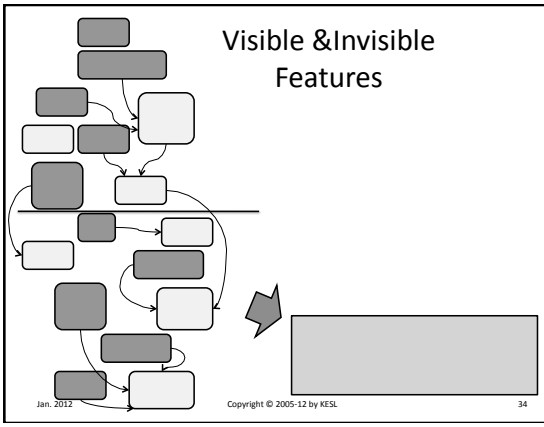
Release Planning

- Time-box = budget
- Fill the time-box with a combination of visible and invisible features
- … while maximizing value
- Product manager: maximize value (green stuff)
- Project manager: maximize budget utilization
- Techie: maximize the fun stuff (yellow) ?



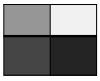
Features

Visible &Invisible Features



What colour is your backlog?

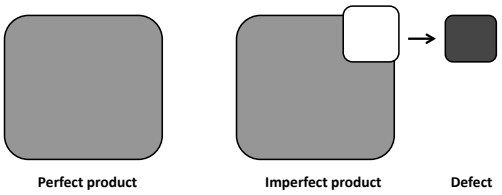
(so far)



Defects

- Defect = Feature with negative value
- Fix (defect) has a positive cost (work)
- Time/place of discovery
  - Inside development (in-house, in process)
  - Outside development (out-house?) in a released product (escaped defects)

Escaped Defect has Value

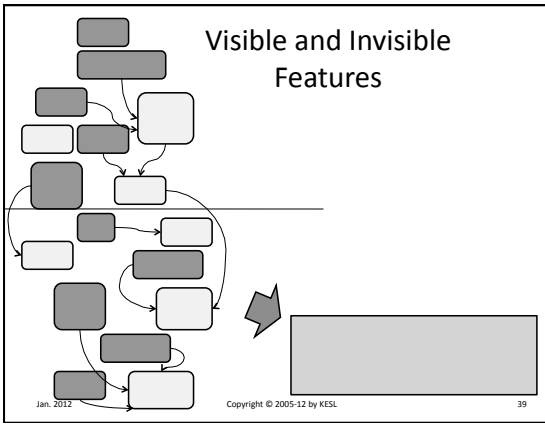


Fixing a Defect has a Cost

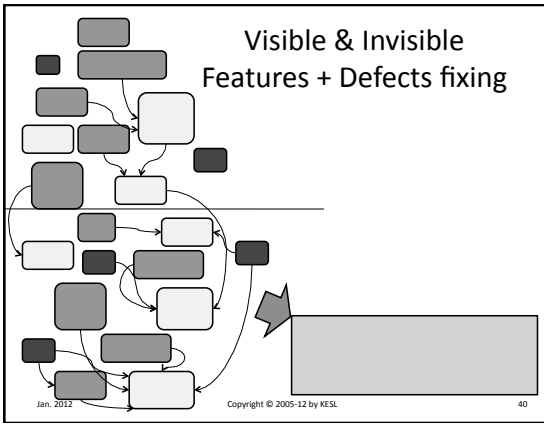
- Defects have both value and cost
- Value of fixing a defect = −Value of the defect
- Cost of fixing a defect (estimated)

- Defects have dependencies
  - Defect fix depend on invisible feature
  - Visible feature depending on a fix

Visible and Invisible Features



Visible & Invisible Features + Defects fixing



Project scope management

- Initiation
- Planning
- Definition
- Verification
- Change control

Source: PMBOK

Initiation

- Get to the point where you actually launch the project, decide to go ahead.
- Inputs: one or several from many different kinds
  - market demand
  - business need
  - customer request
  - technological advance
  - new legal requirements
- Output:
  - Draft Project charter or Project vision, Project manager appointed
  - Constraints and assumptions

Initiation

- How to decide what makes sense?
  - Decision model
  - Calculation method
- Balancing cost and benefits (value)

- We will see more in Portfolio Management

Scope Planning

- Further description of the product
- Inputs = outputs from Initiation
- Outputs:
  - Draft vision
  - Description, Deliverables, Justifications
  - Objectives = quantifiable criteria to be met to declare "success"

Scope Definition

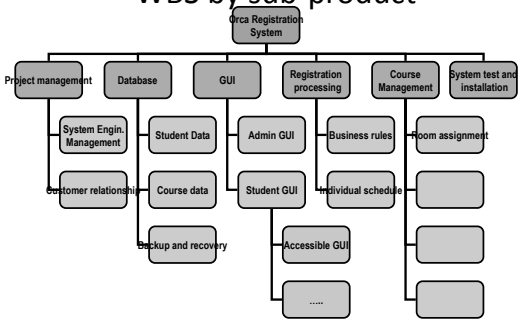
- Identify major deliverables
- Improve accuracy of cost, duration, effort
- Output:
  - Work Breakdown Structure

Work Breakdown Structure (WBS)

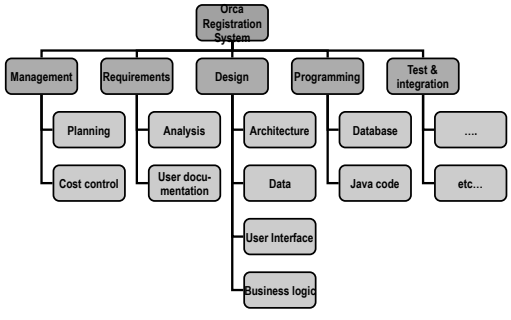
- Decomposition of the work to be performed
- Covers the total scope of the project

- Organized by phase
- By subsystems

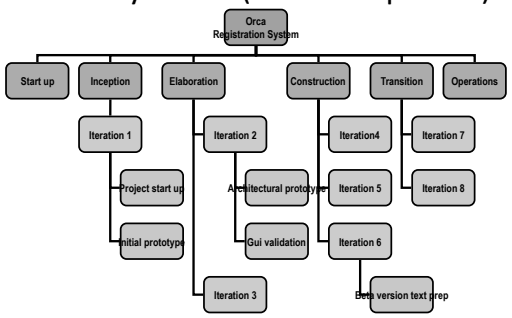
WBS by sub-product



WBS by type of activity



WBS by Phase (with RUP phases)



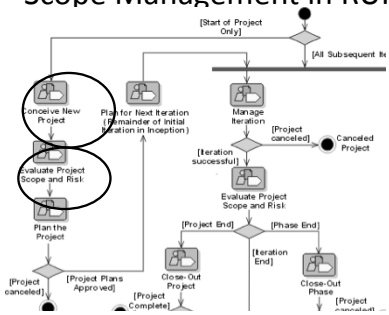
Scope Verification

- Obtain formal acceptance by major stakeholders
  - The project charter (or vision) is right
  - The initial estimates, assumptions, constraints etc are right
- The project is started.

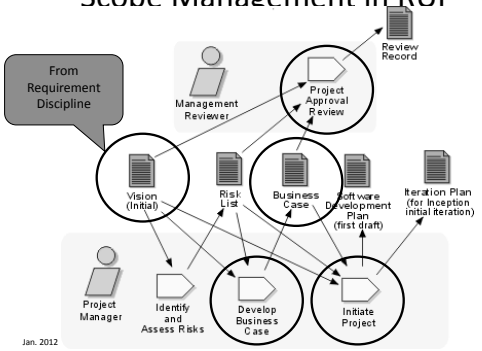
Scope Management

- On-going activity
- Dealing with change requests. E.g.:
  - External event
  - Error or omission in original scope
  - Value-adding change (take advantage of a recent discovery)
  - Implementing a contingency plan to respond to a risk

Scope Management in RUP



Scope Management in RUP



Scope Management in RUP: Artifacts

- Vision Document — A few pages (1 to 12)
  - The "essence" of the product
  - Key needs from the stakeholders
  - Key features of product
  - Defines criteria for success
  - Basis for requirement definition
- Business Case — A few pages (1 to 12)
  - Financial forecast
  - Summary of product (refers to vision)
  - Estimate of expenditures
  - Expected Return on investments
  - Constraints
  - Multiple scenarios (refers to risks)

Roles and Organization

- Project Manager
- Project Review Authority
  - Sr. technical and business management staff
    - VP development
    - Chief architect
    - Director of quality
    - (Future) Project Manager
    - Marketing manager…..
  - Empowered to approve Project Start
  - Empowered to approve changes in scope
    - Vision, Business case, Project plan

Getting a Project Started

- Work simultaneously on multiple artifacts:
  - Vision
  - Business case
  - Risks
  - Project plan
    - Schedule
- Vision and Business Case are "baselined"
  - i.e.., require PRA's approval to be modified

Scope Creep

- Change requests
- Change to requirements that seem to be aligned with original vision but significantly affect cost or schedule, hence business case.
- Re-estimation of the work, the effort, the resources needed, which will affect the business case
- … time to call the PRA !

Scope creep

- "Scope creep is the pejorative name we give to the natural process by which clients discover what they really want." Hal Helms
- Iterative development acknowledges the existence of scope creep. However schedule and budget must have been adjusted to acknowledge this too.

Prioritization

- Classical:
  - High
  - Medium
  - Low
- More realistic
  - Essential
  - Conditional
  - Optional
- DSDM: MoSCoW
  - Must have, should have, could have, would have

Value /= Cost, Value affected by Risk

- Integrating
  - Value
    - from the business / user / customer perspective
    - Benefit
    - Penalty
  - Cost
    - From the development perspective
  - Risk
    - Overall risk, not development and technical risk
    - How much product risk do you want to take

Simple strategy (from K. Wiegers)

- Benefit = relative: scale 1 to 9
- Penalty = relative: scale 1 to 9
- Value = Benefit + Penalty
  - Or: value = b * benefit + p * penalty
- Cost = relative: scale 1 to 9
- Risk = relative: scale 1 to 9
- Priority = value % / (cost % + risk %)
  - Or
- Priority = value % / (c * cost % + r * risk %)

Example

weigh	1	1				1		1		
Feature	**Benefit**	**Penalty**	**Value**	**value %**	**cost**	**cost %**	**risk**	**risk %**	**priority**	**rank**
a	5	3	8	13.3%	4	12.5%	5	20.0%	0.410	5
b	9	3	12	20.0%	8	25.0%	2	8.0%	0.606	1
c	1	9	10	16.7%	6	18.8%	3	12.0%	0.542	3
d	5	7	12	20.0%	2	6.3%	7	28.0%	0.584	2
e	3	2	5	8.3%	5	15.6%	2	8.0%	0.353	6
f	8	5	13	21.7%	7	21.9%	6	24.0%	0.472	4
total			60		32		25			

Monitoring progress and cost

- Earned value system
- (or should it be called Spent-Cost system)

Earned Value

- Management technique to track expenditure, assess whether we are on schedule and budget or if we are drifting.

Concepts:
- Baseline
- Schedule variance
- Cost variance
- Spend comparison
- Indexes

Earned Value: Baseline

Work Unit	A	B	C	D	E	F	Total
Planned value ($)	10	15	10	25	20	20	100

- aka Budgeted Cost of Work Scheduled BCWS
- Derived from a Work Breakdown Structure and some estimates for each element of the decomposition

Earned Value, proper

Work Unit	A	B	C	D	E	F	Total
Planned value ($)	10	15	10	25	20	20	100
Earned value ($)	10	15	10	10	20	--	65

- aka Budgeted Cost of Work Performed BCWP
- What was done, we have "earned"
- If we have done 50% of the work, we earned half of the planned (or budgeted) value

Earned Value: Schedule variance

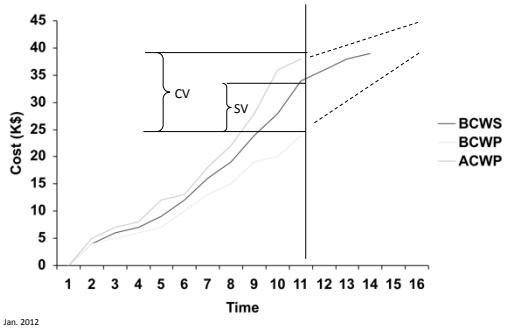
Work Unit	A	B	C	D	E	F	Total
Planned value ($)	10	15	10	25	20	20	100
Earned value ($)	10	15	10	10	20	--	65
Schedule variance	0	0	0	-15	0	-20	-35%

Earned Value: Cost variance

Work Unit	A	B	C	D	E	F	Total
Planned value ($)	10	15	10	25	20	20	100
Earned value ($)	10	15	10	10	20	--	65
Actual Cost	9	22	8	30	22	--	91
Cost Variance	1	-7	2	-20	-2	0	-26 = -40%

- aka Actual Cost of Work Performed ACWP

Graphically



Variances

- Schedule variance

  SV = BCWP - BCWS
- Cost Variance

  CV = BCWP – ACWP
- Need earned value because just monitoring Expenditure versus Budget is meaningless without work actually performed

Earned Value: Cost variance Vs Spend

Work Unit	A	B	C	D	E	F	Total
Planned value ($)	10	15	10	25	20	20	100
Earned value ($)	10	15	10	10	20	--	65
Actual Cost	9	22	8	30	22	--	91
Cost Variance	1	-7	2	-20	-2	0	-26 = -40%
Spend variance	1	-7	2	-5	-2	20	+9 = +9% !!

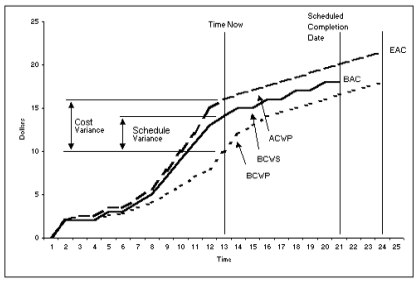
Performance indexes

- Schedule Performance Index

  SPI = BCWP / BCWS
- Ratio of Earned Value and the planned value of completed works. example: SPI = 0.65
- A SPI < 1 is not good
- Cost Performance Index

  CPI = BCWP / ACWP
- Ratio of Earned Value and the actual costs of completed works. example : 0.4
- A CPI < 1 is not good

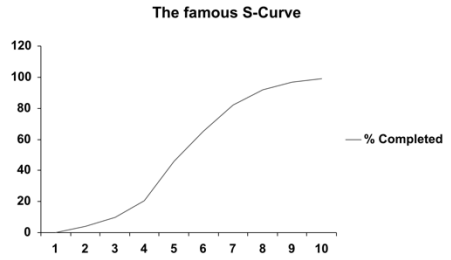
Graphically



Are we on track?

- Estimate at Completion EAC
  EAC = ACWP + ((BAC - BCWP)/CPI)
- The EAC gives an idea of the final costs of a project. It takes into account the original budget at completion (BAC), the Earned Value and the Cost Performance Index of the already completed works.
- Example:
  EAC = 91 + ((100- 65)/0.4) = 178.5

How much should be completed at time T?



Earned Value and Software Projects

- Rarely used
- On large system projects
  - Software only a component
  - Numerous subcontractors
- Earned value depends on having a WBS
- Work Breakdown Structure was already a problem

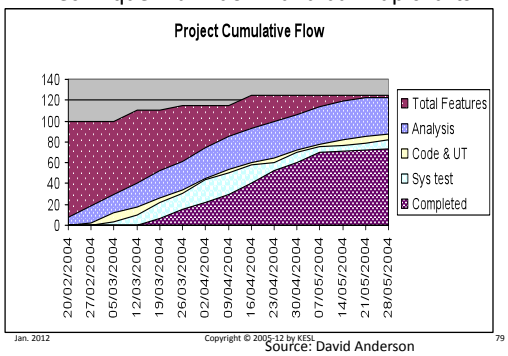
How much should be completed by time T?

- WBS by iteration?
  - Using an S-curve to allocated effort
  - Leads to the trivial and obvious result
- WBS by type of activity?
  - % of design, % of coding, % of testing, etc.
  - Subject to lots of subjectivity
    "80% done 80% left to do" syndrome
- WBS by function (e.g., use case or scenario)?
  - 100 tested -> least controversial

Some hope though

- Use Use cases (or user stories, or story cards) and testable requirements as the work unit.
- Define work estimated
- Adjust estimates
- Close work when UC tested
  - 0% or 100%
  - Somewhat pessimistic, but….
- Drawback: EAC is moving right…
  - but better than no indicator

Technique: Burn-down and burn-up charts



Source: David Anderson

Practice: Daily Stand-up Meeting

- Daily (same time, which time?)
- Standup (12-15 min.)
- Meeting (must attend)
- 3 items:
  - What I have accomplished
  - What I am working on
  - What is blocking me (or worrying me) "Impediments"
- No debate or problem resolution
  - Identify "impediments"

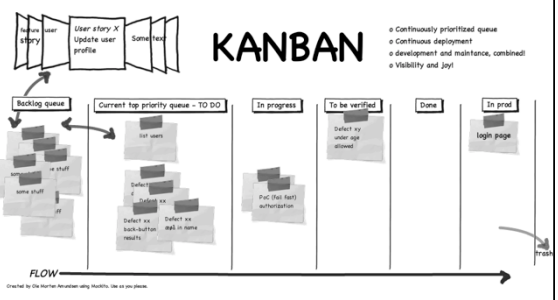
DSM

- Clear roadblocks
- Avoid duplication
- Balance workload
- Buildup shared mental model
- Increase transparency
- Increase accountability

Work-in-progress and "done"

- Starting too many things, not completing them
- Leads to wasted effort, delayed feedback
- No clear definition of completion ("done")

- Technique: kanban board
  - Pull, rather than push
  - Limit work-in-progress

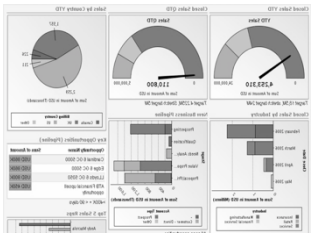


Source: Ole Morten Amundsen

Summary

- Scope
  - Vision
  - Business case
  - "Backlog"
  - Priority
- Scope management
  - Prepare scope data
  - Get approval from PRA
  - Monitor progress ("done")
  - Monitor for scope creep
  - Monitor for cost and budget

On The Project Manager Dashboard



On The Project Manager Dashboard

- Risk List : top 10 this week, by impact
- Defects opened, closed, trend, by severity
- Burn Down chart (Earned business value), based on use-case/scenario/feature/story card "done done" (i.e., tested, documented, etc.)
- Open issues list (by age): TBDs, etc.
- Open positions: TBHs
- .....

10: Managing complexity

Software Project Management
Philippe Kruchten

Module outline

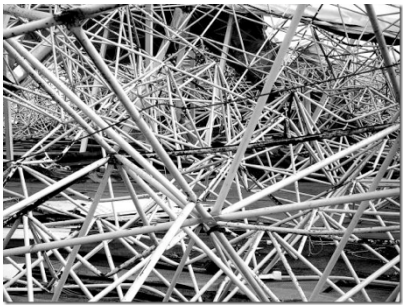
- Complexity and simplicity
- Architecture to the rescue
- Project manager and software architect
- Socio-technical congruence
- Technical debt

"Complexity is the enemy of computer science, and it behooves us, as designers, to minimize it."

Charles Thacker, CACM, July 2010.

"The task of the software development team is to engineer the illusion of simplicity."

Grady Booch, OOAD Book, 1994



Complexity

- What make Systems complex?
  - Scale
    - Many "things"
  - Diversity
    - Many different kinds of "things"
  - Interconnectivity
    - Many "things" connected to many things in different ways
    - Apparent lack of determinism, predictability

Source: McDermid 2000

- **things** = features, developers, stakeholders, users, classes, SLoC, changes, technologies,….

Complexity

- Perceived complexity vs. real complexity
- Intrinsic complexity
  - Scale, Diversity, Interconnectivity
- Extrinsic complexity
  - Embedding in organization
  - Embedding in other larger systems
  - Dependencies, visible and hidden
  - Success (time to market, etc.)
  - Dependability

Source: McDermid 2000

Simplicity

- Parsimony
- Uniformity, regularity
- Clear partition of concerns

- Inverse of complexity?

Occam's Razor in 14th century, Lex parsimoniae

"Make things as simple as possible, but no simpler."

Albert Einstein

KISS

Kelly Johnson

"… perfection is achieved not when there is nothing left to add, but when there is nothing left to take away."

Antoine de St. Exupéry, 1939

Architecture as a partial answer to complexity

- Level of abstraction
- Divide-and-conquer approaches
- Filter for "things"
  - Technologies, requirements, teams, etc…
- Blueprints for other activities

- Congruence (Conway's Law)

Simplicity, driven by architecture

- Parsimony
- Uniformity, regularity
- Clear partition of concerns

- Right level of abstraction
- Modularity, encapsulation

Perception of simplicity

- In the eye of the beholder
- Not perceived uniformly by all stakeholders
  - Education, culture, frequency of interaction, etc.
- Cognitive aspects
- Increase sense of determinism
  - Not chaotic system
- Simplicity = paucity
- Simplicity = limitation
- Simplicity = overcontraints

See: Kurtz Snowden 2003

Towards simple

- Limit number of (visible) things
  - Few technologies, people, interfaces, etc…
- Limit number of (visible) types of things
  - Harder
- Limit interconnections, dependencies
  - Both in numbers and in kinds
- Increase perceived determinism
  - "if I do this, this will happen"
- Engineer the **illusion** of simplicity

Architecture as a partial answer to complexity

- Level of abstraction
- Divide-and-conquer approaches
- Filter for "things"
  - Technologies, requirements, teams, etc…
- Blueprints for other activities
- Congruence (Conway's Law)

Readings

- McDermid, J. A. (2000). *Complexity: concept, causes and control.* Paper presented at the Sixth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2000).
- Maeda, J. (2006). *The laws of simplicity.* Cambridge, MA: MIT Press.
- Cook, R. I. (2000). How complex systems fail. Cognitive Technologies laboratory, University of Chicago.
- Kurtz, C. F., & Snowden, D. J. (2003). The new dynamics of strategy: Sense-making in a complex and complicated world. *IBM Systems Journal, 42*(3), 462-483. (see also Cynefin framework)
- Nielsen, J (1993). *Usability Engineering,* Morgan Kaufman. (see 10 usability guidelines at: http://www.useit.com/papers/heuristic/heuristic_list.html)
- Booch, G. (2004). *The illusion of simplicity* Available from http://www.ibm.com/developerworks/rational/library/2096.html
- Sha, L. (2001). Using Simplicity to Control Complexity. *IEEE Software, 18*(4), 20-28.
- Suh, N.-P. (2005). Complexity in Engineering. *Annals of the CIRP, 54*(2), 46-63.

Business Architecture

- A subset of the enterprise architecture that defines an organization's current and future state, including its strategy, its goals and objectives, the internal environment through a process or functional view, the external environment in which the business operates, and the stakeholders affected by the organization's activities.

BABOK v2 2009

Enterprise Architecture

- Enterprise architecture is a description of an organization's business processes, IT software and hardware, people, operations and projects, and the relationships between them.

Source BABOK v2 2009

Solution architecture

- System architecture
- Software architecture

Software Architecture

Software architecture encompasses the set of significant decisions about

- the organization of a software system,
- the selection of the structural elements and their interfaces by which the system is composed together with their behavior as specified in the collaboration among those elements,
- the composition of these elements into progressively larger subsystems,

Grady Booch, Philippe Kruchten, Rich Reitman, Kurt Bittner; Rational, circa 1995 *(derived from Mary Shaw)*

Software Architecture (cont.)

…

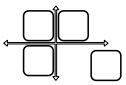
- the architectural style that guides this organization, these elements and their interfaces, their collaborations, and their composition.
- Software architecture is not only concerned with structure and behavior, but also with usage, functionality, performance, resilience, reuse, comprehensibility, economic and technological constraints and tradeoffs, and aesthetics.

Software architecture…

- architecture = { elements, form, rationale } *

Perry & Wolf 1992

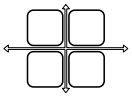
- A skeleton, not the skin
- More than structure
- Embodies or addresses many "ilities"
- Executable, therefore verifiable



ISO/IEC 42010

Architecture: the fundamental concepts or properties of a system in its environment embodied in its elements, their relationships, and in the principles of its design and evolution

A short history of software architecture

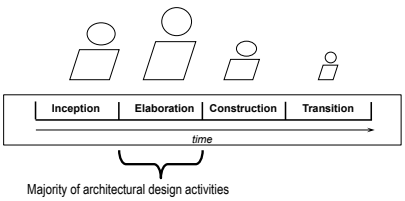


- NATO conference (1969)
- Box & arrows (1960s-1980s)
- Views & viewpoints (1990s-2000)
- ADLs (1980s-2000s)
- Architectural design methods (1990s-2000s)
- Standards, reference architectures (1995-…)
- Architectural design decisions (2004-…)

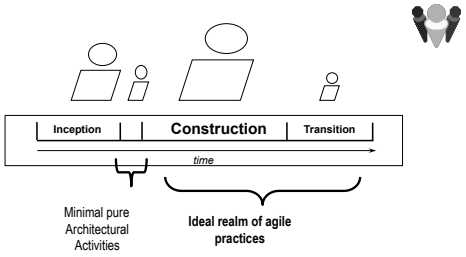
All software-intensive systems have an architecture

- How much effort should you put into it varies greatly
- 75% of the time, the architecture is implicit
  - Choice of technology, platform
  - Still need to understand the architecture
- Novel systems:
  - Much more effort in creating and validating an architecture
- Key drivers are mostly non-functional:
  - Capacity, performance, availability, evolvability, security, …

Big Architectural Effort



Little dedicated architectural effort



Project manager and architect

- Soldered at the hip
- Church and state metaphor
- Hollywood metaphor
  - Producer vs. director

Charter of an Architect or an Architecture Team

- Defining the architecture of the system
- Maintaining the architectural integrity of the system
- Assessing technical risks
- Working out risk mitigation strategies/approaches
- Participation in project planning
- Proposing order and content of development iterations
- Consulting with design, implementation, and integration teams
- Assisting product marketing and future product definitions

Circa 1992, Published in **Kruchten 1999**

Congruence

- Alignment between the technical and the social elements
  - Intent-Work-Product
  - People
- From Dependencies to Coordination
- Use architectural structure to define teams, responsibility, ownership, coordination, communication

Technical Debt

- Concept introduced by Ward Cunningham
- Often mentioned, rarely studied
- All experienced software developers "feel" it.
- Drags long-lived projects and products down

Origin of the metaphor

- Ward Cunningham, at OOPSLA 1992

"Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite…
The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise."

Cunningham, OOPSLA 1992

Technical Debt (S. McConnell)

- Implemented features (visible and invisible) = assets = non-debt
- Type 1: unintentional, non-strategic; poor design decisions, poor coding
- Type 2: intentional and strategic: optimize for the present, not for the future.
  - 2.A short-term: paid off quickly (refactorings, etc.)
    - Large chunks: easy to track
    - Many small bits: cannot track
  - 2.B long-term

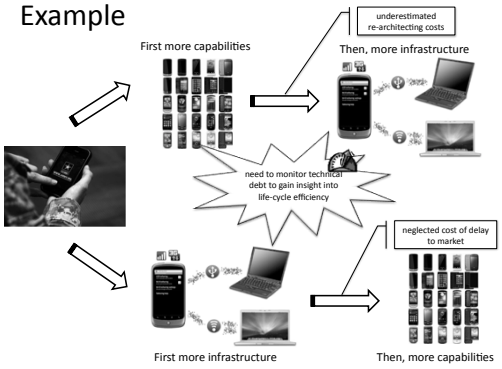
McConnell 2007

Technical Debt (M. Fowler)

	Reckless	Prudent
Deliberate	"We don't have time for design"	"We must ship now and deal with consequences"
Inadvertent	"What's Layering?"	"Now we know how we should have done it"

Fowler 2009, 2010

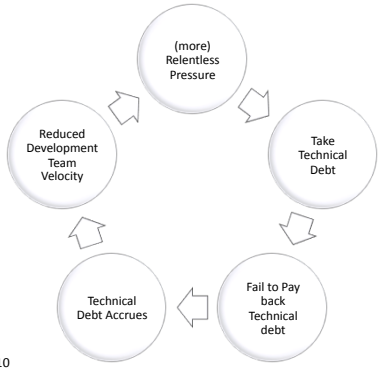
Example



Time is Money (I. Gat)

- Time is money:
  Think of the amount of money the borrowed time represents – the grand total required to eliminate all issues found in the code

Gat 2010



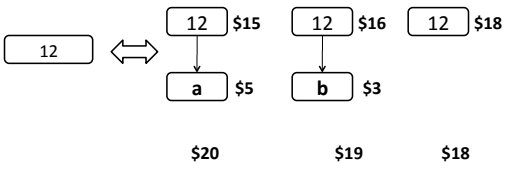
Israel Gat, 2010
http://theagileexecutive.com/2010/09/20/how-to-break-the-vicious-cycle-of-technical-debt/

Technical Debt: invisible and negative value But positive cost

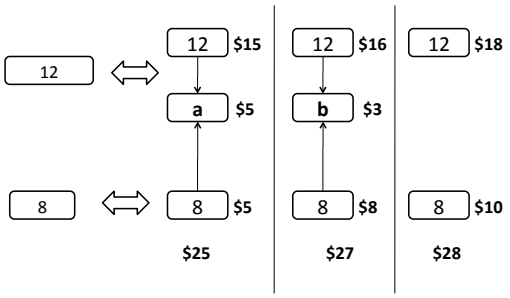
	Visible	Invisible
Positive Value	**Visible Feature**	**Hidden, architectural feature**
Negative Value	**Visible defect**	**Technical Debt**

Kruchten 2009

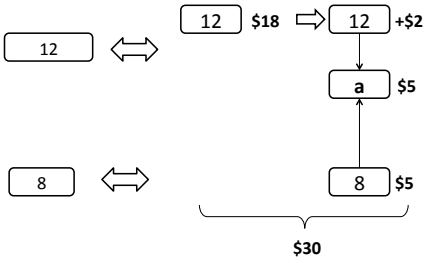
Technical Debt (1)



Technical Debt (2)

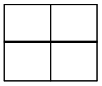


Technical Debt (3)



Technical Debt

- Defect = Visible feature with negative value
- Technical debt = "Invisible feature" with negative value



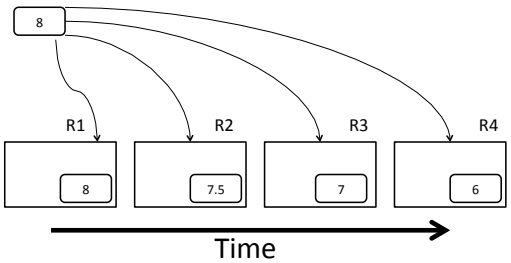
- Cost …. of fixing
- Value …. of repaying technical debt, intersts loss of productivity, etc.

Interests

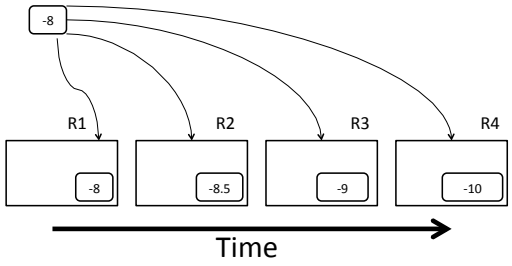
- In presence of technical debt:
  Cost of adding new features is higher
- When repaying (fixing), additional cost for retrofitting already implemented features
- Technical debt not repaid => lead to increased cost, forever
- Cost of fixing (repaying) increases over time

M. Fowler, 2009

Deferring implementation: Value decreases



But technical debt increases over time



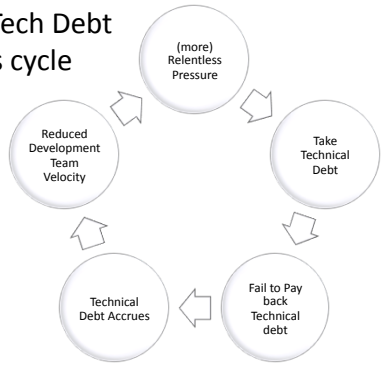
Properties of technical debt

- Type
  - Strategic, architectural
  - Code
  - Test (acceptance test or lack thereof)
  - Other: Design, documentation, training and education?
- Value, and net present value
- Visibility (to whom)
- Impact
  - Localized, scattered
- Environment

Forms of technical debt

- Code level (McConnell type 1)
  - "Code smells"
  - Code duplication
  - Small violation of naming and coding standards
  - Comments
- Architectural design
  - Choice of component, framework, etc.
- Testing debt
  - Unit level and system level (acceptance tests)
- Documentation debt, …

Gat's Tech Debt vicious cycle



Israel Gat, 2010

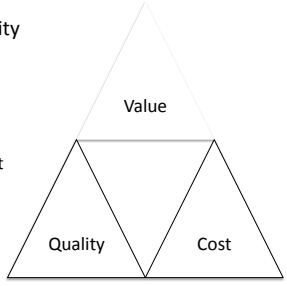
http://theagileexecutive.com/2010/09/20/how-to-break-the-vicious-cycle-of-technical-debt/

Agile mottos

- "Defer decision to the last responsible moment"
- "YAGNI" = You Ain't Gonna Need It
  - But when you do, it is technical debt
  - Technical debt often is the accumulation of too many YAGNI decisions
- "We'll refactor this later"
- "Deliver value, early"
- *Again the tension between the yellow stuff and the green stuff.*

Value, Quality, Constraints

- Value = extrinsic quality
  - Metric: Net present value
- Quality = intrinsic quality
  - Metric: Technical debt
- Constraints = cost, schedule, scope
  - Metric: Cost



Highsmith 2010

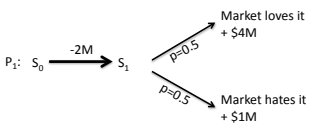
Where the metaphor breaks

- Technical debt does not always have to be repaid
- Negative connotation
- May increase the value of a project for a time
- TD as Investment?

Where the metaphor breaks

- Technical debt is more a rhetorical category than a technical or ontological category.
  - The concept resonates better with the development community.
- Initial investment at T0 in an environment E0. Now in T2, E has changed to E2, a mismatch, has occurred, which creates a debt.
  - The debt is created by the change of environment. The right decision in the right environment at some time may lead to technical debt.

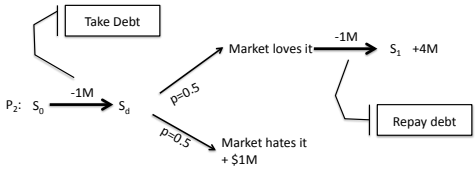
TD and Real Options



NPV (P_1) = -2M + 0.5x4M + 0.5x1M = 0.5M

Source: K. Sullivan, 2010
at TD Workshop SEI 6/2-3

TD and Real Options (2)

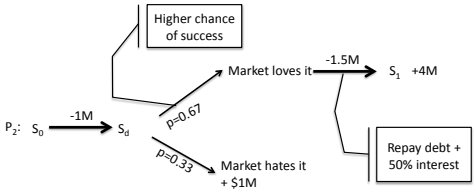


NPV (P_2) = -1M + 0.5x3M + 0.5x1M = 1M

Taking Technical Debt has increased system value.

Source: K. Sullivan, 2010

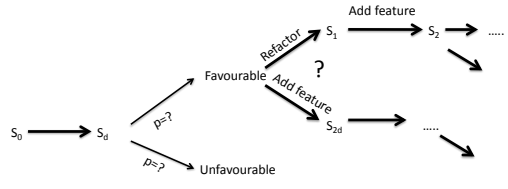
TD and Real Options (3)



NPV (P_3) = -1M + 0.67 x 2.5M + 0.33 x 1M = 1M

More realistically:
Debt + interest
High chances of success

TD and Real Options (3)



Not debt really, but options with different values…
Do we want to invest in architecture, test, etc…

Source: K. Sullivan, 2010

Conclusion

- Technical debt & (lack of) architecture
- Technical debt and risk
- Technical debt and intrinsic code quality
- Technical debt and the definition of "done"
- Technical debt and loss of maintainability and evolvability

Summary

- Complexity and simplicity
- Architecture to the rescue
- Project manager and software architect
- Socio-technical congruence
- Technical debt

What would Octopus say?

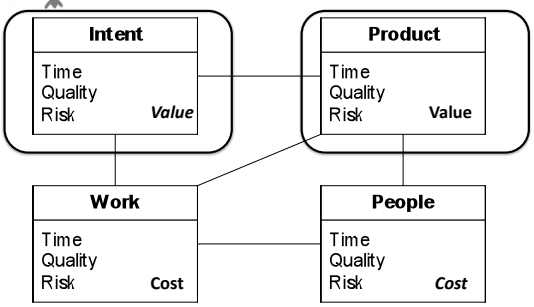
11: Managing Changes

Software Project Management
Philippe Kruchten

Module outline

- Changes over time
  - Intent or Product?
- Software Configuration Management (SCM)
- Change management
- More project measurements
- How adaptive should we be?

Changes over Time



Changes over time

- Changes of Intent
  - Scope change, scope creep
  - Will impact *Work*
  - Will affect Effort and Duration
- Changes in the Product
  - When several people are involved, need to carefully control changes and evolution of artifacts, code and others
  - SCM, Software Configuration Management

Software Configuration Management

- Methods for administering an evolving set of related products and documentation
- IEEE-Std-610:

  Configuration Management is a discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements.

A Common Software Mishap

- Monday: 9:30 Joe tackles bug report 546
  - Find the flaw, fixes it, test the fix
- Monday: 9:30 Eva tackles bug report 854
  - Find the flaw, fixes it, test the fix
- Monday 3:30 Saami runs a simple test, nothing works (systems hangs)
  - the only 2 changes done where by Joe and Eva…. who claim "not guilty"
- What went wrong?

Software Configuration Management

- A supporting process whose purpose is
  - to identify, define, and baseline items;
  - control modifications and releases of these items;
  - report and record status of the items and modification requests;
  - ensure completeness, consistency and correctness of the items;
  - and control storage, handling and delivery of the items.

Configuration & Change Management

- Steve McConnell:
  - Configuration Management is the practice of handling changes systematically so that a system can maintain its integrity over time. Another name for it is "change control."
    It includes techniques for evaluating proposed changes, tracking changes, and keeping copies of the system as it existed at various points in time.

SCM key functions

- Identification
  - Name "items" "versions" and "configurations"
- Control
  - Release
  - Integrity and consistency
- Status
  - Relative to change requests
  - Versions
- Audit

Definitions

- Configuration item (CI): an element considered as a single entity, uniquely identified, but which may evolve over time
- Configuration: a set of configuration items
  - Key relationship: "Consist of"
- Version: one instance in time of a CI
- History: the succession of versions, as a CI evolves in time
- Baseline: a configuration that has been approved to serve as a reference

Definitions

- Marion Kelly:
  - A configuration item (CI) is any part of the development and/or deliverable system (whether software, hardware, firmware, drawings, inventories and/or documentation) which needs to be independently identified, stored, tested, reviewed, used, changed, delivered and/or maintained. CIs can differ widely in complexity and may contain other CIs in a hierarchy.

Typical issues (Life without SCM)

- Simultaneous update, by 2 persons
  - Joe and Eva story
- Coordinated update
  - need to simultaneously change several items to effect one single change
- Shared code
  - Same item used in 2 places (2 products)
  - Impact on product B if a change is made for product A
- Building a complete product
  - exploiting knowledge of what was actually changed

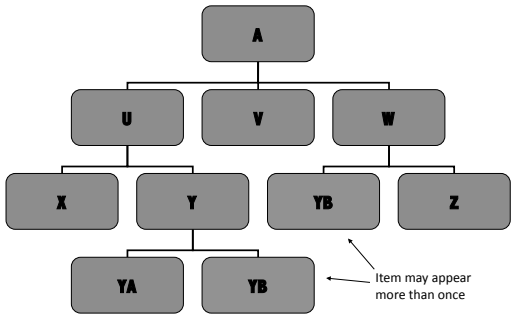
Typical issues (Life without SCM) (cont.)

- Why did it break?
  - what changes were done which caused the system to not work? Defect analysis
- Reverting to a stable state
- What will be affected if I do this change?
  - who need to know
  - what would be broken
- What was delivered to this customer?
  - Identification for defect database
- When, where was this problem introduced? (by whom?)
  - Root cause analysis, process improvement

Typical issues (Life without SCM) (cont.)

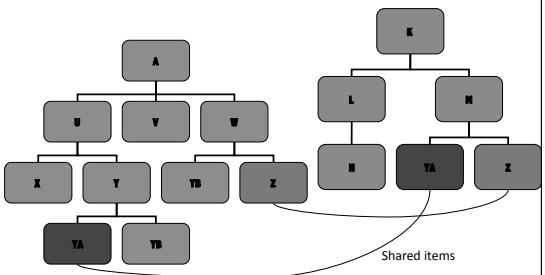
- I am new here? where is all the stuff? How do I find my way?
  - … in 4 millions lines of code
- I am taking over from Sam who left. Do I start everything from scratch?
  - … When CM is done by having Zip disks in one's upper left drawer
- Our server crashed….
- Where are we? What has really been done?
- I am pretty sure I did fix this already, didn't I?
- We need to release now: let us remove this new feature x; it is far too immature.

Product structure: Tree

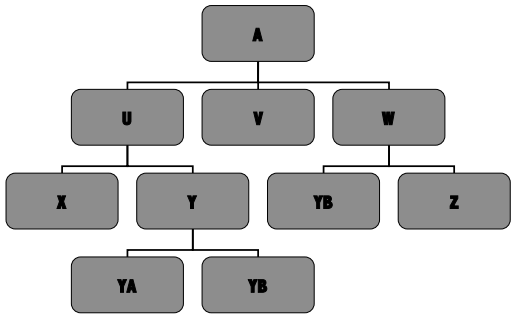


Product Structure: Tree -> Forest

- 2 products A and K both contain items YA and Z



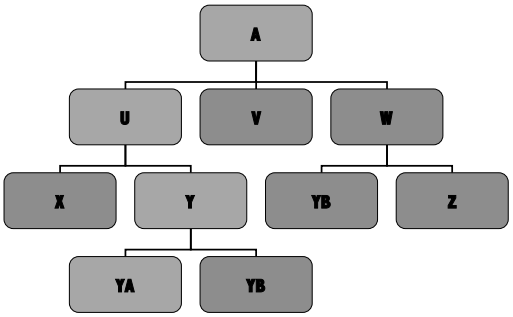
Product structure: change impact



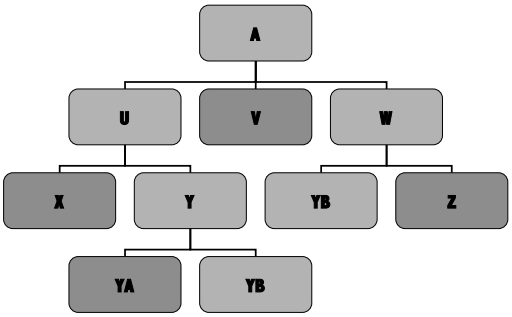
If I change…

- V: only A is affected
- X: U and A are affected
- Y1: Y, U and A are affected
- Y2: Y, U, W and A are affected

Product structure: change impact



Product structure: change impact

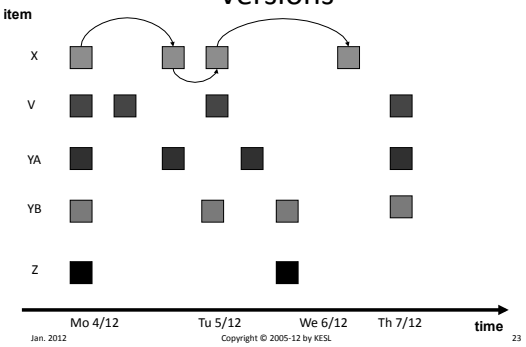


Relationship A "is made of" B

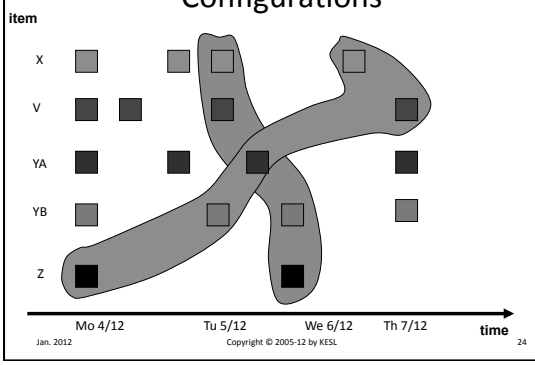
- file A includes file B
- file A referred to definitions in B
- file A is the result of compiling B
- file A includes bits of B by linking
- file A is the result of some transformation of B

- Give a software example for each

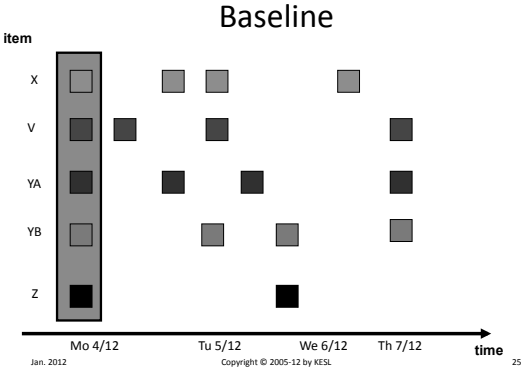
Versions



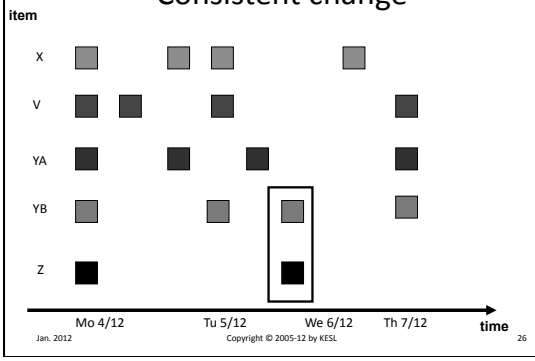
Configurations



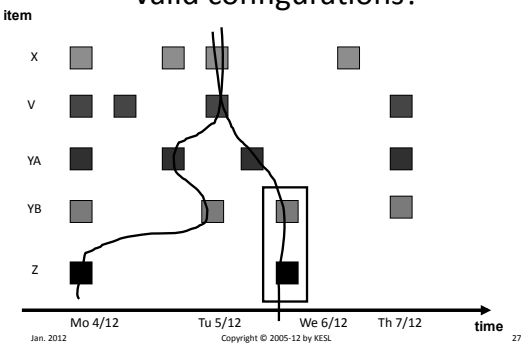
Baseline



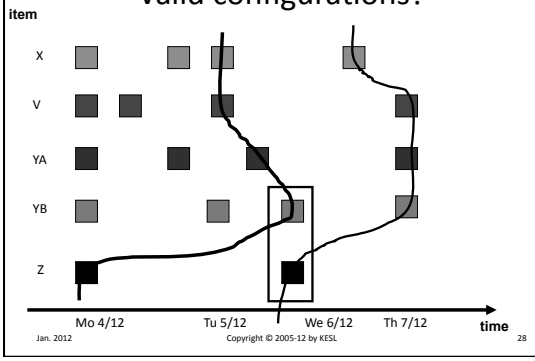
Consistent change



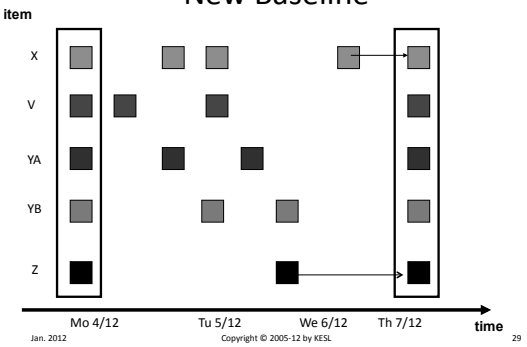
Valid configurations?



Valid configurations?



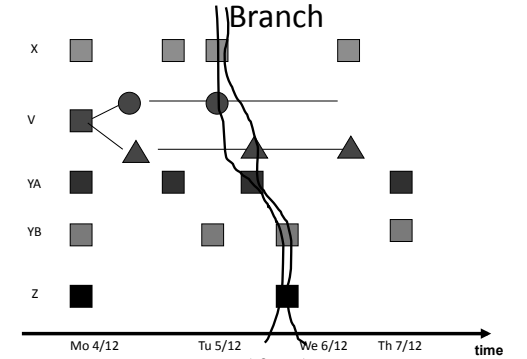
New Baseline



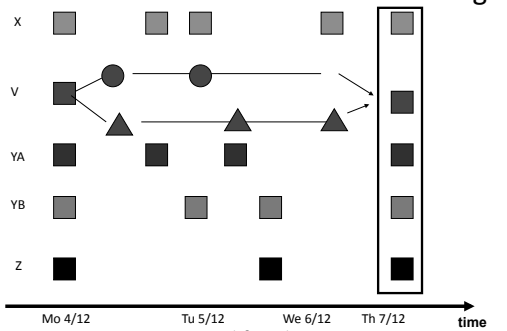
Baseline

- Inventory of a set of versions for a set of CI
- Identifiable and reproducible
- Reference point for future evolutions
- Offer a place to go back when things go real bad
- Minimize storage
  - May archive or even discard steps in-between
- Baseline
  - whole system
  - by subsystem

Branch



Branch: new baseline - chose or merge



Check-out, check-in

- Take temporary ownership
- Basic mechanism to resolve conflict of access (simultaneous update)
- Identification of source of change
  - reason, author, date, context, ….
- [Check-out, modification, check-in]
  - creation of a new version

Private workspace

- Make a "virtual" copy of a set of CI
- No check-out
- Make a real copy at first attempt to update
- If conflict at next baseline: propose a merge

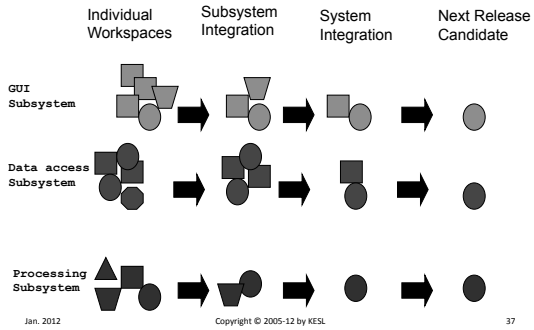
Staging areas, for code integration

- Similar to private workspace, but shared
- Push individual, validated changes, for integration with the rest of the system
- Integrating larger and larger subsystems

Build, release

- Identify a valid configuration
  - consistent set of changes
- Assemble software
  - "makefile" or other reconstruction mechanism
- Test
- If satisfactory, make a new baseline
  - a release

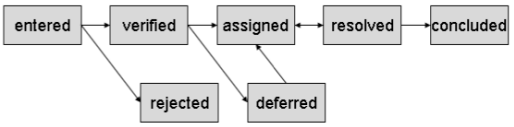
Staging areas: Promotion paradigm



Change Management

- Change order
- Changeset
  - associated to a change order
- Baseline N+1
  - = baseline N 'plus' the changesets of a series of change order
- Undo or "Remove" a change order…

Typical state machine for a change request



Tough Technical Issues in SCM

- Reorganizing:
  - Renaming files
  - Migrating files
  - Merging files (semi-automatic, error-prone)
- Minimizing copies
  - disk space…
- Distributed CM
  - work done on multiple sites
- Security
- Backup, recovery

Configuration Management Plan

- Annex to a good Software Development Plan
- Defines:
  - what is under control
  - how to name it
  - how to evolve it
- Defines CIs, versions, builds, releases
- Policies
  - check-in, check-out
  - individual workspaces
  - staging areas: test, integration
  - customer release management
  - CCB (Change control board)

SCM Plan: Typical Outline

- SCM Environment:
  - Tool used, reference to local user instructions
- Directory Structure:
  - Workspaces and roles (private, test, review, integration, release)
- Naming and storage conventions
- Version and release numbering scheme
- Policies: for docs, for code
  - access rights, check-in/out, change request
  - Reconciliation, Back up, archival, etc.

SCM and Web Content

- Web sites
  - Many elements, many authors, many simple "contain" dependencies
  - Fast turn-around
- Content Management tools
  - Linked with HTML editors, site checkers, graphics and animation editors…
  - Staging of contents (approval process)
  - "Publishing" (or "go live")

Auditing, Status

- Baseline: identification, missing artifacts, etc…
- Have approved change requests (fixes and new feature) been effectively implemented?
- What is the state, the health of the software?
- Extract measurements linking
  - defects
  - changes
  - rework
  - test coverage
  - effort
- Corrective Actions

Change Management

- Configuration management and change management are strongly interrelated.
- Change come in many ways:
  - new features
    - customer & user requests, competitive pressure
  - defect to fixes
  - evolutions of environment
    - OS, COTS products, legal
    - Staff, tool, methods

Change: boon or curse?

- Two opposite philosophies:
- Change is bad
  - Do more planning, and more precise planning
  - Resist change at all cost; follow the plan
- Change is good
  - "Embrace change" (agile mantra)
  - Do more tactical maneuvering, time boxing

Range of strategies

- **Formal versus Informal**

- Change request
- Daily stand up meeting
- Back log management
- Kanban
- Change Control Board

Managing changes

- Like what we have seen for defects in module: Managing quality
- Change requests
  - database
  - open, approved, assigned, in progress, in test, verified, closed
  - priority, impact, etc.

Change Control Board (CCB)

- In medium to large projects
- Coordination between
  - project management
  - customer representative (product manager)
  - test
  - development (architect, integrator…)
- Approve change request (I.e., request to be implemented)
- Define baselines

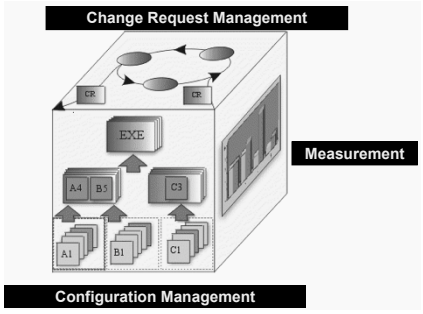
Software maturity : steps to a release

- Feature freeze
  - CCB approves which features and fixes will be released
- Code slush
  - only bug fixes, no new feature in source code
- Code freeze
  - No changes to source base
  - Documentation can still evolve
- Bit freeze
  - No changes at all; final testing
- Code complete
  - Release ready

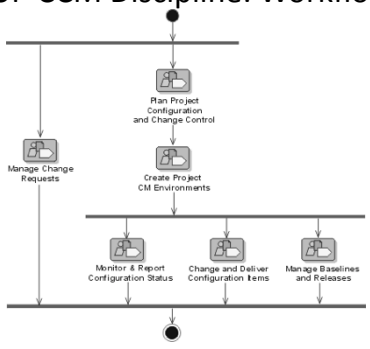
Key roles

- Project manager
- Configuration manager
  or
- Release manager
- Quality Assurance

RUP Configuration & Change Mng

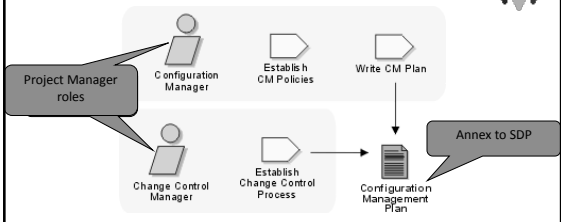


RUP CCM Discipline: Workflow

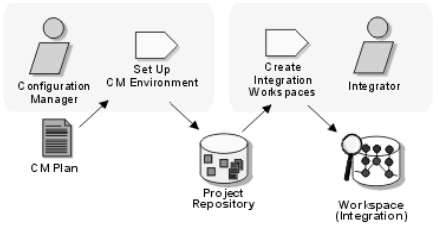


RUP CCM Discipline: Roles & Artifacts

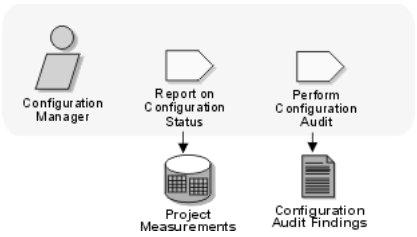
Plan Project Configuration and Change control



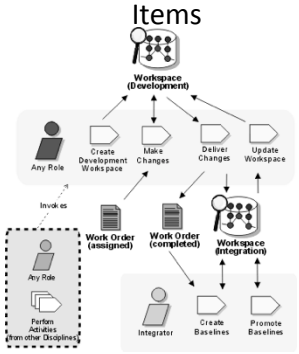
Create CM Environment



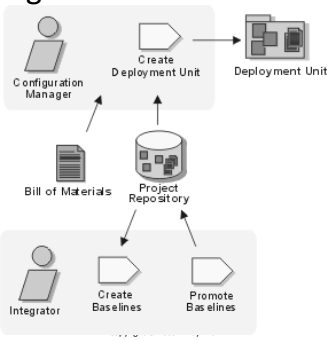
Monitor and Report Configuration Status



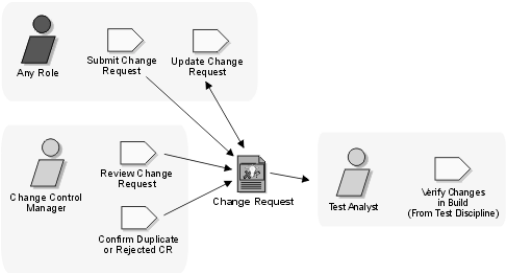
Change and Deliver Configuration Items



Manage Baselines and Releases



Manage Change Requests



Dangers

- Not enough control
- Too much control
- Control too early
  - during elaboration phase, not a the start
  - all at the of elaboration (first major baseline)
- Heavy tooling
- Major changes too late

CCM and the PMBOK

- Configuration Management
  - Not much: 4.3.2.2, CCB
  - Inside Integrated Change Management
- Integrated Change Management
  - Inputs:
    - Project plan
    - Performance reports
    - Change request
  - Outputs:
    - Project plan updates
    - Corrective actions & lessons learned

CCM and the PMBOK

- PMBOK: "plan the work, then work the plan"
- In software: Much more adaptive approach

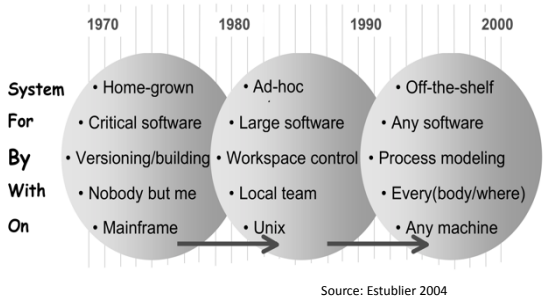
Adaptation versus Anticipation

- How adaptive should we be?

CCM and IEEE SW eng. Standards

- IEEE Std.828-1998: Standard for Software Configuration Management Plans
  1. scope
  2. responsibilities (who)
  3. activities (what)
  4. schedule (when)
  5. resources (how)
  6. plan upkeep
- IEEE Std.1042-1987: Guide to Software Configuration Management

Evolution of CM Tools



Source: Estublier 2004

SCM Tools

- Old guard: SCCS, DCS, PVCS, DDE…
- MST: SourceSafe
- Open source favourite: CVS, Subversion
- The Rolls Royce: ClearCase ($$$)
- New darling: GIT

12: Managing Software Assets

Software Project Management
Philippe Kruchten

Module outline

- Software as an Asset
- Software Reuse
- Software Product Line
- Software and Intellectual Property
  - Copyrights & Trademarks
  - Patents & Trade secrets
- Open Source Software
- Knowledge as an Asset ?

Software is a Asset

- Value of Software
  - Valuation
  - Effort to date?
  - Market value?
- People or code, or both?

Getting value out of our assets

- Software Reuse
  - Reuse? Or simply “use”
- Product Line
- Software portfolio

Benefits of reuse

- Increased reliability
  - Components exercised in working systems
- Reduced process risk
  - Less uncertainty in development costs
- Effective use of specialists
  - Reuse components instead of people
- Standards compliance
  - Embed standards in reusable components
- Accelerated development
  - Avoid original development and hence speed-up production

Flavours of Reuse

- Libraries of code (functions)
  - Data structure, math/stat, GUI, strings, ….
- Systematic software reuse
- COTS-Based Development
- Component-based development
- Component-Based Software Engineering
- Asset-Based Development
- Asset-Based Software Engineering
- Open Source software development (?)

What is reused?

- Not just source code
  - requirements, design models, tests, etc..
- General purpose components
- Highly specialized components
- Domain engineering
- *We do reuse people, though.*

Levels of reuse

- Ad hoc
  - If I find something useful, …. scavenging
- Facilitated
  - Encouraged by the organization
- Managed
  - Organization enforces some reuse practices
- Designed
  - Organization actively invests in designing for reuse

Impact of Reuse

- Ad hoc
  - Cost: nil; benefits: minimal
- Facilitated
  - Cost: low; reuse level: 10-15%
  - Need repository, metadata for search
  - Incentives to employees for contribution and for reuse
  - Reuse "evangelist"

Impact of Reuse (cont.)

- Managed
  - Cost: medium; level of reuse: 15-50%
  - Active people and groups to organizes assets
  - Explicit process; explicit goals; measures
  - Complex repositories, extensive metadata
- Designed
  - Cost: high; reuse level: 40-90%
  - Permeates the whole organization
  - Manage requirements, portfolios; generators

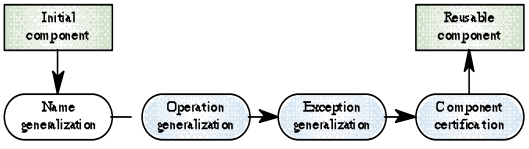
Design For Reuse; Design With Reuse

- Design for reuse:
  - Extra effort in the software development
  - Generalization
  - Documentation
- Design with reuse

Design For Reuse

- Name generalization
- Operation generalization
- Exception generalization
- Component certification

Source: Sommerville 2000



Design With Reuse

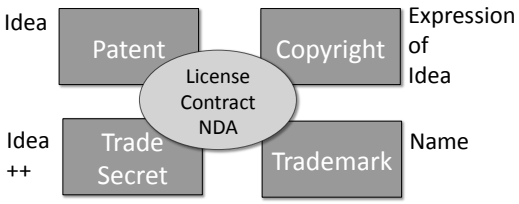
- How to find the right component to reuse?
- Confidence in level of quality, stability
- How to reuse effectively
  - Documentation, example, support
- Effort:
  - Qualify
  - Select
  - Adapt
  - Integrate

Reuse and Project Management

- Myth: Software reuse will just happen
- Reuse will happen to a large scale only if it is a management decision to make it happen and if the decision is backed by appropriate investment in tools, people, process, and education.

Protecting your assets

- Intellectual Property (IP) rights



Trademark

- Help indentify a product and distinguish from competition
- "Distinctive" name
- Valid forever
- Name used as a adjective!!

Patent

- Protect an invention
- Novel, nonobvious, and useful
- Right to exclude others from using the invention
- Valid about 20 years
- Cost $1500 to $10000
- Owned by persons (who can assign it to others)

Copyright

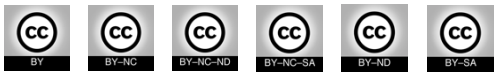
- Protect the expression of an idea, not the functionality
- Valid for life of author +50 or 70 years
  - Very long ("Mickey mouse protection act")
- Automatic, though you may make it clear:
  - Copyright © 2010 by Philippe Kruchten
- Illegal to remove 3rd party copyright notice

Creative Commons Licenses

creative commons

Allow to better specify what others can do:

- Attribution (by)
- Non Commercial (nc)
- No derivative works (nd)
- Share alike (sa)



Trade Secret

- Information not know, and protected
- Protect against theft, unauthorized disclosure
- Valid… until secret lost

Non disclosure agreement (NDA)

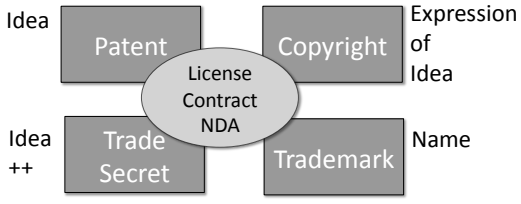
- Protect trade secrets
- … and other useful information someone visiting may gather around…
- Often an integral part of an employment contract

Employment contracts and IP

- In general, your ideas belong to your employer, if they are software-related.
- Even if you have them at home!
- Specify what is yours before starting to work
  - Without violating another non disclosure agreement (or trade secret)
- Keep written trace of what you do

Protecting your assets

- Intellectual Property (IP) rights (IPR)



Open Source (as a form of reuse)

- 30,000 projects on freshmeat.net
- 70,000 projects on sourceforge.net
- 5400 Perl modules on the cpan.com
- 10,000 ports on the FreeBSD
- Free Software Foundation
- Open Source Software Development

Open Source Development

- SHARE (up to the mid 60's)
- BSD (Berkeley Software Distribution)
- Free Software Foundation
  - Richard Stallman (MIT)
  - GNU (GNU is Not Unix)
- Eric Raymond: The Cathedral and the Bazaar (paper , then book in 1999)
- SourceForge, SlashDot

Open Source Successes

- EMACS (editor)
- MySQL (relational database)
- Tomcat (application server)
- Linux (OS)
- Eclipse (programming environment)
- Apache
- Mozilla (browser)
- Indirect: people working together

Open Source Initiative

- Open Source Definition
  - www.opensource.com
- Certification mark

Cost?

- Free? Open-source is not freeware
  - cost? not necessarily;
  - free access to source code: yes
- Hidden cost?
- Distributions and fixes, support
  - RedHat Cygnus VA etc…
- Return On Investment, when cost is zero?
- Licensing issues…

Benefits

- Quality?
  - Shame
  - Many eyeballs
- Availability
- Support
- Only for a very tiny fraction of the projects

A Utopian Society?

- Fueled in large part by research grants in academia; then by consulting
- Very disciplined process; CM is key
- Very hierarchical process
  - Meritocracy
- Ugly politics
- Lots of posturing
- Only where this is no competitive advantage to be had

ABC of Open-source licenses

"You can freely copy this code, as long as you follow these rules …."

- Propagation (of license and copyright notice)
- Put-back (improvement and fixes)
- Virality (=adjacent code carry same license)
- Patents grants
- As is (no implied warranty)
- Non commercial

Licensing issues

- Need to package separately proprietary software from GPL code
  - or the proprietary code becomes contaminated and must be GPL'ed
- But many different other schemes exist
  - see www.opensource.com
  - Mozilla, Python, etc…
- Different rules; must play by the rules

On the Dark Side

- Licensing
- Copyright
- Trade secrets
- Patents

- all to limit, restrict access to source code, to limit distribution, prevent modifications

Open Source and Project Management

- Be aware of the rules
- Weigh pros and cons
- Adjust own's license to match that of reuse software
- May not work in contractual work

- May have to put limit on developers involvement
  - addictive nature of meritocracy….

Simple guidelines for project managers

- Do not permit the uncontrolled importation of software onto company computers.
  - both open source and proprietary
- Know how the software will be used.
  - will it be used in-house as is, or modified and embedded in some other product
- Have a means for documenting what software, and what version of that software, is in use.
  - remember SCM software configuration management…

Simple guidelines for project managers

- Require documentation of all internal software development projects.
  - keep track of modification to OS; keep track of authors and which projects they have worked on
- Avoid contamination
  - developer who has seen proprietary code should not be involved in a similar project
  - same with access to patents

Simple guidelines for project managers

- Contributions to open-source project
  - Review (or have reviewed) contributors agreements (CA)
  - Ask newcomers for previous CA
- Decision to open source software is a delicate one
  - Not done my individuals on their own initiative

Software Patents: Good or Bad Idea?

- Patenting an idea? A math theorem?
- Bad examples (?)
  - Lemelson computer vision case
  - Compression algorithm
  - RSA encryption
- Large companies: playing the numbers
- Leaving a minefield for the small and medium software developers.
- Patent trolls

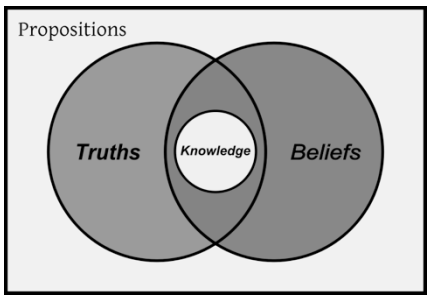
Summary

- Software reuse is a good thing
  - Software reuse does not simply happen, it
  - must be a planned decision
- Open Source Development is a good thing
  - But must be handled very carefully by software project managers with strict rules to avoid getting in a trap
  - May need to put a cap on developer's involvement
- Software patents seem a bad idea
  - But we have to live with them as big players are using them as a weapon

Knowledge

- Expertise, and skills acquired by a person through experience or education; the theoretical or practical understanding of a subject
- What is known in a particular field or in total; facts and information
- Awareness or familiarity gained by experience of a fact or situation
- Plato wrote that knowledge is "justified true belief"

Plato, *Theaetetus*



Source: Wikipedia Commons

Knowledge as an asset

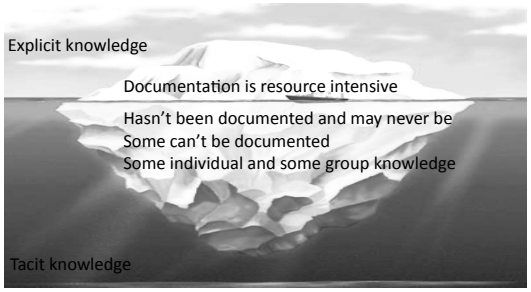
- "Intellectual capital"
- Knowledge workers
- "Knowledge is power" *Bacon*
- Knowledge management:
  - sharing, distributing, creating, capturing and understanding the knowledge of an organization

"The major problem with intellectual capital is that it has legs and walks home every day."

Rus & Lindvall 2002

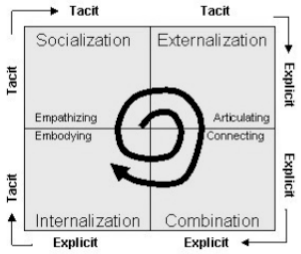
Tacit Knowledge
vs.
Explicit Knowledge

Tip of the iceberg?



Source: http://www.anecdote.com.au/archives/2007/08/what_do_we_mean.html

SECI Model (Spiral of Knowledge)

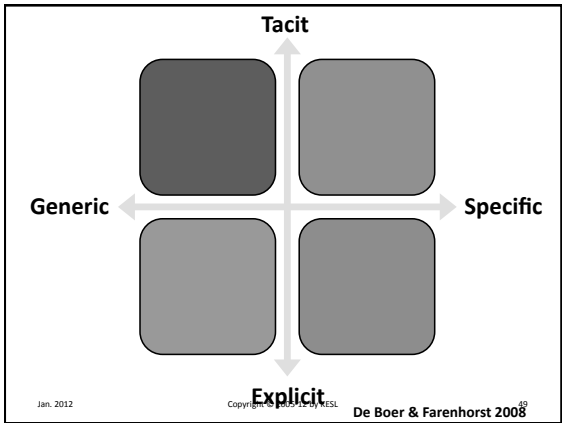


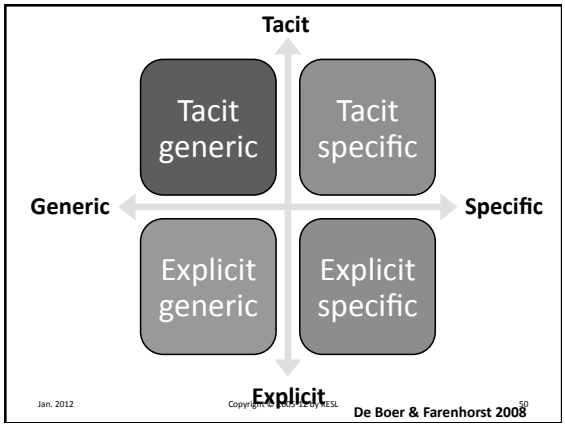
Nonaka & Takeuchi 1995

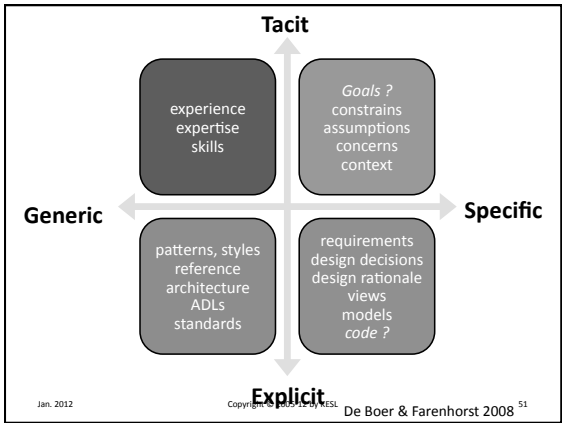
SECI

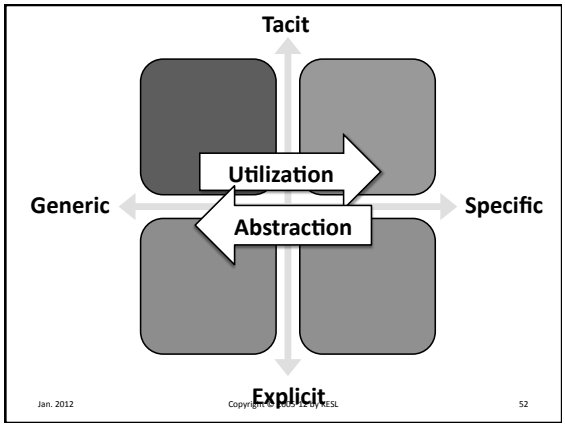
- Socialization
  - people speak to each other; dialogue; sharing experience; brainstorming
- Externalization
  - some well-established knowledge is being transcribed; writing it down; creating metaphors; modeling
- Combination
  - bits of knowledge are being (re)-used in various ways; categorizing; creating methods; defining "best practices"
- Internalization
  - Other access to this new knowledge; training; learning by doing

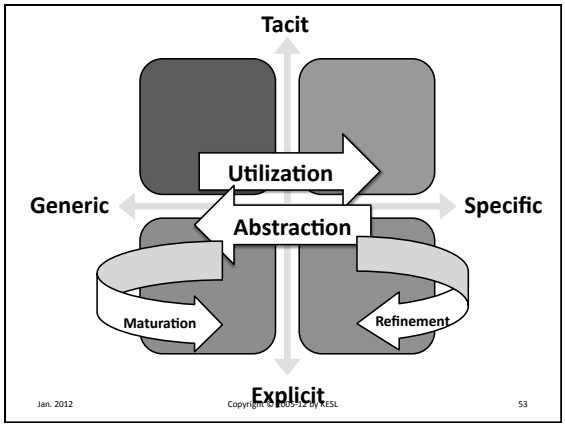
System - Specific Knowledge
vs.
System - Generic Knowledge

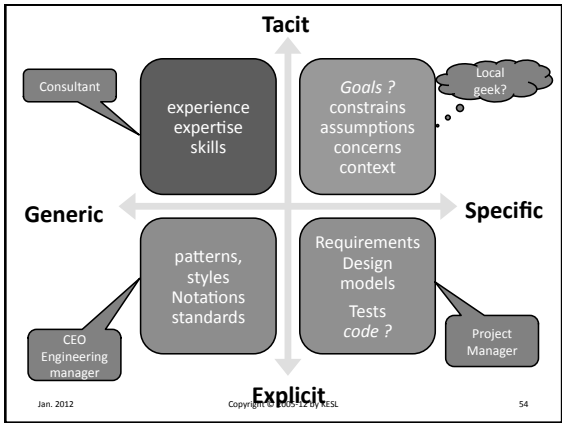

Tacit
Generic
Specific
Explicit
Jan. 2012
Copyright © 2005-12 by YESL
De Boer & Farenhorst 2008
49


Tacit
Tacit generic
Tacit specific
Generic
Specific
Explicit generic
Explicit specific
Explicit
Jan. 2012
Copyright © 2005-12 by YESL
De Boer & Farenhorst 2008
50


Tacit
experience
expertise
skills
Goals ?
constrains
assumptions
concerns
context
Generic
Specific
patterns, styles
reference
architecture
ADLs
standards
requirements
design decisions
design rationale
views
models
code ?
Explicit
Jan. 2012
Copyright © 2005-12 by YESL
De Boer & Farenhorst 2008
51


Tacit
Utilization
Generic
Specific
Abstraction
Explicit
Jan. 2012
Copyright © 2005-12 by YESL
52


Tacit
Utilization
Generic
Specific
Abstraction
Maturation
Refinement
Explicit
Jan. 2012
Copyright © 2005-12 by YESL
53


Tacit
Consultant
experience
expertise
skills
Goals ?
constrains
assumptions
concerns
context
Local geek?
Generic
Specific
patterns,
styles
Notations
standards
Requirements
Design
models
Tests
code ?
CEO
Engineering
manager
Project
Manager
Explicit
Jan. 2012
Copyright © 2005-12 by YESL
54

Knowledge management strategies

- Codification
  - Capture information, store it, retrieve it

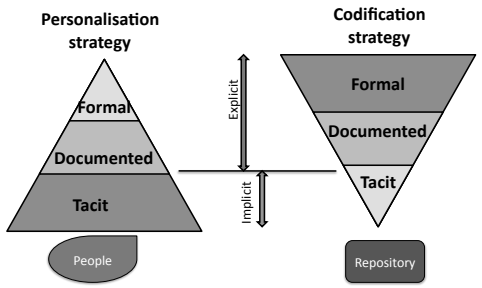
Repository

- Personalisation
  - Define who knows what, "yellow pages"

People

Jan. 2012 **Hansen et al. 1999** 55

Strategy & knowledge type



Jan. 2012 56

Source: Paris Avgeriou

Codification

Repository

- Initially the focus of the community
  - The engineering attitude!
  - Heavyweight tool support
  - Ontologies and elaborate templates
- Fits mostly with BUFD (e.g. RUP)
- Pure codification has several issues
  - Too expensive to create/maintain
  - Bureaucratic
  - Lack of motivation

Jan. 2012 | 57

Personalization

People

- Seems elusive at first
- Mostly "yellow pages"
- But has merit
  - Non-intrusive
  - Tooling lighter/easier
  - Human-friendly
- Fits mostly with Agile methods
- Most of Knowledge sharing in practice is personalized!

Jan. 2012 | 58

The hybrid approach

Repository

People

- Combining the best of both worlds
- Depends on
  - team size, organization, culture, geography

Two rules of thumb:

1. Temporal
   - Inception: Personalize
   - Elaboration: Codify
2. Variability
   - Artifacts that change often: Personalize
   - Artifacts that change infrequently: Codify

Jan. 2012 59

Summary

- Software is an asset
- Protect the asset
- Leverage existing assets

- More valuable is knowledge about the software
- Protect this knowledge
- Leverage this knowledge

Jan. 2012 60

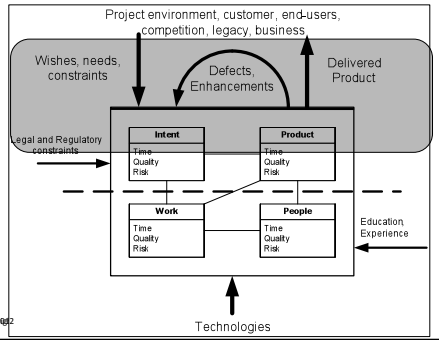
13: Managing Software Products

Software Project Management
Philippe Kruchten

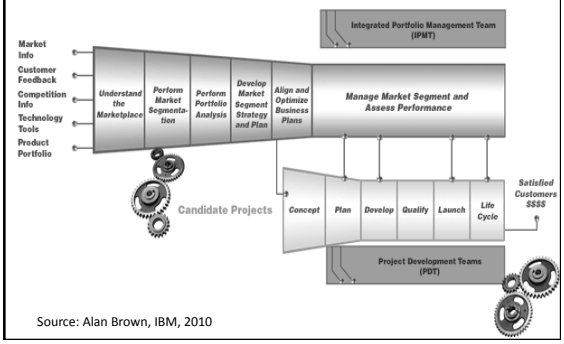
Product

- There is more in a software-intensive product than just developing high quality code
- When the code is done, done, are you really done?
- A whole set of other activities may have to take place that will eat a good chunk of your resources, your team… since they are the only ones who really know

SW Product development, in context

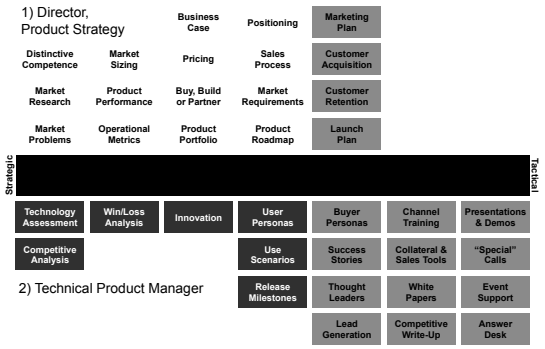


Product Development



Source: Alan Brown, IBM, 2010

Product management triad



Other elements of a complete product?

Take away for Software Project Managers

- Software development rarely done in total isolation
- Interface with other disciplines will take time, resources
- Understanding of the various roles
- Collaboration, dialogue, … education
- Mutual respect, not constant conflict

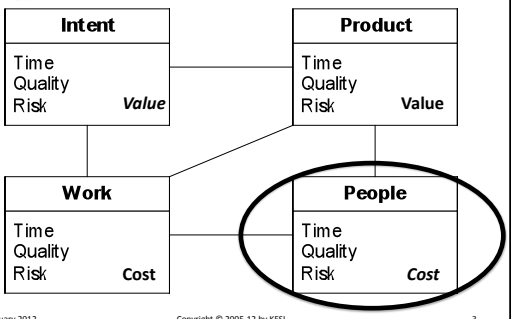
14: Managing people (1)

Software Project Management
Philippe Kruchten

Module outline

- People, individuals, and teams
- Roles *vs.* individuals
- Organizational structures
- Teams
- Role of the project manager
- Communication and collaboration
- Culture

People



We already met People…

- Time box & staffing profile
- Effort
- Cost
- Brook's law
- Involved in risk identification
- Involved in estimation and planning
- …

People management

- Not about barking orders, "kicking asses"

It is about:
- Assembling the right set of people
- Enabling them to be effective
- Helping them achieve the common goal

- and be happy about it
  (so that they'll stay and do another one with you)

Major Areas

- Organizational Planning
- Staff Acquisition
- Team Development

Source: PMBOK

Organizational Planning

- Roles, responsibilities
- Group, teams
- Reporting structure(s)

Source: PMBOK

- Staffing management plan
  - describe when and where people with specified skills will be needed on the project

Staff Acquisition

- Staffing management plan
- Staffing pool description
- Recruitment practices
- Negotiations
- Assignments

Source: PMBOK

Team Development

- Team building
- Performance reports, appraisals
  - Performance improvement
- Rewards and recognition
- Collocation
- Training
- Facilities

Source: PMBOK

Humans, not machines

- Time
  - about 40 hours a week, to be sustainable
- Communication
  - N (N – 1)/2 paths
  - Common language (including culture and process)
- Agent
  - Not homogeneous
    - Skill, personal goals, personalities
  - Bounded rationality (H. Simon)

Role *versus* Person

- Role defined by:
  - Activities
  - Responsibilities
  - Competencies
- Persons (individuals) have:
  - Competencies
  - Availability
  - *Cost, location*
  - *Taste, interest ?*

Mapping Roles to Persons

- How many hats do you wear today?
- Small teams: many roles per person
- Larger team: greater specialization
- All needed roles (activities) have to be covered.

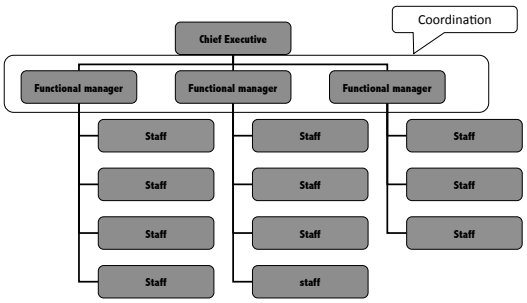
Need for organizational structure

- Coordination
  - Relations between individuals
  - Tasks & results to be achieved
- Software projects are not factories, banks or hospitals
  - High amount of information
  - High level of uncertainty, unknowns, risks
  - Not "assembly line" style
    - Narrow specialty, repetitive work

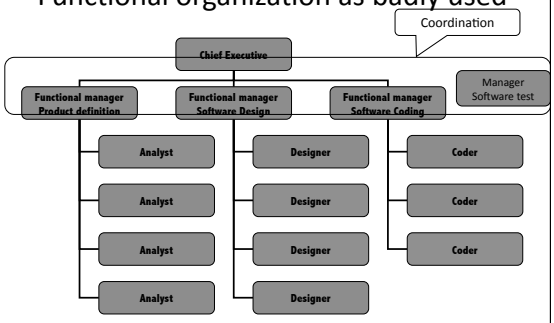
Typical organization styles

- Functional
  - Often found in organizations what have few projects
- Matrix
  - The norm in all large organizations that have projects
- Projectized
  - For small org that have only one single project, or for some special, highly critical projects

Functional organization



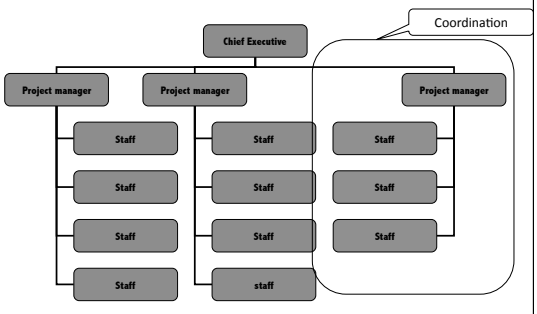
Functional organization as badly used



Functional organization

- Often matches the steps of the waterfall
  - requirements
  - design
  - coding
  - testing
- No real sense of project mission
  - No dedicated project manager
- Hand-over artifacts
- Not very suitable for software projects

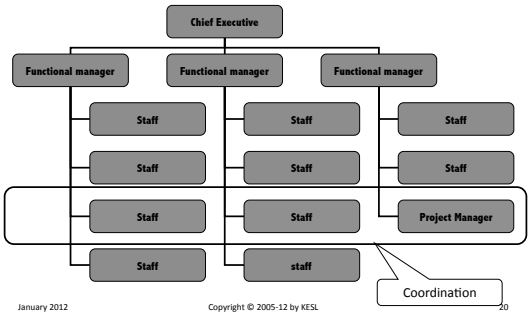
Project organization



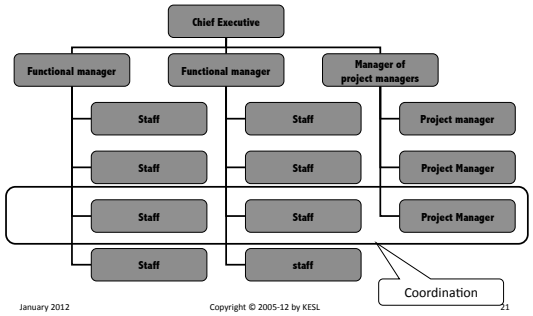
Project Organization

- OK for small organizations, with few projects
- As number of project grows: inefficiencies
  - Not flexible enough in managing time and people
  - Sense of competition between projects, "ownership of people"
- As organization grows, need to share resources, know-how, expertise, process, tooling, etc. across projects.

Matrix organization



Matrix organization (2)



Matrix organizations

- Functional managers manage pools of people of the same feather (specialty)
  - Hiring, training, tooling, process definition, career management, etc.
  - Example: base software, application software, integration and test, electronics, mechanics, power, product definition
- Project managers
  - have the staff on loan for the duration of the project (sometime part-time)
- Most people have "2 bosses"

Matrix organizations allows to

- Balance workload between projects
- Use peer from outside the project for reviews
- Rotate staff to avoid concentration of knowledge on a few indispensable people
- Assess people performance more objectively
  - e.g.: ranking by specialty, not globally.

Alternates

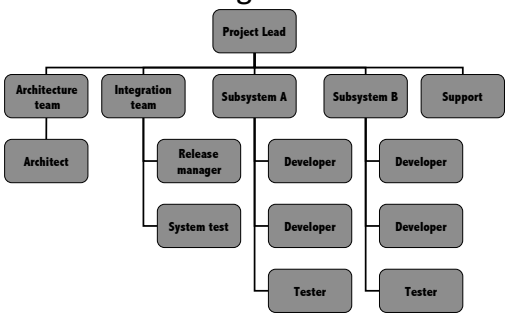
- SWAT team
  - "Skilled with advanced tools"
  - 5-6 people, often generalist, collocated, highly motivated, short term goals
- Harlan Mills: Chief Programmer team

Inside the project

- Organize around the architecture, not the way around
  - Conway's law: the structure of a software system will reflect the structure of the organization that builds it
- Do not organize the project functionally: All testers, all GUI folks, all analysts
  - It'll die by lack of communications
- If feasible, collocate
  - Project "war room"

 25

Medium to Large : team of teams



 26

Team

Source: Katzenbach & Smith

- Small group of people (2 to 12)
- Common objective
- Common mental models
  - Common language and culture
  - Reduce amount of communication
- Mutual respect
- Complementarities

- Different from group, committee, board, …

January 2012 27

Teamicide

- Bureaucracy
- Time fragmentation
- Physical separation
- Phony deadlines
- Defensive management

DeMarco & Lister: PeopleWare

 28

Organizing a team

- Use fewer and better people
- Fit tasks to the capabilities and motivations of the people available
- Help people get the most out of themselves
  - Peter principle: become obsolete
  - Reverse Peter principle: become indispensable
- Balance personalities
- Remove people who do not fit

 29

Agile Practices

- Daily Stand-up Meeting (aka scrum)
- Whole team
- Sustainable pace
  - ("40 hour week")
- Direct communication
  - Pair programming
  - Customer on site
- Walls, board, cards and post-it notes

 30

Working in a team

- Who identifies task?
- Who distributes them?
- Manage the backlog

Project manager…

- Align work interests with work assignments
  - Easy to say, hard to do
  - Work interest can evolve by training, education, etc.
  - Cheerleading and lying will not work well
  - Monetary rewards are mixed blessing
- Show people you appreciate them
- Provide thinking-oriented office space
  - avoid open bays

Project manager …

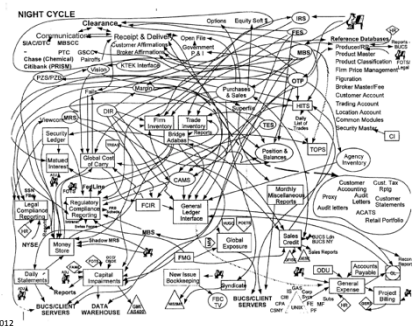
… is at the service of its team, not the other way around ! Team = PM's customers

- Coaching and mentoring
- Setting goals and priorities
- Resolving problems and conflicts
- Providing resources
  - machines, tools, people, etc.
- Career development, training
- Technical guidance (if possible)

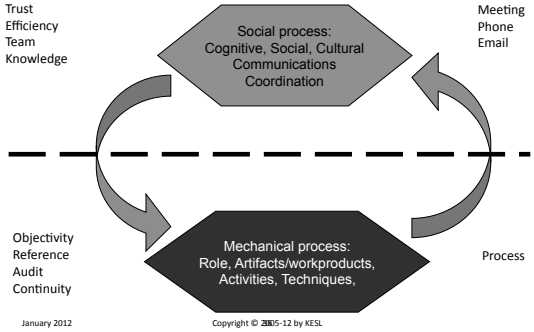
Multiplexing

- Minimize amount of open work
- Get closure as soon as possible
- Define "done" or "completed"
- Spread the knowledge
- Minimize multiplexing

Multiplexing

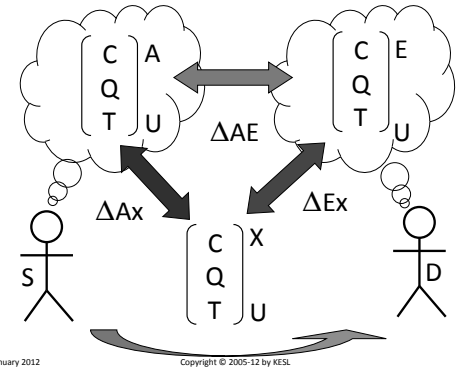


2 facets of software development

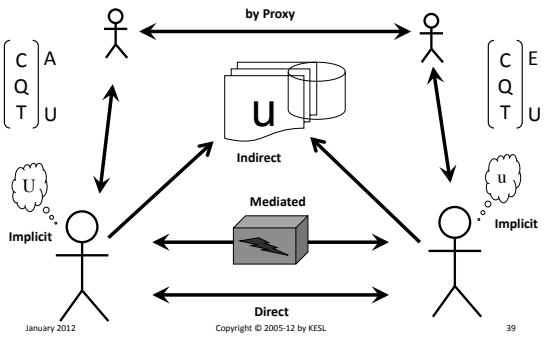


Communication

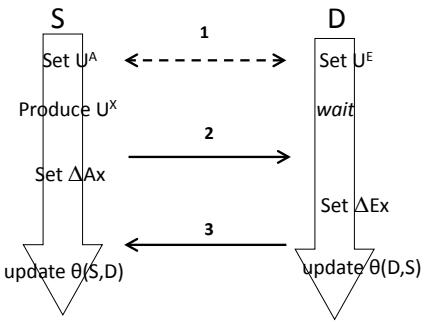
- Modalities
  - Conversation
  - Written: casual
  - Written: formal
- Frequency
- Intermediaries, proxies
- Feedback



Communications



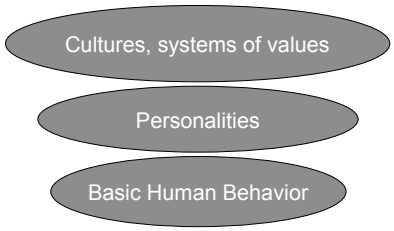
Waltz



Trust

- Successful communication builds trust
- Failures of communication lead to distrust
- Trust is slower to build than distrust
- Lack of trust lads to bureaucracies, checks, defensive attitudes, entrenchment, etc.

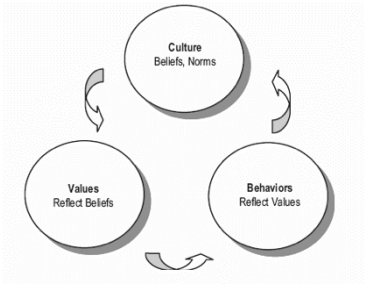
We are all different



Culture is …

- "…a fuzzy set of attitudes, beliefs, behavioural norms, and basic assumptions and values that are shared by a group of people, and that influence each member's behaviour and his/ her interpretations of the 'meaning' of other people's behaviour" *(Spencer-Oatey, 2000).*

Culture



Thomsett 2007

Nested Levels of Culture

- National culture
- Corporate culture
- Team or group culture

- Culture /= Personality

Culture as an Iceberg

Meeting other cultures

- Ethnocentric stage
  - Denial (blame issues on personality or misbehaviours)
  - Defense (and try to force things one way)
  - Minimization (push it under the rug0
- Ethnorelativist stage

  Not one culture is central and reference for judging others
  - Acceptance
  - Adaptation
  - Integration
  - *xenophilia ?*

Cultural factors

- Low context, high context
  - HC: unspoken meanings (jp, cn, fr)
  - LC: just what the words say (us, de)
- Time:
  - Polychronic
    - many things interleaved (Middle east, France)
  - Monochronic
    - one thing at a time, "time is money" (US, Scand.)

Source: E. Hall

Cultural factors: G. Hofstede

IBM employees around the world

Multivariate analysis, lead to 5 dimensions:

- Power distance
- Collectivism versus individualism
- Femininity versus masculinity
- Uncertainty avoidance
- Long-term versus short-term orientation

Source: Hofstede

Other factors: F. Trompenars

- Universalism vs. particularism
  - Judging on fixed rules, or based on circumstances ?
- Individualism vs. communitarianism
  - Self, or group?
- Neutral vs. emotional
  - showing emotions in business setting?
- Specific vs. diffuse
  - How far do we get involved?

Source: Trompenars

	Neutral	Emotional
Specific	USA (east coast), Scand. Approval/ disapproval	USA West coast, Canada Sympathy/Outrage
Diffuse	Japan Esteem/Disrespect	South of Europe Love/Hate

Other factors: F. Trompenars (cont.)

- Achievement vs. ascription (attitude toward titles, degrees,…)

And a few secondary ones, such as:

- Attitude to time
- Attitude to the environment (i.e., nature)
- Gender, race, class, religion

Source: Trompenars

Impact on software development

- Management
- Communication
- Meetings
- Task allocation
- Requirement
- Negotiation
- Bug reporting

Case 1

Monday 10:am

- A: -- we will need all features by Friday at 9:00am, to do the final release to send to the lab.
- B: -- Yes.

Friday 12:00 noon:

- A: -- … but you have not pushed your stuff in the CM system!!!
- B: -- Yes.

Case 2

—I have now some data on the defects.
—Yes, I know. I have already started to address the issues they reported.
—How come…?
—I read the fax in the fax machine earlier today
—But it was addressed to me!
—Yes, but it was in the fax machine… I do not see what is the issue here.
—At least you could have told me and cancelled this meeting.
—I wanted to speak about the new candidate,,,

Case 3

- News release: Companies A and B have reached an agreement, thanks to this last minute compromise.
- A: Agreement gains moral sanction by having resulted from compromising
- B: By compromising, something is lost, honour is not upheld, the principles are diluted.

Case 4

- J., functional manager, is interviewing some 10 candidates for a software development position. An 11th candidate is his wife nephew, who has a hard time finding a job, because he did not quite finish his bachelor's degree. He cancels all remaining interviews and hires him.
- J is in a collectivist, polychronic, high context, hierarchical, feminine society (a)
- J is in an individualist, monochronic, low context, masculine society (b)
- J lives in (a) but works for a company headquartered in (b)

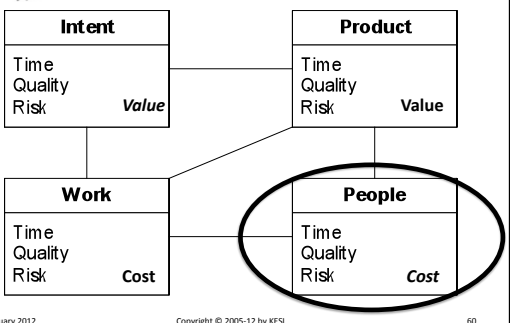
Corporate Culture

- Small set of core values
- Example:
  - Integrity
  - Customer focus
  - Results, value
- Applied thoroughly, not just "lip service"
- Management sets the example
- Congruent organization

Summary

- Software is built by (team of) people
- Tools they use to manipulate softeware are secondary to
  - Communication
  - Collaboration
- Communication and collaboration are affected by competence, education, culture

People: quality and cost



15: Managing people (2)

Software Project Management
Philippe Kruchten

Module outline

- Human resource cycle
- Personality
- Professionalism
- Ethics

Hard HR Management Practices

1. Hiring
   Internal, external
2. Performance appraisal
   - individual
   - global (ranking)
3. Dismissals, Lay-offs
4. Conflict resolution

- Do not improvise
  - Moral, financial, legal issues
- Get support from HR specialists
  - Consultants

Hiring

- Costly (time, travel, relocation, mistakes)

1. Define job and skills required
   - Job description
   - Think long term, career evolution, mix
   - Not tactical short-term need (use consultant)
2. Find candidates
   - Internal (HR, other projects)
   - Networking (previous employments?)
   - Internet
   - Ads in specialized press
   - Campuses (job fairs), interns and coops
   - Recruiting firms

Hiring (2)

3. Select a small group to interview in depth
   - How to read a resume
   - Eliminate bad apples
     - stretching the truth
     - taking undue credit
     - buzzword soup
   - Telephone quick filter
   - Get a second opinion
   - Be aware of your own tastes and biases

Hiring (3)

4. Interviews
   - Objectivity
   - Sorting out real achievements
   - How did you do… (methods, tools, etc..)
   - "How would you do…"
   - Careful on personal questions
     - Gender bias
     - Ethnicity bias
   - Keep at the professional level
     - Citizenship status
     - Family to relocate and assist
5. Making a choice
   - Consensus, balance, fit

Hiring (4)

6. Closing
- Defining a salary
  - Complete benefit package
  - Industry surveys
  - Internal grid
    - Fairness with current staff
  - Negotiations
- Check references
  - sometimes background check, security clearance, etc.
- Making an offer
- Work contract
  - Non-disclosure / confidentiality

Performance appraisal

- Once or twice a year
- Mostly in writing (legal issues)

1. Self assessment
   - Achievements: product
   - Achievements: collaboration, teamwork, leadership, management
   - own strengths and weaknesses
   - Issues
   - Nominate peers
   - Career progression wishes
   - (manager evaluation)
2. Peer assessment

Performance appraisal (2)

3. Supervisor assessment
   - In each category
     - outstanding, exceptional (10%)
     - Beyond normal expectation(30%)
     - Fulfills expectations (55%)
     - Below expectations (5%)
   - Look at the future (short-term and long-term)
   - Specify objective goals and actions
4. Face-to-face meeting
   - 30 minutes, quiet, listen, take notes

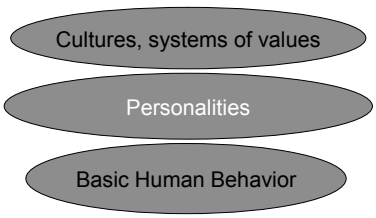
Performance appraisal (3)

5. Final assessment
   - acknowledgment
6. Performance improvement plan (PIP!)
   - 1 to 3 months
   - very specific goals, objective & verifiable
   - mentorship, accompaniment
   - weekly 5-10 minute meetings

Ranking

- Objectives:
  - Fairness across a large organization
  - Identification of company's strengths
  - Preparing a "reduction in force"
  - Raising awareness of managers (educational value)
- Strict ranking versus groupings
  - 1, 2, 3, 4, 5, … ,57 or simply in "buckets":
  - Outstanding (10%), Very good (30), Average (60%), Low (10%)
- By type of activities
  - Managers, developers, support staff, etc…

We are all different



Personality

- 1700's
- Beginning of 20th century (Freud, Adler, etc…)
- Carl Jung
- Myers-Briggs
  - MBTI = Myers-Briggs Type Indicator
  - Kersey's temperaments (www.kersey.com)

Example: Jung & Myers Briggs

- Extraversion E ↔ I Introversion
  - Breadth of knowledge, quick action (E)
  - depth of knowledge, reflection (I)
  - turned toward people and things, or inner world of ideas and images
- Sensing S ↔ N iNtuition
  - Information gathering
  - practical, hands-on (S)
  - theoretical implications, new possibilities (N)
  - trust your senses, or meanings and patterns

Example: Jung & Myers Briggs (cont.)

- Thinking T ↔ F Feeling
  - draw conclusion, make judgment objectively, dispassionately, analytically (T)
  - Weigh human factor, societal aspects, personal convictions (F)
  - rely on objective principles or human feelings
- Judging J ↔ P Perceiving
  - Collect data, make judgment, stay on path (J)
  - alert to changing situation, and quick to adjust (P)
  - structured, planned decision making, vs. flexible adaptable information taking

MBTI

- 16 types
- Example: INTJ
  - =introvert, intuitive, thinking, judging
- Most software folks are introvert (75%)
  - ISTJ is dominant: 24% (11%)
  - architects designers tend to be INTP or INTJ
    - 6% (4%)

Use for project manager

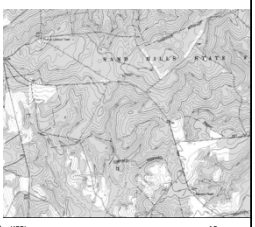
- Fit
- Mix
- Understand conflicts and frictions
- Adapt style, tasks, etc. to type
- Know oneself, and understand one's own interaction style
- Ethical issue?

Other tests?… lots!

- left brain – right brain
- visual – aural
- 9 types: enneagram test

Professionalism

- What's a profession?
- Software development as a profession
  - Body of knowledge
  - Education, curriculum, CPD
  - Experience
  - Standards of quality
- Government involvement
  - Protection of the public
  - Licensing
- Certification
- Code of Ethics



Profession

- Body of People, *who share a*
- Body of Specialized Knowledge
  - Body of Knowledge (BOK)
- Competence, *defined as both*
  - Education
  - Experience
- Rights and Duties *or Responsibilities*
- Code of conduct, code of ethics
- Advocacy, membership support
  - Altruism, protection, promotion, …

Not a new concept

- Craftsmen, master craftsmen, apprentices
- Guilds
  - Started in India
  - Grew in Europe in the middle ages
- Examples:
  - Builders (free masons!)
  - Carpenters
  - Architects
- Then:
  - Physicians ("college")
  - Lawyers ("the bar")
  - Academics ("the faculty")
  - ….

A *Software* Profession ??

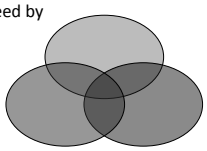
- Body of Knowledge:
  - Software Engineering Body of Knowledge (SWEBOK)
  - Can. Information Processing Society (CIPS) BOK
- Standard curriculum:
  - Computing Curricula 2005
    - CS + CE +SE +IS
- Code of conduct and ethics, *yes*
- Standards of quality, *mmm*
  - IEEE Software Engineering Standards Committee
  - ISO, ARINC, RTCA ….
- Professional organizations:
  - Informal, formal, academic, regulatory

Examples of professional organizations

- VanDev, VanLUG, VanQ, Agile Vancouver
- ACM, IEEE, INCOSE
- CIPS, PMI
- APEGBC, CCPE (now *Engineers Canada*)

Profession: the third logic? (E. Freidson)

- Rule of the free market
  - Products and services succeed by their own merit
  - Value driven most by cost
- Rule of the bureaucracy
  - Planning
- Rule of the organized professionals

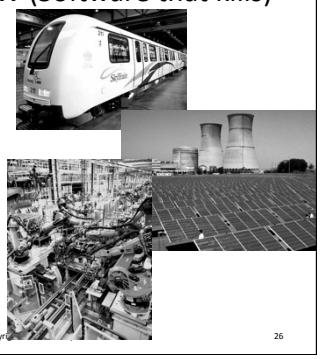


Profession (more)

- Risks to the public
  - Physical harm, death
  - Damage to environment
  - Massive financial losses
  - Crime
- Government involvement
- Self-regulated professions
  - Defines strict membership rules
  - Provides internal discipline
  - Accreditation of higher education programs

Safety-critical SW (Software that kills)

- Transportation
  - Avionics, ATC
  - Train
  - Cars
- Energy
  - Nuclear
- Biomedical instrumentation
- Industrial robots
- ….. etc



But also, mission-critical software

- Massive loss of money
  - Finance
- Potential for crime (fraud, tax evasion,…)
  - Sarbanes-Oxley, a.k.a. SOX (etc.)
- High liability

Standards of Quality

- Quality of process
- Quality of product
- Formal standards
- De facto standards

IEEE ISO Microsoft Office RATIONAL UNIFIED PROCESS W3C

Membership and Certification

- Adding to a university degree
- Currency of knowledge or highly specialized
- One time certifications, or periodic renewal
- Examples:
  - CIPS ISP
  - IEEE CSDP
  - PMI PMP, PgMP
  - CCPE P.Eng.
- And many others:
  - ScrumMaster,
  - Microsoft and other vendors
  - Dozens in quality: CSQA, CSQE, etc. etc.

Four examples

I.S.P. IEEE Computer Society Certified Software Development Professional csdp PMP PMI Project Management Professional APEG Professional Engineers and Geoscientists of BC

Certification

- Sort out the good ones from the bad
- Tough standards, not ease of exam
- Recognition, not just cost

- Laudable personal goal
- Help make a (small) difference in a career
- In some circumstances, may be a requirement for employment, or contract

Ethics

- What is right, and what is wrong?
- What is just?

- Are there bounds to what we can do?
- Who defines such bounds?

- Aren't laws enough?
- Should we trust our intuition?

Software engineering code of ethics

1 PUBLIC - Software engineers shall act consistently with the public interest.
2 CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer and that is consistent with the public interest.
3 PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4 JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
5 MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6 PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7 COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.
8 SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and promote an ethical approach to the practice of the profession.

See long form on web site

Source: IEEE/ACM

Highlights

- Act responsibly
- Confidentiality
- Respect of intellectual property
- Conflict of interest
- Protection of the public, of the environment
- Fairness
- Respect of peers, of other parties
- Keeping up to date: CPD = Continuing Professional Development

FPE (Frequently Proffered Excuses)

- Everybody does it. They get away with it.
- I did not do it (me, personally).
- I would have lost my job…
- I was in a hurry.
- It passed all the tests.
- We said in the disclaimer that it could go wrong.
- Does this apply in their country, too?
- Nobody had told me this at work (nor at school).
- What about my freedom of speech?
- The customer is always right, right?

Ethical dilemmas, conflicts

- Freedom of speech vs. protecting children

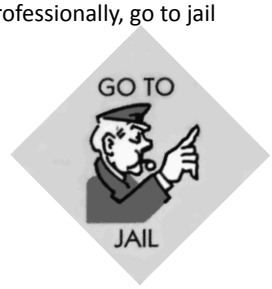
- Shareholder *vs* customer
  - On time, but flawed
  - Late, but correct

Making ethical decisions

- Is it Legal?
  Bell, book and candle test
  Will I be violating any civil law or institutional policy?
- Is it balanced?
  Is it fair to all concerned?
  Does it promote win/win situations?
- How will it make me feel about myself?
  Will I be proud?
  Would I feel good if my hometown newspaper published my decision?
  Would I feel good if my family know about my choice?

When things really go sour…

- Act unprofessionally, go to jail



Continuing Professional Development

- Very fast turn around of technologies
  - 5 years
- Learn how to learn & Learn continuo
  - Focus on the fundamentals (e.g., discrete
- Read books
- Read trade magazines
  - Paper or online
- Attend conference, seminars, etc.
- Join a couple of professional groups

YOU are the profession

- Get involved
- Participate
- Volunteer
- Shape it

- Don't be just there sitting, moaning and bitching on the side, waiting to be told what to do…

Summary

- Professionalism
- Behave ethically
- Keep your competence on top
- Aim at certification
- Contribute

Further reading

- S. McConnell, *Professional Software Development,* Pearson, 2004.
- E. Krause, *Death of the Guilds*, Yale U. Press, 1999
- IEEE-ACM *Software Engineering Code of Ethics and Professional Practice*, Long version, 5.2, 1999
- E. Freidson, *Professionalism, the Third Logic: On the Practice of Knowledge,* U. Chicago Press, 2001

16: Managing external stakeholders

Software Project Management
Philippe Kruchten

Module outline

- Software Acquisition
  - Supplier, acquirer
  - Acquisition process
- Standards
  - PMBOK
  - IEEE 1062: 1998
    - Recommended practice for software acquisition
  - Progressive acquisition in RUP
  - ISO 12207

Definitions

- Acquisition
- Procurement
  - the process of acquiring goods or services
- Acquirer
  - syn.: buyer, customer
- Supplier
  - syn.: seller, vendor, contractor (service), subcontractor

Definitions (Cont.)

- COTS = Commercial Off-The-Shelf
  - software defined by market driven needs, available on the open market, user by a spectrum of commercial users.
- MOTS = Modified Off-The-Shelf
- RFP = Request for proposal
- RFI = Request for information

Why acquiring? What to acquire?

- Make or buy
- Whole or part
- Buying help or buying product
- Single supplier: added risk?
- Tailor made or off-the-shelf

- Outsourcing

PMBOK

- Section 12: Project Procurement Management
  - Procurement planning
  - Solicitation planning
  - Source selection
  - Contract administration
  - Contract close-out

Typical sequence of actions (IEEE 1062)

Phase	Start	End	Steps
Planning	Idea	RFP	Set up organization Determine requirements
Contracting	RFP issued	Sign contract	Identify suppliers Prepare contract Select supplier
Implementation	Contract signed	Receive Software	Manage supplier performance
Product acceptance	Software received	Accept software	Accepting the software
Follow-on	Software Accepted	Product no longer in use	Usage period

Issues with traditional SW acquisition

- Can't express your needs up front
- Technology changes over time
- Large and complex systems (of systems)
- Can't replace all systems at once

- Loss of visibility during implementation
- Lack of expertise to control implementation

Major variables affecting SW acquisition

- Degree of customization
  - COTS
  - Partly customized
  - Entirely custom
- Scale of acquisition
  - Single components – full system
- Pricing, contract type
  - Fixed price (or lump-sum)
  - Price or cost with incentives
  - Time and materials

Contract

- Mutual consent
  - Open, no pressure, certain, visibility
- Lawful objective
  - Purpose and terms must fall within the law
- Capacities to perform
- Consideration
  - what is being promised by both parties
- Form

Issues with contracts for software

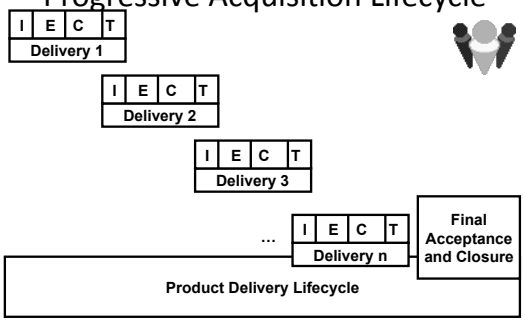
- Best value at least cost
  - Very risky, as cost easier to assess than value
- Adversarial relationships
  - love affair until contract signed
- Bias in flavour of the contract authors
- Obscure language
- Mystery "standard" clauses
- More than 2 parties
  - supplier, acquirer, and users
- Fixed price

Progressive Acquisition

- Similar to iterative development
- Do things in steps
- Driven by risk management strategy

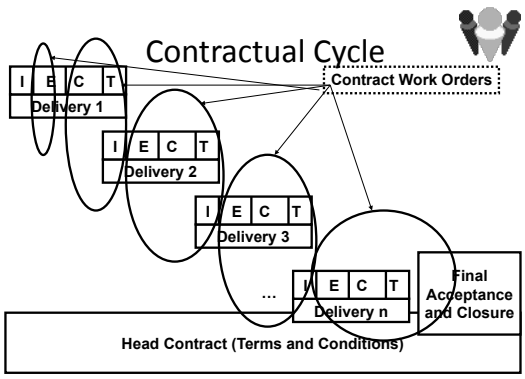
- Staged payments and deliveries
- 1 master contract or head contract
  - Several subsidiary contracts, or contract work-orders

Progressive Acquisition Lifecycle



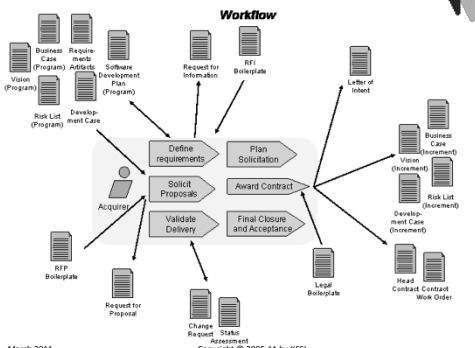
March 2011 Copyright © 2005-11 by KESL 13

Contractual Cycle



March 2011 Copyright © 2005-11 by KESL 14

PA in RUP



March 2011 Copyright © 2005-11 by KESL 15

Activity: Define Requirements

- Standard RUP Activities and Workflows
  - Business Modeling
  - Develop Business Case
  - Analyze the Problem
  - Architectural Analysis
- Build versus Buy Decision
- Structure Contracting Program

March 2011 Copyright © 2005-11 by KESL 16

Activity: Plan Solicitation

- Update Needs and Risks
- Analyze Market
- Select Best Contract Form
- Establish Selection Process
- Issue RFI (Optional)
- List Suppliers to Solicit

Copyright © 2005-11 by KESL 17

Activity: Solicit Proposals

- Prepare Request for Proposal
- Clarify Request for Proposal
- Analyze Proposals

March 2011 Copyright © 2005-11 by KESL 18

Activity: Award Contract

- Notify Successful Supplier
- Joint Delivery Definition
- Negotiate Head Contract
- Negotiate Contract Work Order
- Provide Supplier with Resources

Activity: Validate Delivery

- Ensure Contract Terms Met
- Request Contract Adjustments
- Address Supplier Claims
- Obtain Sign-off
- Manage Final Payment
- Close Contract Document
- Contract Close-out Review
- Archive Contract Information

Contract Types

	Fixed Price	Cost reimbursable
Form of payment	▪Firm lump-sum	▪Cost plus percentage of cost ▪Cost plus fixed fee
Form of incentive	▪Fixed price plus incentive fee	▪Cost plus incentive fee

- Incentives to improve
  - Delivery schedule
  - Costs
  - Technical performance
  - Reliability, defect level
- ▪ Penalties
  - Delays
  - Quality
- ▪ Warranties

Project Manager and Acquisition

- Sounds easier
- Is harder
  - On both side of the divide
- Interpersonal, inter-organizational issues
- Document based
- Distrust
- More roles involved (law, finance)
- Immaturity of one party
- Cultural divide (language, distance, ethnologic culture, corporate culture)

Program Management

- Project of projects
- System made of systems
- Objective: “divide and conquer”
- Program manager
  - several project managers
  - sometimes common functions
- Two level management, one order of magnitude (10x) more complex.

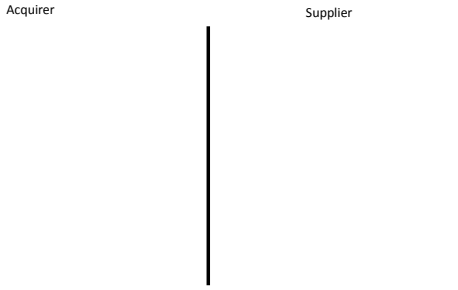
Outsourcing, offshoring

- Parts or all of a SW project done somewhere else
  - India, Brazil, Thailand, Ukraine, Poland…
  - Russia, Hungary, China, Baltic countries, Balkan countries…
- up to one order of magnitude difference in salaries, for equivalent expertise
- Push to level 5 certification (hum, hm,…)

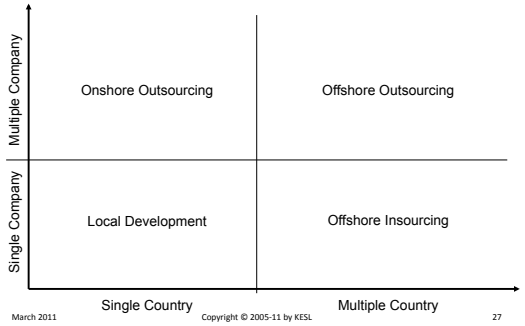
Offshoring

- Organization
  - Acquirer
  - Small supplier beachhead acting as a relay
    - In the vicinity of the Acquirer
    - Communication funnel
    - Facilitator
    - Intercultural "translator" (bi-coded individuals)
  - Supplier

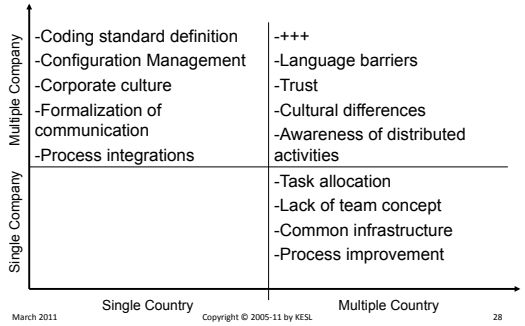
Offshoring: where to cut?



GSD Organizations



Challenges



Review

- Procurement = acquisition
  - What to acquire, from whom, why
- Multiple interfaces -> Increased complexity
- Reliance on written artifacts
- Proceed iteratively, if possible
  - RFI, RFP, Contract
    - Head contract, contract work orders
- Program Management
- Project Portfolio Management

What would Octopus say?

17: Managing the software development process

Software Project Management
Philippe Kruchten

Module outline

- Selecting a process
- Adapting the process
- Software Development Process
  - SEPG = Software Engineering Process Group
- Change management
- Feedback loop
  - Retrospectives
  - Process related measures and indicators
- Software Process Improvement

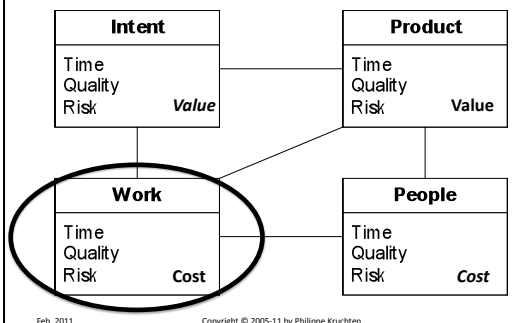
Examination header

- This examination is made of 7 independent questions. You may answer them in any order, but indicate clearly the question number.
- They are "short essay" questions. Be concise: answer the question, and no more. Do not regurgitate all the course material that may seem to be vaguely related.
- When in doubt, make assumptions, state your assumptions.
- Write legibly, and use other paper as draft or thinking pad or doodle board.
- Put your name on each leaflet.
- You only have 2 h 30 min, that is, 21 min/question, so think about "time-boxing".
- You do **not** need a fancy calculator: only a few additions, multiplication and divisions, 3 digits.

Software Development Process

- Lifecycle
  - Iterative
  - Phase and milestone
- Technique, methods
  - Object-oriented, UML, etc..
- Prescribed artifacts/workproducts
- Adequate tool support
- Common language/team culture

Process = Template for Work



Process = Template for Work

- Do not re-invent the wheel
- Good practices
- Re-usable chunks:
  - Check lists,
  - Artifact / document templates
  - Elements of your SDP
- Sequences of typical activities
- Process quality (ISO 9126)

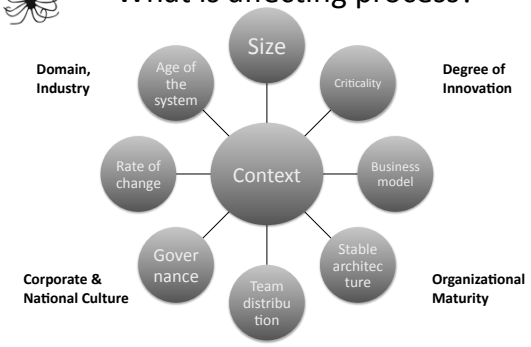
SEPG = Software Engineering Process Group

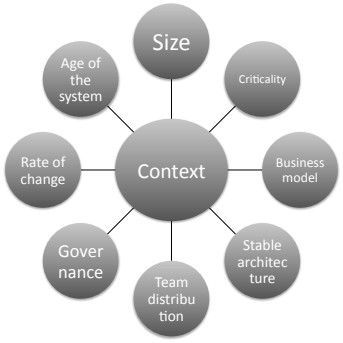
- Who is in charge of the process?
- Small project: project manager
- Sometimes: quality assurance
- Danger: some external group not having anything at stake in the project
- Often: a small group inside the project, with or without external help: SEPG
- In the end: Process is everyone's business

Selecting a process

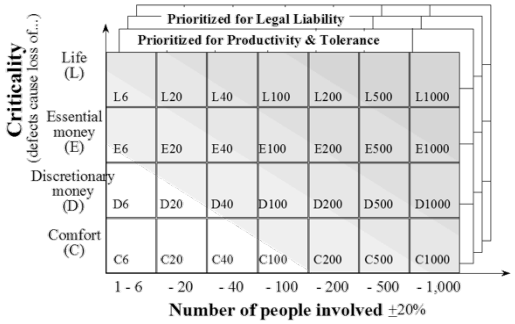
- How much process do you really need?
- Degree of Ceremony?
- Agile vs. Plan-driven
- Documented (by reference) in Software Development Plan

What is affecting process?





Cockburn's table



Adapting the process

- No "one size fits all"
- Select base process
  - Size of project
- Select lifecycle
- Select artifacts
  - Need of communication
  - Need to comply to external standards
- Select methods, the practices
- Select tools to match the methods
- Provide training

Change Management

- Hard to change the way people work
- Change in small increments
  - Not everything at once
- Change = Pain + Desire
- Need commitment from all parties involved
- Need proper training, and ways to experiment at low risk

Change Management

- Change is often perceived as a threat
- Leads to "emotional resistance", fear
- Needs to be overcome by a stronger emotion: desire
- But the initial changes better be good, or they would condemn for a long time any future changes

Hard Changes

- Change of lifecycle model
  - Waterfall to Iterative
- Change of size, up or down
  - People "port" their habits and expectations
- Change in context
  - In house to acquisition (or vice versa)
- Change in the reward model
  - results, not participation
- Change of technology
  - Easiest to alleviate with training (and good tools)

Feedback Loop

- Often: stop and think
  - Once per iteration at least
  - Are we doing the right thing?
  - Are we doing the thing right?
  - Can we do better?
  - Use indicators:
    - Quality (defect analysis)
    - Issues and risks getting stale (open for long time)
    - Productivity (velocity), Burn down chart
    - Delays, pushing objectives down the time line
- Post-mortem, retrospective, debriefing
  - capture lessons learned

Retrospectives

- The Check-Act part of PDCA
- Neutral facilitator
- 1 to 6 hours (off-site?)
- Reality-based, collective learning experience; results in actions
- Bring in project data
- Acknowledge that feelings do count too
- Focus on "it" not "him" or "them"

"Prime Directive"

Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.

Source: Norm Kerth

- Everyone makes mistakes; the Prime Directive reminds us to support, not attack, our colleagues.

Four Key Questions

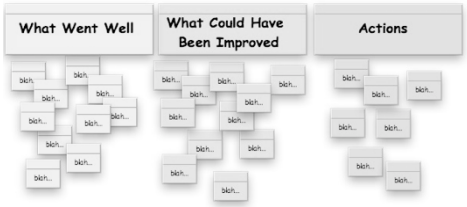
- What did we do well, that if we don't discuss we might forget?
- What did we learn?
- What should we do differently next time?
- What still puzzles us?

Source: Norm Kerth

Retrospectives

- Have participants brainstorm ideas in six categories: enjoyable, frustrating, puzzling; same, more, less.
- Write each idea on a card or post-it note
- Next, place the cards on a board. Group the cards that look similar. Everybody participates; nobody speaks.
- Circle and name the resulting categories. Choose one, then brainstorm root causes and solutions. Pick one: it's your retrospective objective. Follow through in the iteration to come.
- When retrospectives get boring, try other formats. This is just a starting place.

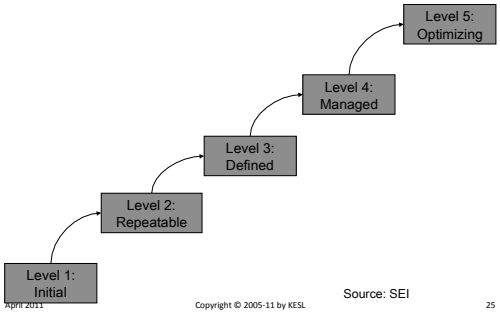
Source: Jim Shore



Software Process Improvement

- Project or Enterprise-wide initiative ?
- Getting better:
  - Quality
  - predictability
  - Efficiency
  - Time-to-market)
- CMMi certification
- ISO 9000 certification
- A project in itself

Capability Maturity Model: the Ladder



Source: SEI

Summary

- Process and people are key in software development
- Picking the right process
  - Adapted to the project
  - Lean and mean
  - Avoid “software bureaucracy”
- Improve
  - Build explicit feedback loops (pdCA)

18: Software Development Governance

Software Project Management
Philippe Kruchten

Module outline

- Definitions, concepts
- Governance across a whole organization
- Software development governance
- Simple governance framework
- Practices, tools, standard
- Governance and management

Governance

- Governance is a process intended to ensure that the goals and values of the software development project are aligned with the goals and values of the organization it belongs to or it serves.

- Although project teams are largely on their own, they are not uncontrolled. Management establishes enough checkpoints to prevent instability, ambiguity, and tension from turning into chaos. At the same time, management avoids the type of rigid control that impairs creativity and spontaneity. Instead, the emphasis is on 'self-control', 'control through peer pressure' and 'control by love'.

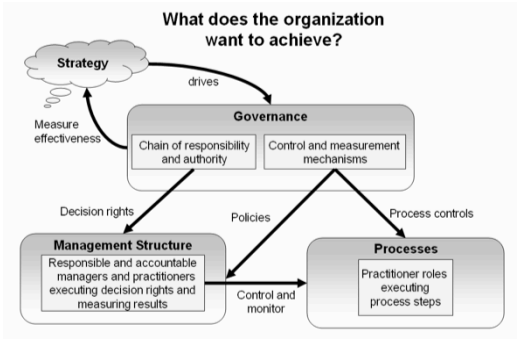
Takeuchi & Nonaka, 1986

Governance, *functionally*

- Governance defines goals, values, objective of the organization that should be used to make decisions
- Governance decides whom is empowered to make this or that kind of decision, based on what kind of criteria and information
- Governance ensures that all parts of the organization are actually aligned with these goals, values and objectives

Governance, *structurally*

- What are the bodies, boards, committees, teams, projects, people, etc. involved?
- How do they communicate?
- When do they communicate? How often?
- Chains of authorities
- Measures, controls, mechanisms, policies to enable all the actors to carry out their respective responsibilities



Source: Sunita Chulani *et al.* IBM 2006

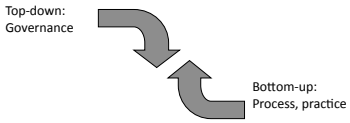
Governance

- How do project starts and stops?
- What happens when things go wrong?
- Who defines "success" or "failure"?

- Goals:
  - Manage value (e.g., backlog co-management)
  - Develop flexibility (e.g., iteration and retrospective)
  - Control risks and changes (e.g., Daily huddle)

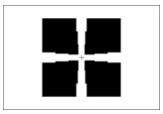
Governance

- Environmental pressure that forces natural selection of the good enough set of practices (and hopefully abort non-viable variants)

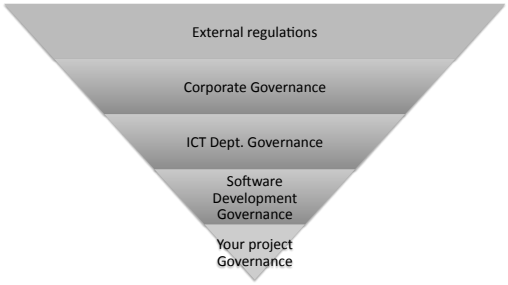


Software Development Governance

- How does all this apply to software development?
- Is your project aligned with the goals and objectives of the (enclosing) organization?



Nested levels of governance



Issues

- Who decide about the life and death of a project?
- Who defines "success" and "failure"?
- Who manages the project managers?

Example

- John is the president of you software development company
- John wants to know about your project:
  - Is it worth doing? ROI? Useful?
  - Is it progressing well? Risks and uncertainties under control?
  - Not squandering resources, time money or other?
  - Should the effort and money be invested in something else, more useful?
  - Are you the right person to manage this project?
- "Trust me, John." may not be quite sufficient

Example (2)

- John has another governance process to follow, involving the shareholders, the board of directors, a government agency, etc.
- How can John proceed?
  - Command-and-control
  - Software police (distrust)
  - Huge "One size fits all" processes
  - Bureaucracy

Example (3)

- Alternative:
  - Mechanisms, incentives, processes that will encourage everyone (you included) to behave and work in a manner aligned with the objectives of the corporation
  - Minor checks in place, lightweight mechanisms and measures, dashboards

Compliance

- Adhering or conforming to a rule, specification, policy, standard, regulation or law
- Internal, external

Transparency

- Property of an organization that implies openness, communication, integrity, accountability
- No secret little deals, small lies, or different information given to diverse parties

Simple Governance Framework

- Structural facet
  - Defines decisions rights: who can decide about what
  - Empowers people to make decisions
  - Chains of responsibilities, reporting structure
- Dynamic facet
  - Policies (rules, procedures, guidelines), measurements, controls to enable people to carry on their responsibilities

Apache Software Foundation

- Static facet
  - User
  - Developer
  - Committer
    - Access to the repository; has singed a CLA
- Plus in the Project Management Committee
  - PMC member
  - PMC chair

Source: Clay Williams, IBM 2008

Apache Software Foundation

- Dynamic facet:

  Policy "To place a new committer on a project"

  This policy is enacted by the PMC, and has the following steps:
  - Vote in the new committer
  - Ensure committer signs the CLA
  - CLA must be acknowledged by ASF secretary or board member
  - Create new account for the committer
  - PMC chair gives new committer access to project repository

Tactics

- Governance body
- Governance scope
- Governance goals
- Governance mechanisms
  - Information flowing up
  - Decisions or changes flowing down
  - Incentives
  - Building a local culture

Practices supporting governance

- Iterations
- Risks, uncertainties, assumptions
- Cost estimates, expenditures
- Progress tracking:
  - burn down chart, scrum board, Kanban board
- Defect trends
- Quality metrics
- Retrospectives
- Daily stand-up meetings
- Etc.

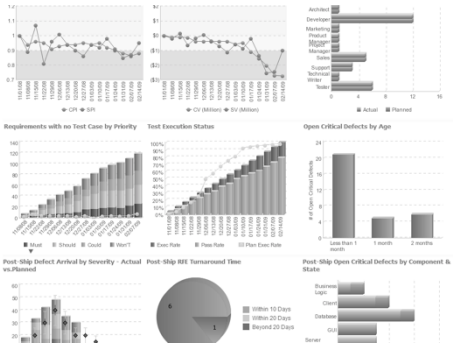
Practices by type

Belief systems	Boundaries
Shared vision Values Team charter Sense making	Scope Decision rights Taboos Standards
Diagnostic measures	**Interactive measures**
Budget Velocity Backlog length Story walls	Stand-ups Showcases Retrospectives Steering committees

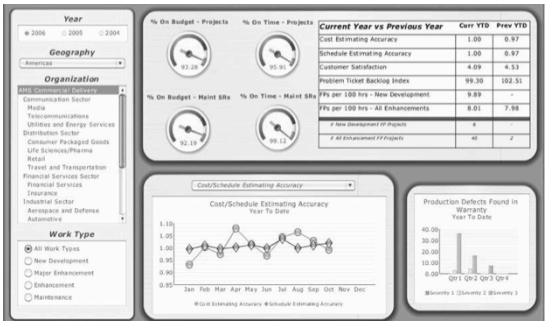
J. King, based on R. Simons 1995

Source: Alan Brown, IBM, 2010

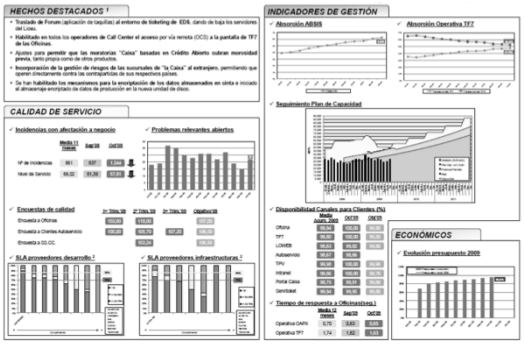
Dashboard



Source: Alan Brown, IBM, 2010



Source: Alan Brown, IBM, 2010



Standards, etc.

- CoBIT, *IT Governance Institute(2005)*
  - Control Objectives for information-related technology
- ITIL, *UK Government*
  - Information Technology Infrastructure Library
- OPM-3, *Project Management Institute*
  - Organizational Project Management Maturity Model

Governance *vs*. management

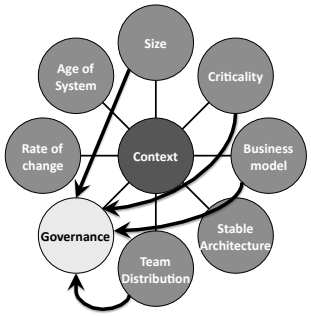
- Touch points, identical vocabulary
- Governance
  - is outside of the project
  - Looks down into the project
- Governance as "metamanagement"
- Self-managed team, not self-governing teams
  - Case of the small startup?

External governance constraints

- Law
- Regulations specific to a given domain:
  - 21-CFR-11
  - Basel 2, 3
  - Sarbanes-Oxley act of 2002 (SOX)
  - DO178B, RTCA
  - ….
- Bring additional governance down the chain
  - Rules, policies, …

Context attributes

1. Size
2. Criticality
3. Age of system
4. Rate of change
5. Business model
6. Stable architecture
7. Team distribution
8. Governance

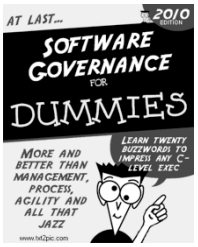


Summary

- Governance: a process to ensure alignment of objectives, and values
- A nested concept
- Metamanagement
- Static aspects:
  - body, scope, chains of authorities and responsibilities
- Dynamic aspects:
  - procedures, policies, measures, etc.
- All projects have some form of governance
  - Often not made explicit, or improvised

Buzzword bingo

- Governance
- Alignment
- Contingency
- Chain of authority
- Compliance
- 21-CFR-11
- Transparency
- Dashboard
- Empowerment
- SOA
- SOX
- EA
- Body
- Risk
- Board
- CoBIT
- ITIL
- Basel 2



19: Managing a portfolio of projects

Software Project Management
Philippe Kruchten

Module outline

- Still to be done

Portfolio Management

- Relatively independent projects
- Objective:
  - Optimize resources and return across organization
  - Align with organization goals and objectives
- Need comparable performance metrics….
- =/ Program management

Portfolio Management

- Business units are territorial
- Pet projects and slush funds
- Prioritization
- Criteria:
  - Business goals, business cases
  - Core services and products
  - Risks

Portfolio Management: Benefits

- Right mix: maximize overall return
- Balance risks
  - like investment portfolio…..
- Balance resource allocation
  - danger!
  - need slack
- Oversight
- Identification of synergies, overlaps

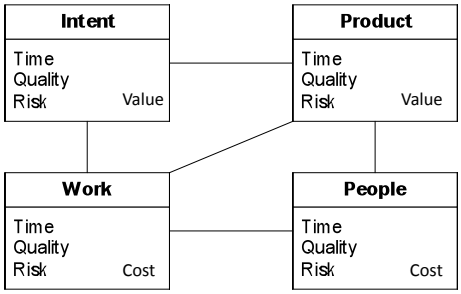
20: Wrap up

Software Project Management
Philippe Kruchten

Module outline

- Back to the conceptual model and the octopus
- Key questions
- Role of the software project manager

The review

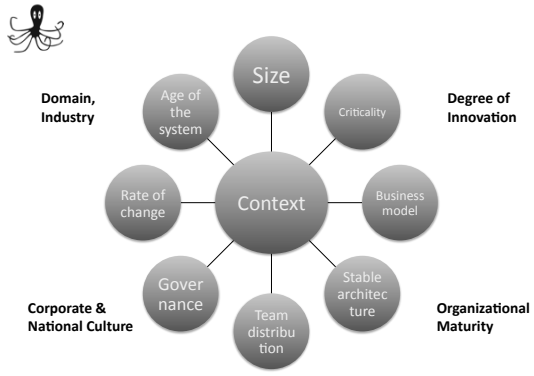


Key Questions

- When will we be done?
  - Time, effort, work
- How much is it going to cost?
  - Cost, effort, people
- Will they like it?
  - Value, quality

What does the project manager do?

- Role
- It depends…
- On the context



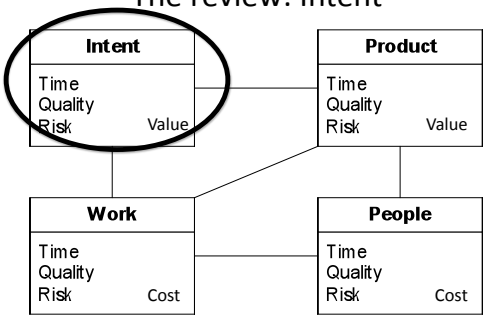
Where does your project fit?

- For each of the 8 factors, where does your project fit?
- How is this going to impact the practices, tools, documents, etc. that you will use?

What drive complexity?

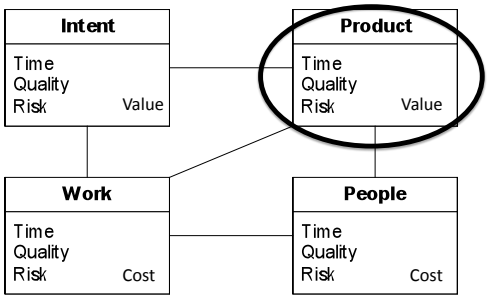
- Number of *things*
- Diversity of *things*
- Interdependence between *things*

The review: Intent

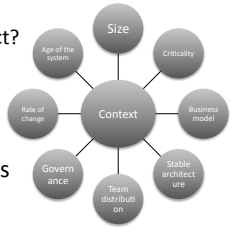


- What shape does the intent take?
  - Use case, road map, feature list, …
  - Quality, precision, assumptions
- Who defines intent?
- How does intent evolve?
- How do you control scope creep?

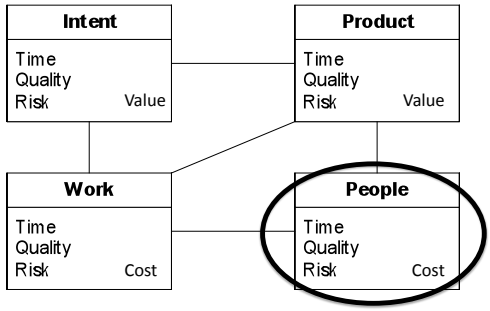
The review: Product



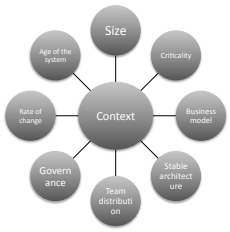
- What do you deliver?
  - What is a complete product?
  - What does "done" mean?
- Product roadmap
  - Release backlog
- Components, technologies
- Assets



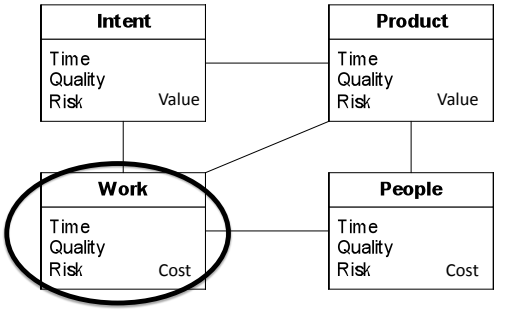
The review: People



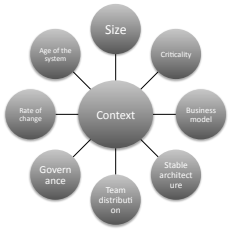
- Who is the team?
- Where are they?
  - Distribution, culture
- Competencies?
- Collaboration, trust
- Task allocation



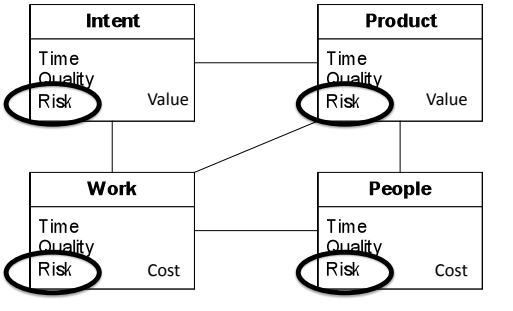
The review: Work



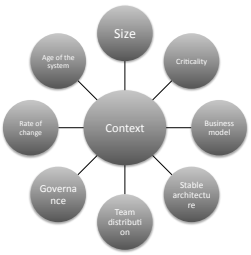
What is our process?
Practices, tools, methods,
What is our lifecycle?
Phases, milestones, iterations
How do we represent it
Gannt chart, scrum board, kanban, …
iteration backlog



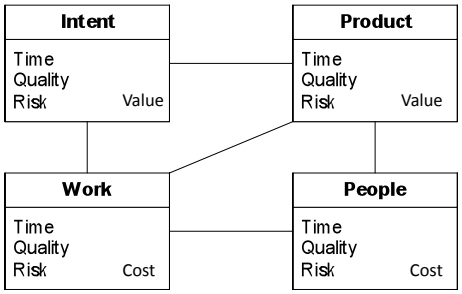
The review: Risk



- How do we define success?
- What is known, and unknown?
- What is uncertain?
- How do we document assumptions?
- How to we resolve uncertainty?
- What are our risks?



The review: Value and Cost



SW: A Different Management

- Lifecycle is iterative and incremental
- Activities are parallel and concurrent, not phased
- Planning done around deliverables (features), not activities
- Adjustments in scope (time boxing, priorities)
- Collaboration between stakeholders, and among team members
- Facilitation: project manager as a facilitator

Role of the Project Manager

- Multilingual
  - Ability to translate between stakeholders
- Gatekeeper
  - Ability to control or even reduce scope
- Maestro
  - Ability to have all stakeholder work together, collaboratively
- Cattle driver
  - Ability to keep the project moving forward
- "Clark Kent"
  - Ability to articulate clearly objectives, requirements and progress

  Source: Standish Group