PDC2008
PROFESSIONAL DEVELOPERS CONFERENCE

agile perspectives

→ Mary Poppendieck
Poppendieck L.L.C

→ Grigori Melnik
Microsoft Corporation

(USE THIS SPACE FOR PRODUCT LOGOS WHEN WHITE BACKGROUND IS REQUIRED)

DELETE WHITE RECTANGLES IF NOT BEING USED

Plan for the day

- 10:00-10:30 Intro – agile tour
- 10:30-12:00 Reality of agile – Industry perspective
- 13:15-14:15 Reality of agile – Microsoft/p&p perspective
- 14:15-16:30 Staged discussion – 4 AtIssue topics
- 16:45-17:45 Agile Unleashed panel

How will this workshop help you?

- Identify areas for improvement on your projects
- Think about ways to make those changes
  - By explaining how *we* do it
- Help you think about things in a multitude of new ways
  - Change your frame of reference
  - Understand key principles of agile teams

*The market would consist of **specialists in system building**, who would be able to use tried parts for all the more commonplace parts of their systems… The ultimate consumer of systems based on components ought to see considerably improved **reliability and performance**, … and also to **avoid the now prevalent failings** of the more mundane parts of systems, which have been specified by **experts**, and have then been written by **hacks**.*

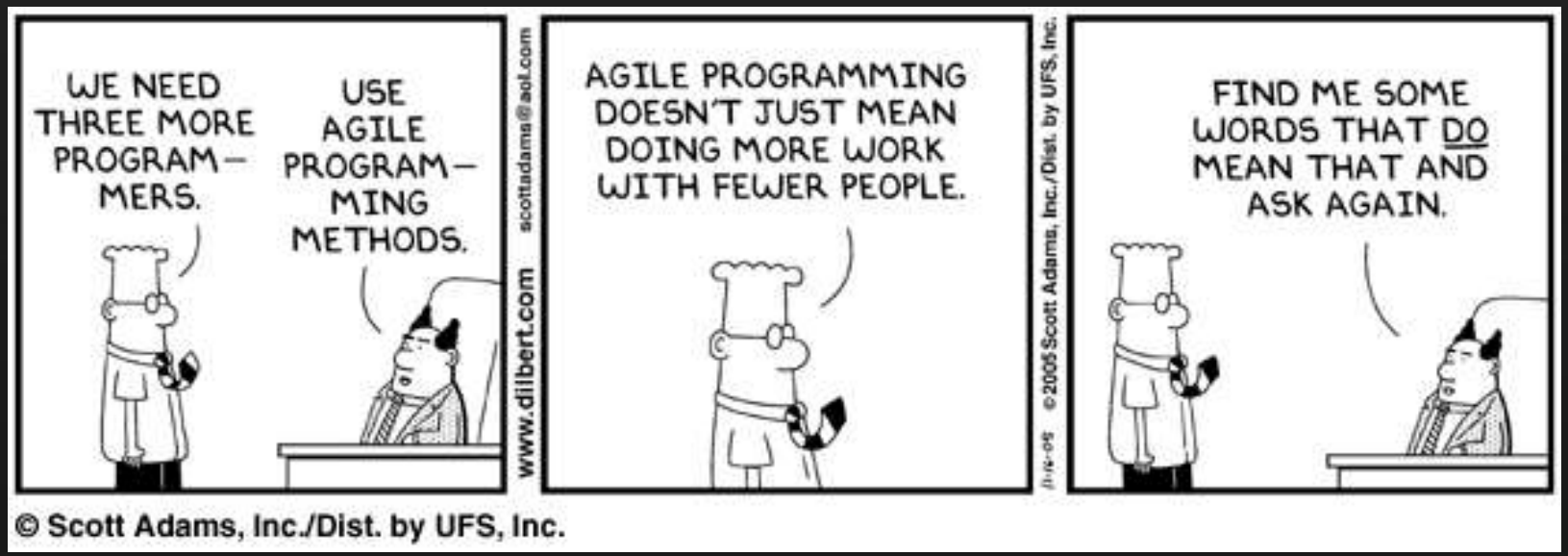
Taylorism/Fordism

- Basic principles
  - Standardized products
  - Repeated tasks having potential for automation
  - Unautomated tasks analyzed using work study methods
    - work is extremely task focused
    - work is specialized with divisional labor
  - Production lines with the work moving to the workers
    - focus on **repeatable process!**

2001

Agile Mindset

Continuously delivering value by self-organizing teams in the face of changing requirements



Retrospective Tour

- 2000-2001 Suitable contexts
- 2002 Scalability
- 2003 Adaptability
- 2004 Methodologies zoo
- 2005 Convergence
- 2006-2007 Entering mainstream
- 2008-… Agile v2?

Reality of Agile: Industry Perspective

- What Works, What Doesn’t, and Why
- Mary Poppendieck

What Works – What Doesn't

What Works

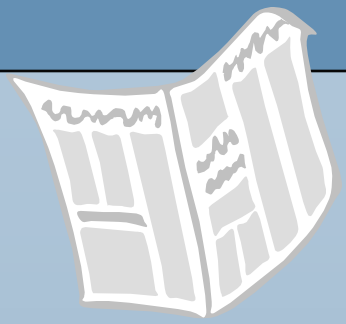
1. Technical Practices
2. Small Batches
3. Pull Scheduling
4. Focus on Learning

What Doesn't

1. Complexity
2. Handoffs

lean

A Lesson From Our History



1972: New York Times Information Bank

Structured Programming

Edsger Dijkstra: ***[Quality by Design.]*** The quality of the product can never be established afterwards. Whether the correctness of a piece of software can be guaranteed or not depends greatly on the structure. … Testing is a very inefficient way of convincing oneself of the correctness of a program.

Dave Parnas: ***[Information hiding.]*** Divide program into modules based on their responsibility and the likeliness of future change, not on order of flow.

Top Down Programming

Terry Baker: ***[Continuous Integration]*** An evolutionary approach to systems development….integration is completed parallel with, rather than after, unit coding….As a result, acceptance and system testing have been nearly error free.

A Lesson From Our History

Chief Programmer Team

1. *Lead & Backup pair – responsible for designing and programming the system.*
2. *Both deeply involved in design and programming. Review each other's work.*
3. *Lead programmer supervises other programmers and reviews their work.*
4. *Backup capable of taking over for lead. Continually tests the system.*
5. *Library – repository for all code. Common code ownership.*

[Technical Leader, Pairing, Common Code Ownership]

Results of New York Times Project

100+ times more productive and higher quality than typical at the time:

- ✓ 10,000 LOC and one detected fault per person-year (83,000 LOC total)
- ✓ 21 faults in acceptance testing; 25 further faults in the first year of operation

BUT – Resource Management Prevails

Because we cannot depend on the existence of a super-programmer in any project and because we must transfer our people around in order to apply the right number of resources at the right time to a contract, we have found that it is important to build our systems without dependence upon any particularly strong individual.

– J.D. Aron: 1969 NATO Software Engineering Conference, Rome

What is Software Engineering?

1976: Software Engineering – Barry Boehm

The Problem

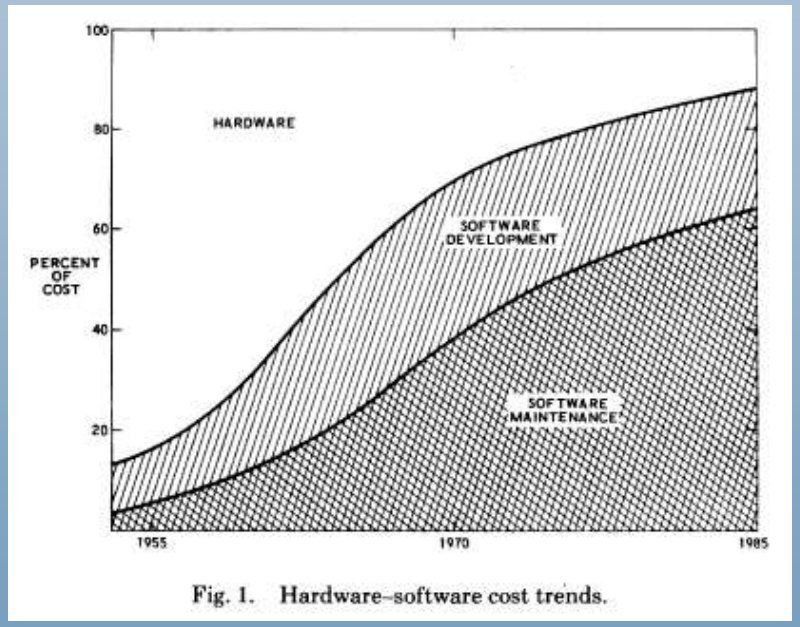


Fig. 1. Hardware–software cost trends.

IEEE Transactions on Computers

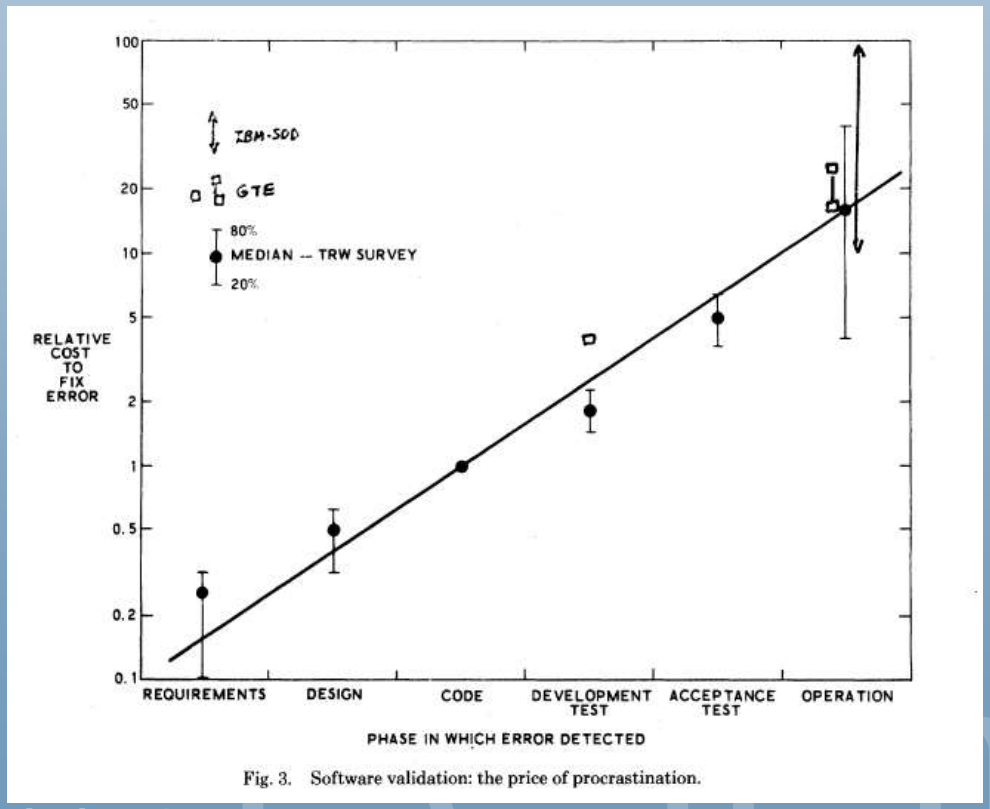


Fig. 3. Software validation: the price of procrastination.

What is Software Engineering?

1976 – Barry Boehm:
The Solution:

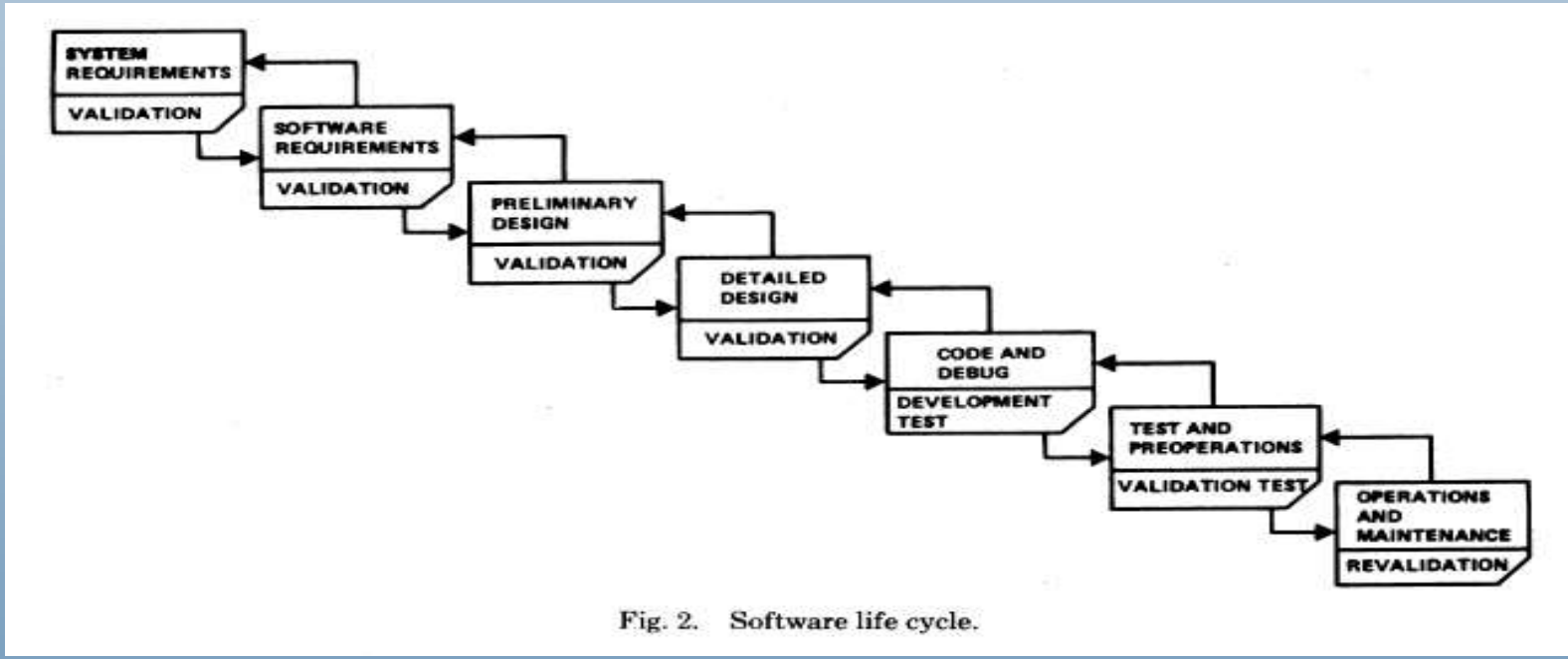


Fig. 2. Software life cycle.

Footnote:

At a panel discussion at ICSE 2007 in Minneapolis, Barry Boehm discussed his exponential cost model, under fire from Tom DeMarco. Barry said that completing requirements before proceeding worked well for large government projects in the 1970's, but the approach proved inappropriate for later projects. However, people were so "brainwashed" by his earlier work that the "lifecycle" solution could no longer be questioned.

What is Software Engineering?

1982: Life Cycle Concept Considered Harmful

Daniel McCracken & Michael Jackson

– ACM Software Engineering Notes, April 1982

1. Any form of life cycle is a project management structure imposed on system development. To contend that any life cycle scheme, even with variations, can be applied to all system development is either to fly in the face of reality or to assume a life cycle so rudimentary as to be vacuous.

2. The life cycle concept perpetuates our failure so far, as an industry, to build an effective bridge across the communication gap between end-user and systems analyst. In many ways it constrains future thinking to fit the mold created in response to failures of the past.

3. The life cycle concept rigidifies thinking, and thus serves as poorly as possible the demand that systems be responsive to change. We all know that systems and their requirements inevitably change over time. ....

To impose the concept on emerging methods in which much greater responsiveness to change is possible, seems to us to be sadly shortsighted.

Let us Not Confuse:

Technical Excellence	**Project Management**
Low Dependency Architecture	***Complete Requirements***
Quality by Design	***Quality by Testing***
Technical Disciplines	***Maturity Levels***
Respect for Complexity	***Scope Control***
Skilled Technical Leaders	***Resource Management***
Learning / Feedback Cycles	***Timeboxes***
Success = Accomplishing the System's Overall Objective	***Success = Achieving Planned Cost, Schedule and Scope***
Robust Over Time	**Fragile Over Time**

Quality by Design

A Quality Process Builds Quality IN.

- ✓ Rather than trying to test quality in later.

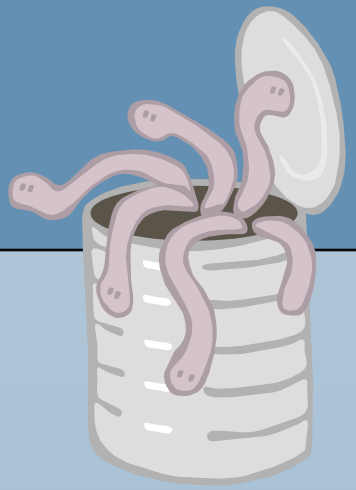
Where do we get the idea that it is okay to find defects at the end of the development process?

- ✓ *There is extensive evidence that finding and removing defects the moment they occur leads to dramatic quality and productivity improvements.*

Quality by Design

- ✓ Code that reveals its intentions
- ✓ Design/code reviews
- ✓ Immediate, automated testing
- ✓ Continuous, nested integration
- ✓ Escaped defect analysis & feedback

Application Testing



Continuous Integration	Test to Specification	Test to Failure
Product Design	Functional Tests	Exploratory Testing
Interaction Design	Presentation Layer Tests	Usability Testing
Technical Design	Unit Tests	PSR / -ility / Stress Tests

Manual: As early as possible

Manual: As early as possible

Tool-Based: As early as practical

Automated: Every Day

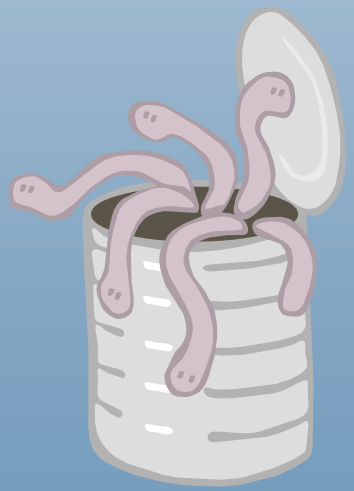
Automated: Every Build

Test Layer Separately

System Testing / UAT

Frequent Integration	Test to Specification	Test to Failure
Before Deployment	Regression / End-End Tests	PSR* / -ility / Stress Tests
After Deployment	Escaped Defects	Failure Scenarios

Tool-Based: Every Iteration



Automated: Every Iteration

Analyze cause of every defect

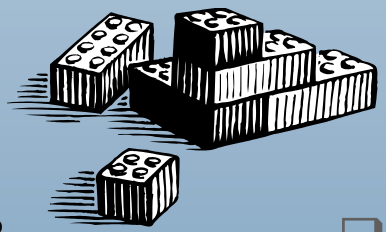
Planned Fault Injection

*PSR = Performance, Security

Building Block Disciplines

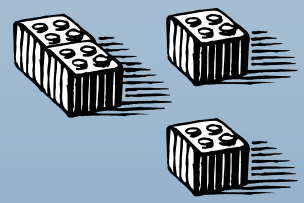
Mistake-Proofing

- Design/Code Reviews
- Configuration/Version Management
- One Click Build (Private & Public)
- Continuous Integration
- Automated Unit Tests
- Automated Functional Tests
- System Testing with each Iteration
- Stress Testing (App & System Level)
- ***STOP*** if the tests don't pass
- Automated Release / Install Packaging
- Escaped Defect Analysis & Feedback



Simplicity

- Architectural Standards
- Development Standards
  - Naming
  - Coding
  - Logging
  - Security
  - Usability



- Standard Tools
  - IDE's
  - Code Checkers
  - Configuration Management
  - Build/Test Harnesses
- Refactoring
  - Continuous Improvement of the Code Base

lean

What Works – What Doesn't

What Works

1. Technical Practices
2. Small Batches
3. Pull Scheduling
4. Focus on Learning

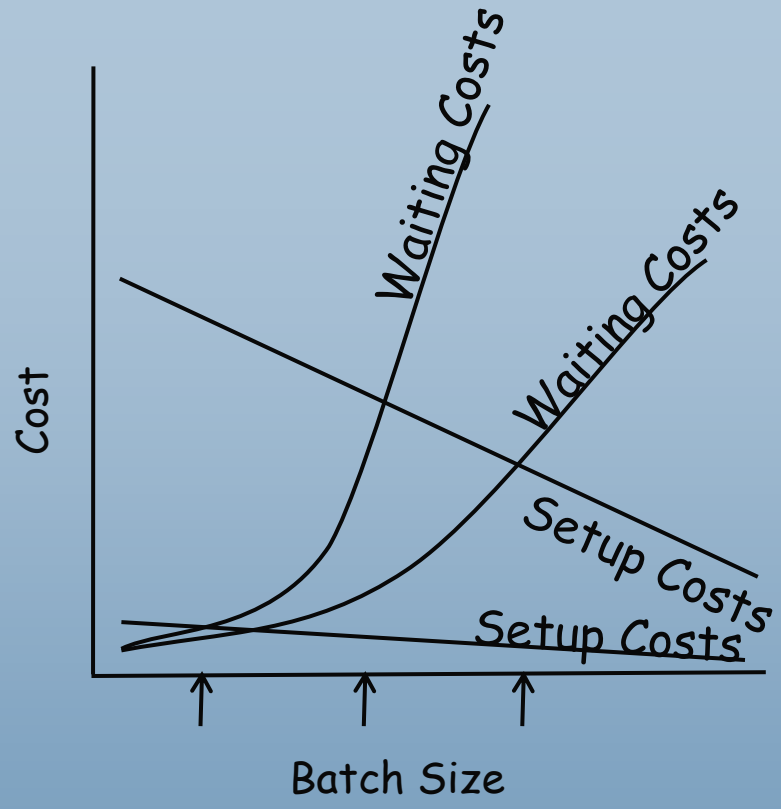
What Doesn't

1. Complexity
2. Handoffs

Batch Size

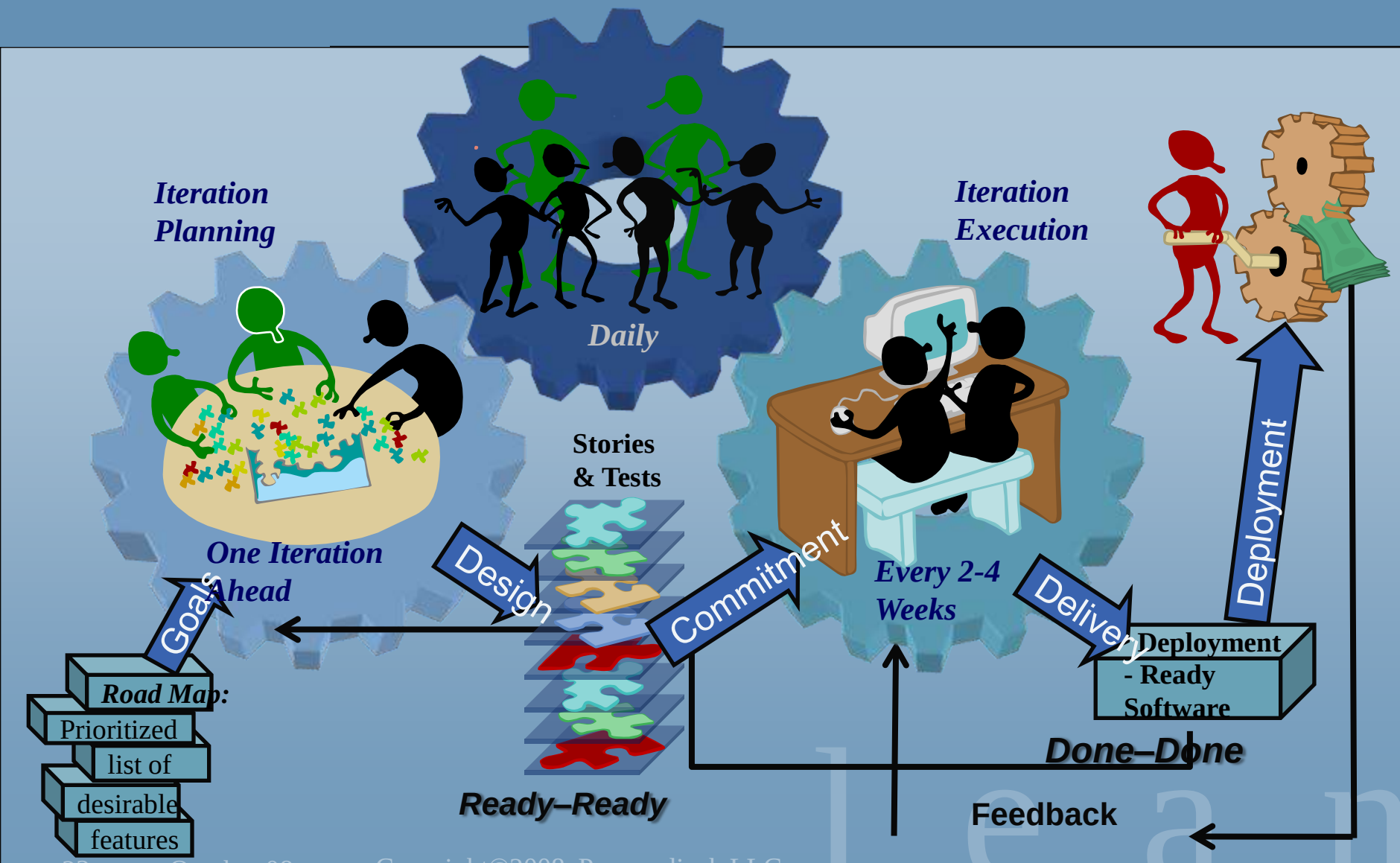
What is Batch Size?

- ✓ The amount of information transferred at one time
  - × The % of specifications completed before development begins
  - × The amount of code tested in a system test
- ✓ Compare:
  - × Cost of setup (linear)
    - ❖ Test set-up and execution
  - × Cost of waiting (hyperbolic)
    - ❖ Find/fix defects long after injection
- ✓ Waiting costs are:
  - × Usually hidden & delayed
  - × Often larger than expected
- ✓ The Lean Approach:
  - × Recognize true waiting costs
  - × Drive down setup overhead



From Don Reinertsen
www.roundtable.com/MRTIndex/LeanPD ART-reinertsen-INT2-1.html

Iterative Development



Reduce Set-up Time

Manufacturing

Common Knowledge:

- ✓ Die changed have a huge overhead
- ✓ Don't change dies very often

Taiichi Ohno:

- ✓ Economics requires frequent die change
- ✓ *One Digit Exchange of Die*

Software Development

Common Knowledge:

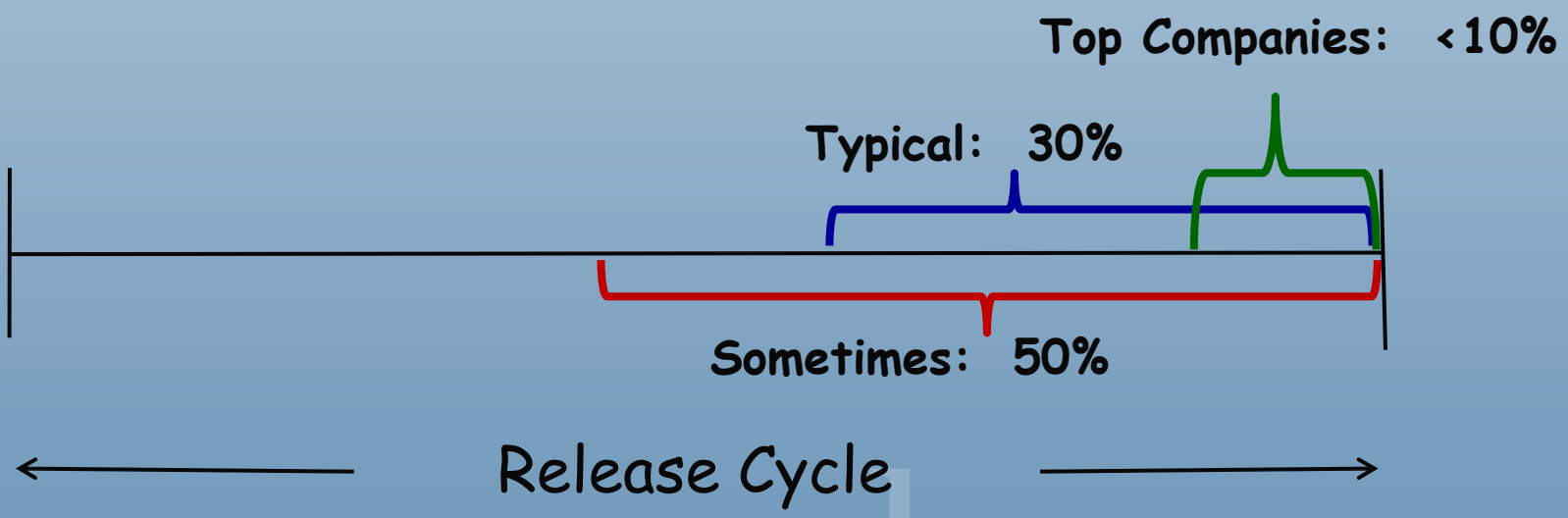
- ✓ Releases have a huge overhead
- ✓ Don't release very often

Lean:

- ✓ Economics requires many frequent releases
- ✓ *One Digit Releases*

How Good are You?

When in your release cycle do you try to freeze code and test the system? What percent of the release cycle remains for this "hardening"?



What Works – What Doesn't

What Works

1. Technical Practices
2. Small Batches
3. Pull Scheduling
4. Focus on Learning

What Doesn't

1. Complexity
2. Handoffs

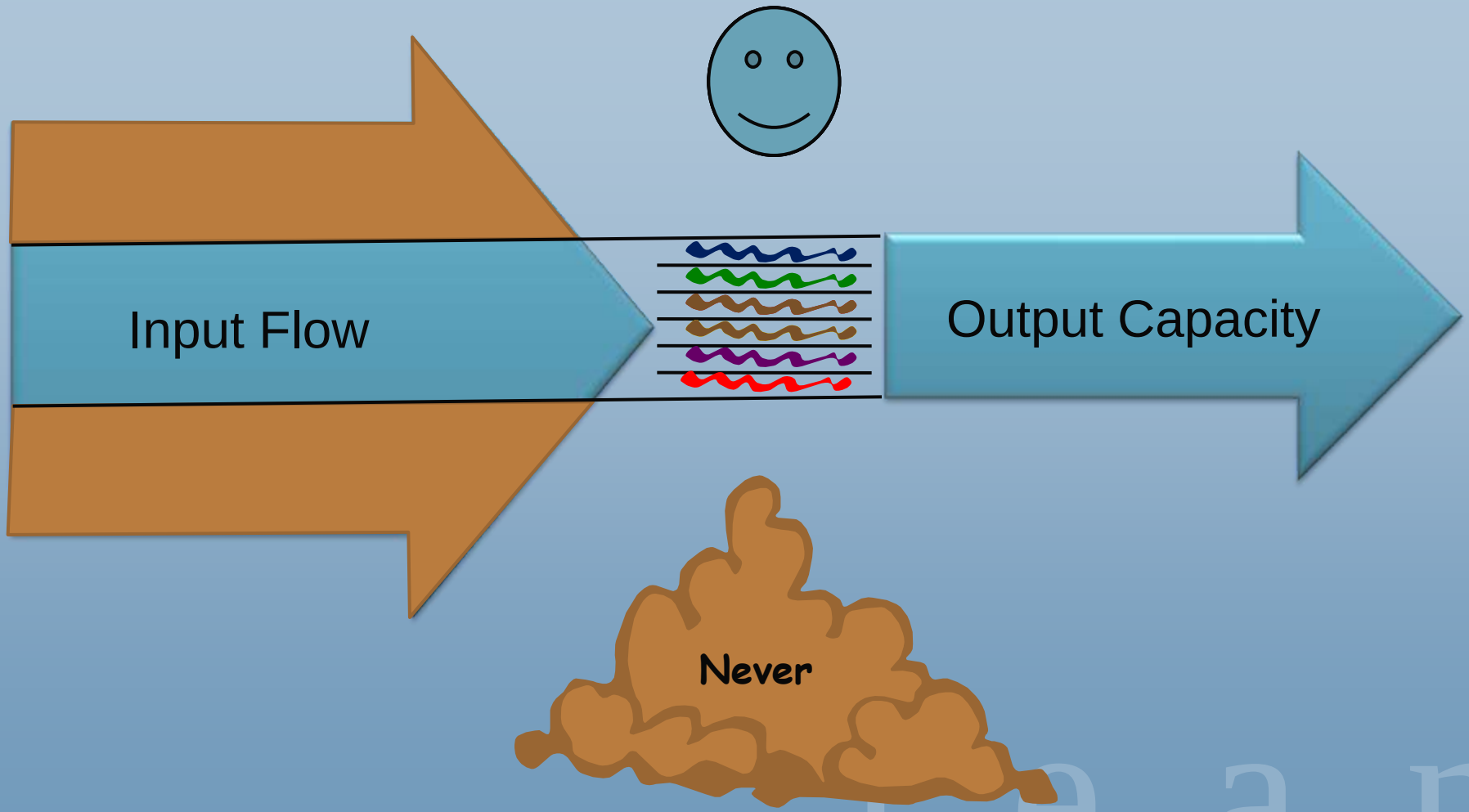
lean

Timebox, Don't Scopebox

Ask NOT: ***How long will this take?***

Ask instead: ***What can be done by this date?***

Pull Scheduling Small Requests



Pull Scheduling: Larger Systems

8/1
Start

9/15
Release 1:
Feature Set 1

11/1
Release 2
Feature Set 2

12/15
Release 3
Feature Set 3

2/1
Release 4
Feature Set 4

3/15
Release 5
Feature Set 5

5/1
Release 6
Feature Set 6

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Aug **Sep** **Oct** **Nov** **Dec** **Jan** **Feb** **Mar** **Apr** **May**

Start
Product
Concept
Approved

3 Months
Knowledge Review:
Customer Interest
Technical Approach

6 Months
Knowledge Review:
Proof of Concept
Alpha Release Decisions

9 Month Timebox
Knowledge Review:
Alpha Release
Beta Release Decisions

12 Month Timebox
Knowledge Review:
Beta Release
1st Release Decisions

15 Months
First
Production
Release

l e a n

What Works – What Doesn't

What Works

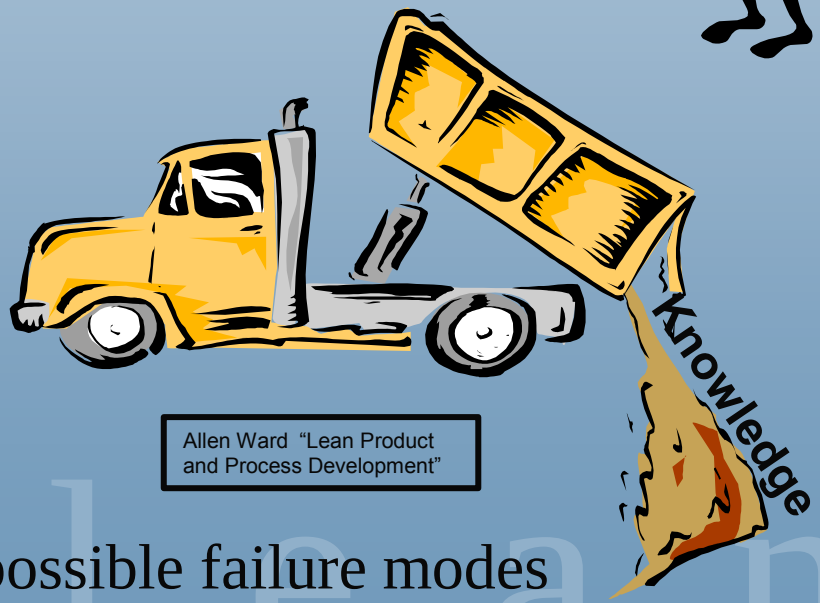
1. Technical Practices
2. Small Batches
3. Pull Scheduling
4. Focus on Learning

What Doesn't

1. Complexity
2. Handoffs

Wishful Thinking

1. Doing the same thing over and over again and expecting different results
   - ✓ Einstein's definition of Insanity
2. Making Decisions without Data
3. Discarding Knowledge
   - ✓ Failure to put knowledge & experience into usable form
   - ✓ Failure to involve the people who have relevant knowledge
4. Testing to Specification
   - ✓ Assuming the spec defines all possible failure modes



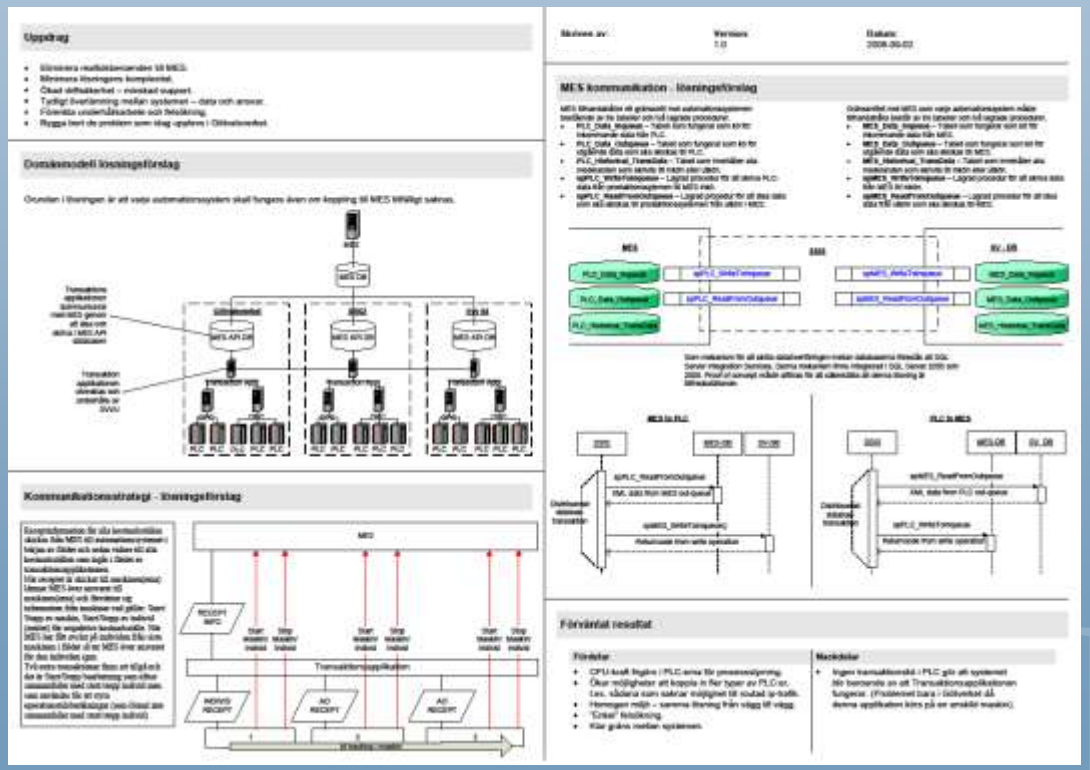
Allen Ward "Lean Product and Process Development"

Knowledge Briefs

The A3 Report

A concise, useful summary of knowledge that:

– condenses findings and
– captures important results.



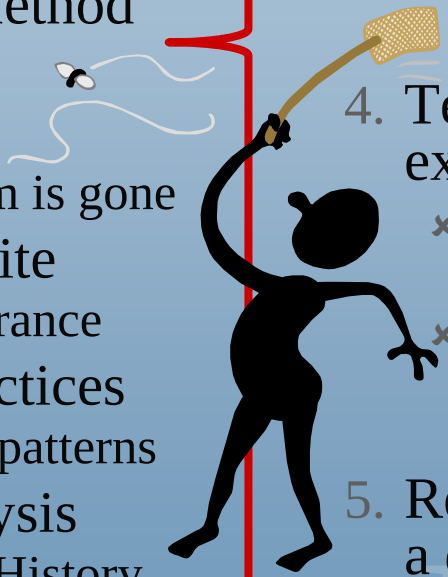
Software Examples

Patterns
Use Cases
Problem Under Investigation
Proposal to Change a Process
Business Goals of the System
Customer Interest Summary
Product Concept
Project Charter
Release Goal
Iteration Goal
High Level Architecture
Design Review Results

Knowledge-Based Debugging

Escaped Defect Analysis

1. Log all Problems
   - Capture relevent information
2. Reproduce the failure
   - Or analyze static data
3. Use the Scientific Method
   - Establish a diagnosis
4. Correct the Defect
   - Prove that the problem is gone
5. Improve the Test Suite
   - Prevent future reoccurance
6. Improve Design Practices
   - Find patterns and antipatterns
7. Use Statistical Analysis
   - Predict Defects from History



Use the Scientific Method

1. Observe the failure
2. Come up with a hypothesis as to the cause of the failure
3. Make predictions based on the hypothesis
4. Test the hypothesis with experiments & observations
   - If the experiment saitsfies the predictions, refine hypothesis
   - If the experiment does not satisfy the predictions, create an alteranate hypothesis
5. Repeat Steps 3 and 4 until a diagnosis is established.

See "*Predicting Bugs from History*" by Andreas Zeller
Chapter 4 in "Software Evolution" Mens & Demeyer, Ed.

See "*Why Programs Fail*" by Andreas Zeller

Relentless Improvement

KAIZEN

Kai

改

Change

Solve One Problem at a Time

Data-Based Problem Analysis

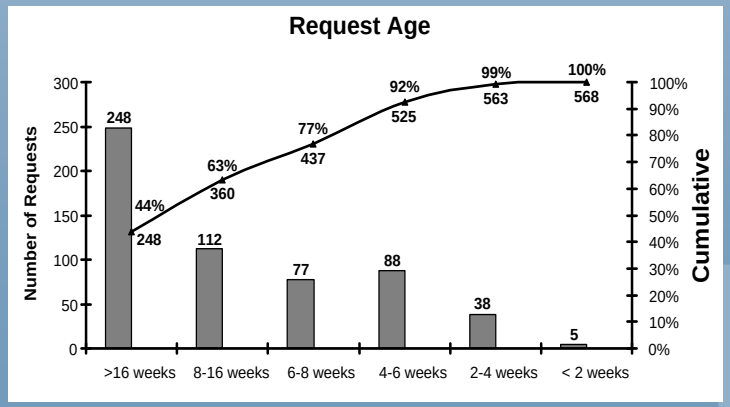
- ✓ What's Important?
  - × Pareto Analysis
- ✓ Root Cause Analysis
  - × Ishikawa (Fishbone) Diagram
  - × Five Why's?

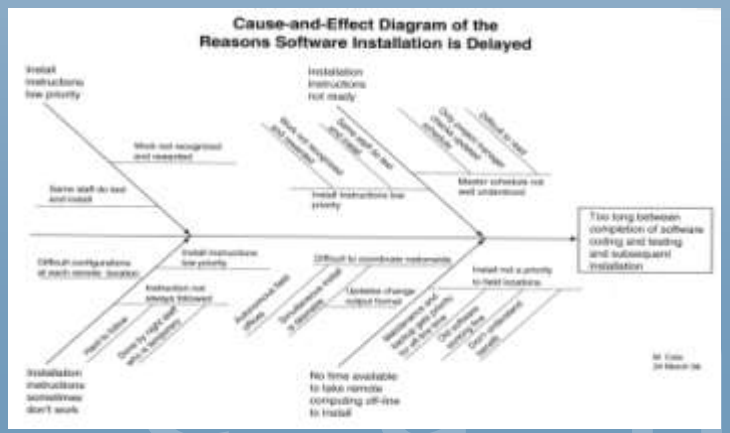
Many Quick Experiments

Zen

善

Good





Problem Solving Template

From Duward Sobek: http://www.coe.montana.edu/IE/faculty/sobek/A3/index.htm

To: _____
By: _____
Date: _____

PROBLEM

- Note any contextual or background information necessary to fully understand the issue.
- Indicate how this problem affects the company's goals or is related to its values.

CURRENT CONDITION:

- Insert a diagram that illustrates how the current process works.
- Label the diagram so that anyone knowledgeable about the process can understand.
- Note the major problems (we like to put them in storm bursts to set them apart)
- Include quantified measures of the extent of the problem – graphical representations are best!

ROOT CAUSE ANALYSIS:

- List the main problem(s)
- Ask appropriate "why?" questions until you reach the root cause. A rule-of-thumb: you haven't reached the root cause until you've asked "why?" at least 5 times!
- List the answers to each why question

Problem
↳ first immediate cause
↳ cause for the first immediate cause
↳ deeper cause to the preceding cause
↳ etc.

TARGET CONDITION:

- Insert a diagram that illustrates how the proposed process will work, with labels.
- Note or list the countermeasure(s) that will address the root cause(s) identified.
- Predict the expected improvement in the measure of interest (specifically and quantitatively)

PLANNED EXPERIMENTS

- List the actions which must be done in order to realize the Target Condition, along with the individual responsible for the action and a due date.
- Add other items, such as cost, that are relevant to the implementation.

MEASUREMENT & FOLLOWUP

What Works – What Doesn't

What Works

1. Technical Practices
2. Small Batches
3. Pull Scheduling
4. Focus on Learning

What Doesn't

1. Complexity
2. Handoffs

Respect Complexity

Step 1: Accept that Change is ***not*** the Enemy.

- ✓ 60-80% of all software is developed after first release.

Step 2: Recognize that Complexity ***is*** the Enemy.

- ✓ The classic problems of developing software products derive from this essential complexity and its nonlinear increase with size. *Fred Brooks*

Step 3: Don't be Fooled by Rising Levels of Abstraction.

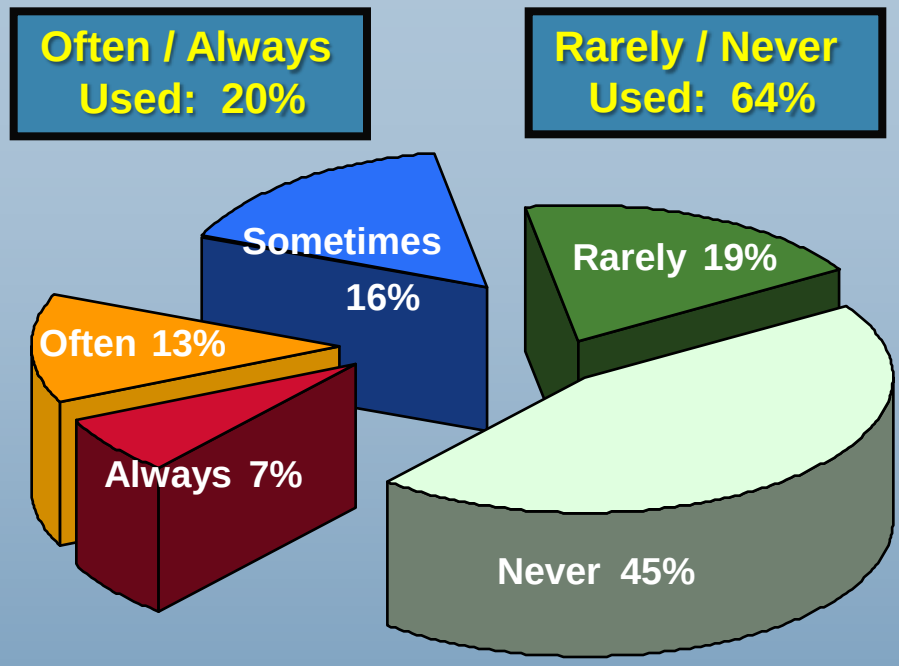
- ✓ High level languages removed drudgery from programming, making the job *more* complex and requiring *higher caliber* people. *Dijkstra*

Step 4: Keep it Simple – Write Less Code!

- ✓ A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away. *Antoine de Saint-Exupery*

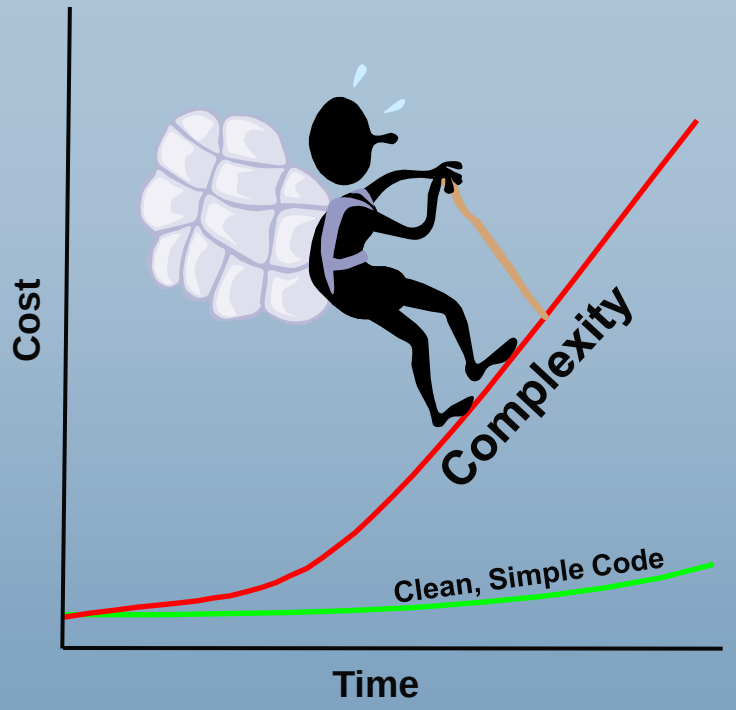
Extra Features

Features / Functions Used in a Typical System



Standish Group Study Reported at XP2002 by Jim Johnson, Chairman

Cost of Complexity



The Biggest opportunity for increasing Software Development Productivity: Write Less Code!

What Works – What Doesn't

What Works

1. Technical Practices
2. Small Batches
3. Pull Scheduling
4. Focus on Learning

What Doesn't

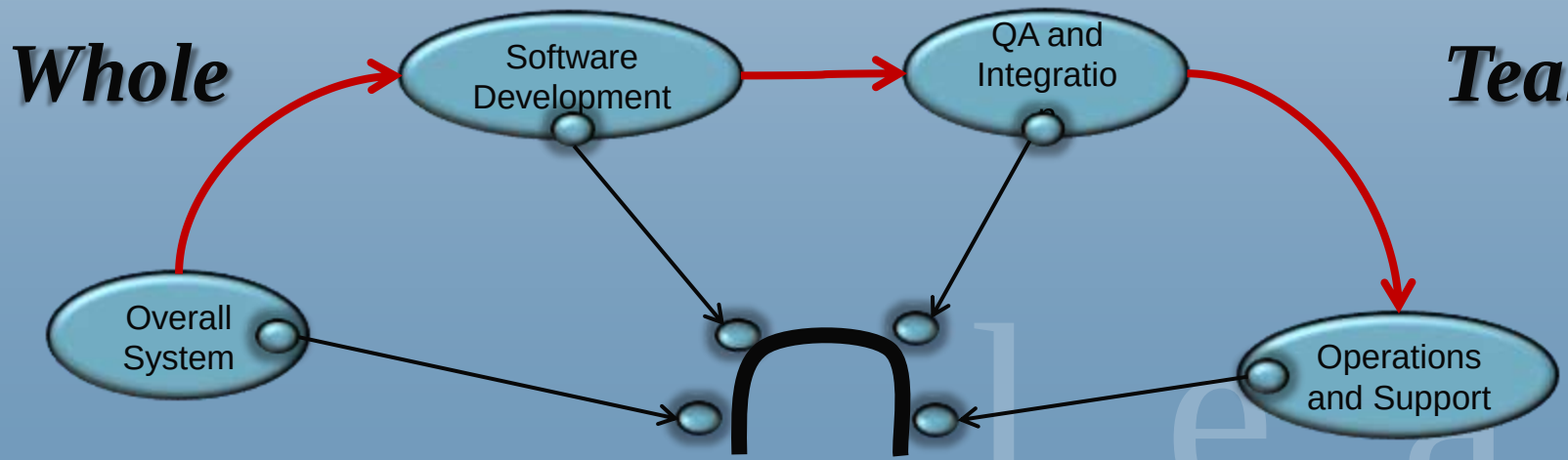
1. Complexity
2. Handoffs

Handoffs

*A hand-off occurs whenever we separate:**

- ✓ Responsibility – What to do
- ✓ Knowledge – How to do it
- ✓ Action – Actually doing it
- ✓ Feedback – Learning from results

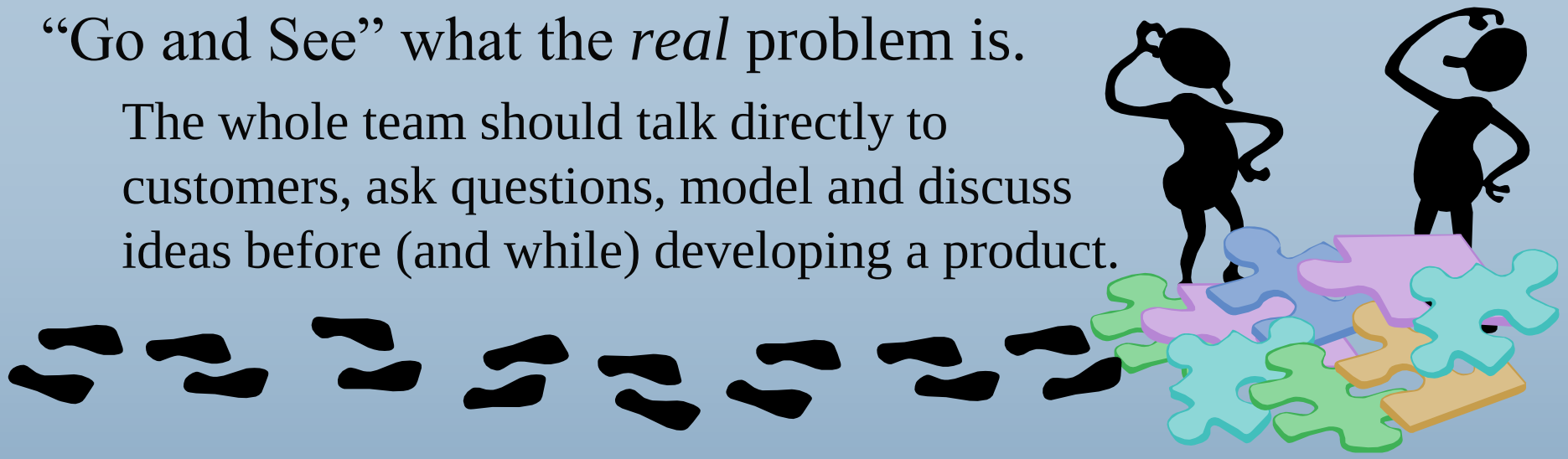
*Allen Ward "Lean Product and Process Development"



Go and See

"Go and See" what the *real* problem is.

The whole team should talk directly to customers, ask questions, model and discuss ideas before (and while) developing a product.



"When we started doing a lot of banking, we hired some product managers with banking experience. One day one of them comes to a meeting that included me and banking engineers and says, "I want these features." And I replied, "If you ever tell an engineer what features you want, I'm going to throw you out on the street. You're going to tell the engineers what problem the consumer has. And then the engineers are going to provide you with a better solution than you'll ever get by telling them to put some dopey feature in there." Bill Campbell – Chairman, Intuit. Counselor – Apple, Google, etc.

Reality of Agile: Microsoft Perspective

- Context
  - patterns & practices – Mecca of agile
    - XP and Scrum
  - Big product units – a somewhat different story

Customer-Connected Engineering

- Communication
  - Breadth: Codeplex communities
  - Depth: Customer Advisory Boards
- Think in terms of stories not features
  - Software from the customer perspective
- Frequent checkpoints with customers
  - Using frequent drops to the communities
  - Customer workshops
  - Advisory meetings

Source Control Clients

Source control client connection instructions

Project Name: entlib
TFS Server URL: https://tfs01.codeplex.com
Subversion URL: https://entlib.svn.codeplex.co

Recent Check-Ins

1-7 of **7** Change Sets < Previous **1** Next > Show **All** Cha

Change Set	Download / Browse	Date	Comment	By	Downl
42311	Download Browse	Fri at 3:11 PM	Automatic Check-in from p&p build server.	ctavares	1
41652	Download Browse	Oct 13 at 10:33 AM	Automatic Check-in from p&p build server.	ctavares	239
40861	Download Browse	Sep 29 at 11:11 AM	Automatic Check-in from p&p build server.	ctavares	193
39896	Download Browse	Sep 15 at 12:36 PM	Automatic Check-in from p&p build server.	ctavares	365
38960	Download Browse	Sep 4 at 3:21 PM	Automatic Check-in from p&p build server.	ctavares	216
36894	Download Browse	Aug 19 at 3:42 PM	Automatic Check-in from p&p build server.	ctavares	255
36891	Download Browse	Aug 19 at 3:28 PM	Automatic Check-in from p&p build server.	ctavares	32

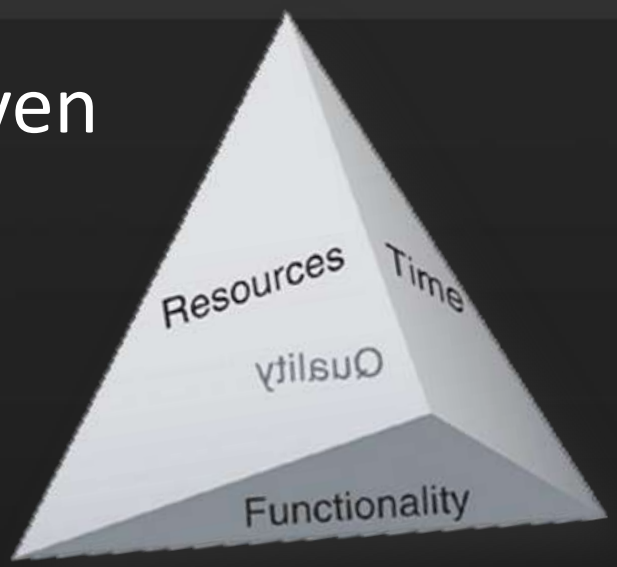
Be Clear on What Success Means

- On time?
- On budget?
- In scope?

- That's not project success – that's success of project estimation at best
- Real success is how much **value** the project brings to business
  - Remember Motorola Iridium???

Planning and Estimation

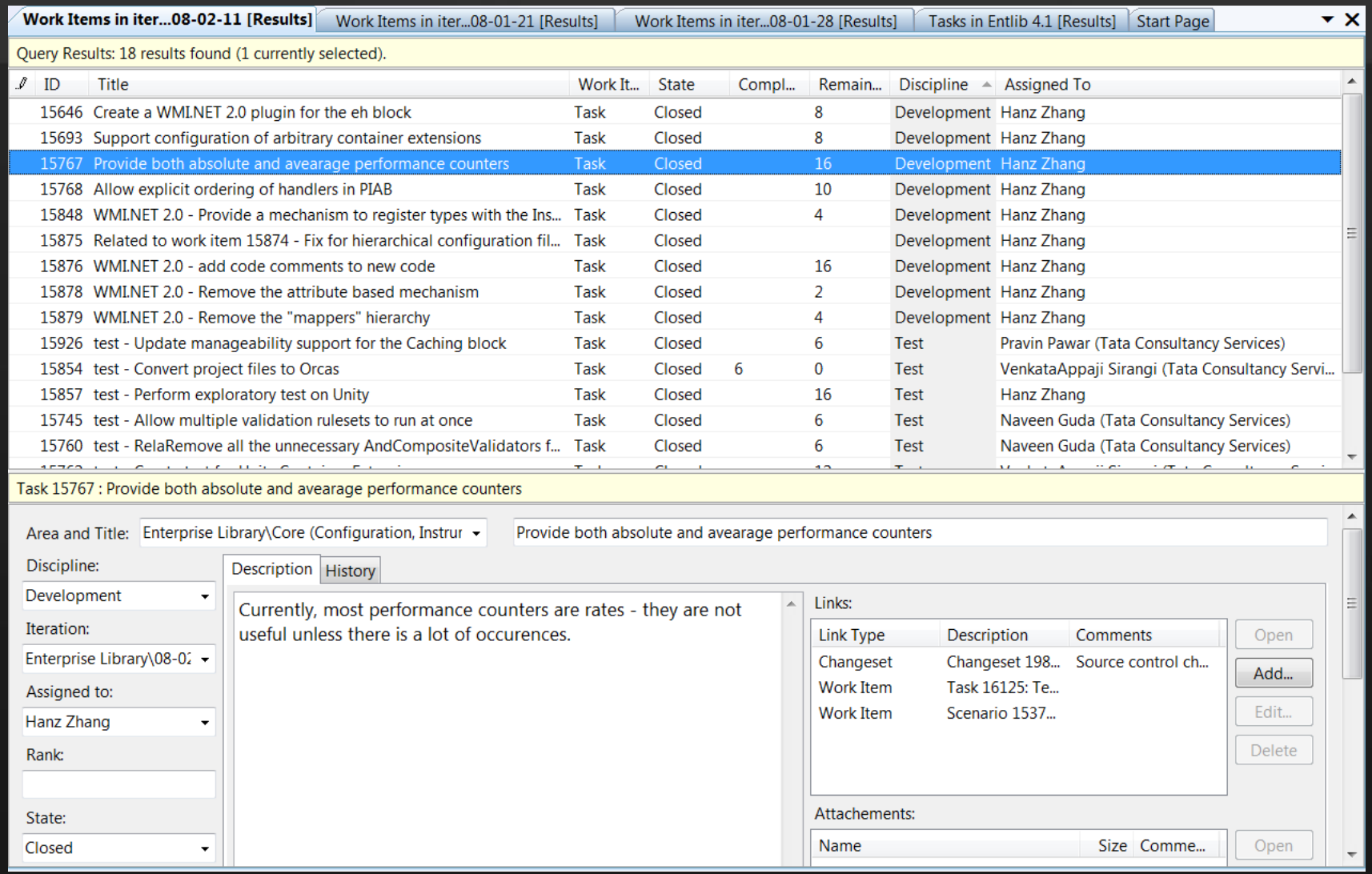
- Planning-driven vs. plan-driven
- Maintain prioritized story backlog
- Choose initial t-shirt size
- The planning game
- Monitor velocity
- Plan for iteration zero



"Plans are nothing; planning is everything."
- Dwight D. Eisenhower.

Low-Fi Iteration Planning: Warm

Hi-Fi Iteration Planning: Still Warm



Iteration 3/4 - Low hanging fruit

CFD


DELL

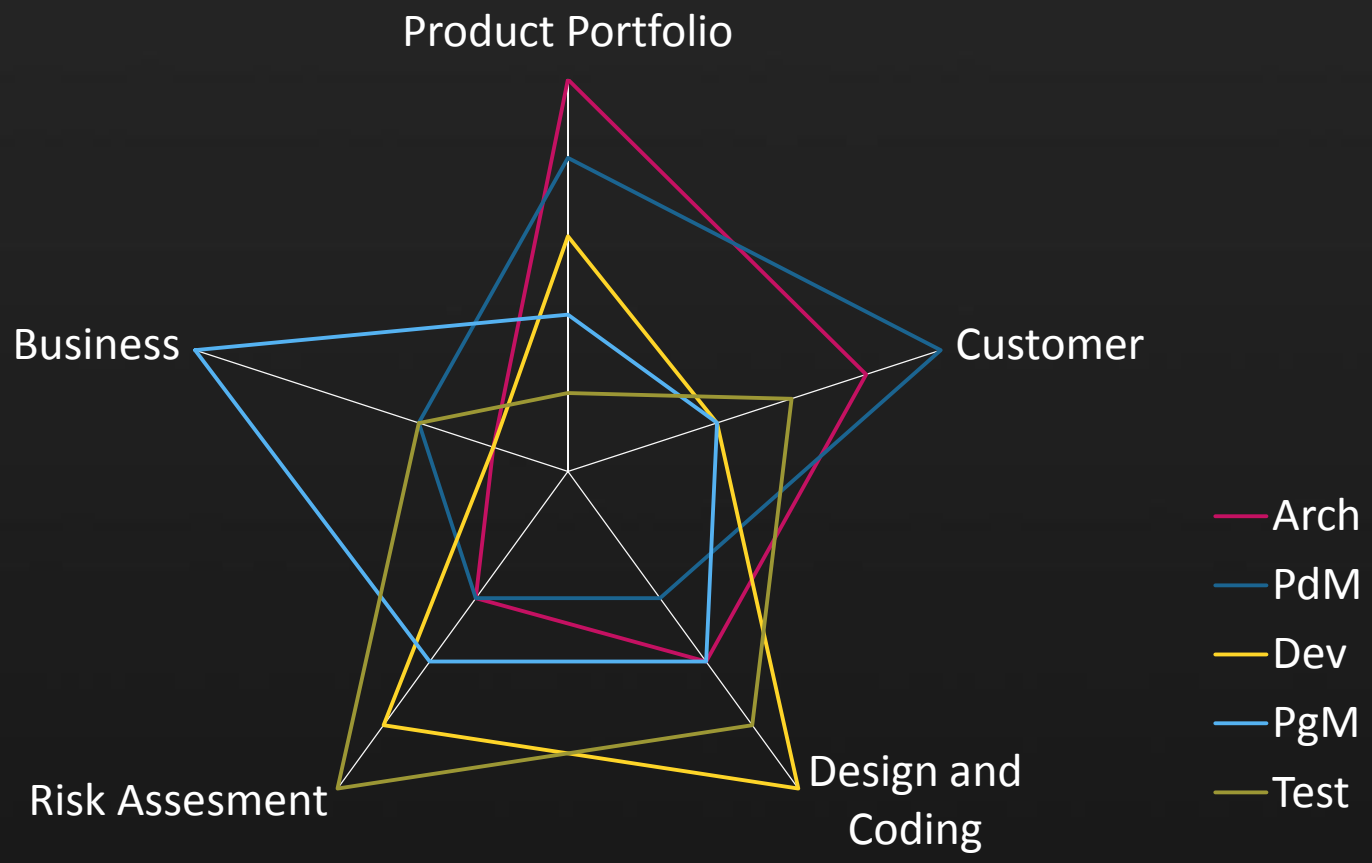
Team Formation

- Program manager
- Dev lead + developers
- Test lead + testers
- Technical Writer(s)
- Domain experts (SMEs)

- Core teams with consistent members
  - Consultants available
- But it's about what you do, *not* job titles!
- Small teams but not too small

Team Tasks… The Game

What do you do on the team?



pig > cow > chicken > larus glaucescens

Are we done?

vs

The Done-Done State

- Team **agrees** to non-ambiguously describe what must take place for a feature/release to be considered complete.
- Defining and adhering to a done-done state affect **time-to-market** and **visibility**.
- The closer you come to **deployable system**, the more **confidence** you have in your progress and the less time to release.
- Cost is reduced because you pay for defect fixes early
  - Think of prevention vs inspection

Done-done (Feature/Story Level) - Example

- The acceptance criteria are specified and agreed upon
- The team has a test/set of tests (preferably automated) that prove the acceptance criteria are met
- The code to make the acceptance tests pass is written
- The unit tests and code are checked in
- The CI server can successfully build the code base
- The acceptance tests pass on the bits the CI server creates
- No other acceptance tests or unit tests are broken
- User documentation is updated
- The customer proxy signs off on the story

Done-done (Release Level) - Example

- All MCR features are included in the RC build.
- All included features have been accepted by the customer.
- A security review has been conducted.
- The test team is confident that none of the included features has a significant risk of causing problems in the production environment (MQR is met)
- There are clear, concise deployment and rollback instructions for the operations team.
- There are clear trouble-shooting scripts and knowledge base articles for use by the help desk representatives.

Done-done – Guidelines

- Reporting on **partial work done is error prone**; at worst, we are 90% done 90% of the time
- The closer a requirement is delivered to deployable, the **less uncertainty** your team has about the true state of the system.
- Remember, only functionality that is **delivered to the customer** has **real value**.
- The closer a requirement is delivered to a deployable state, the more defects you have found and eliminated.
- The done-done state should push your team members without breaking them.

Demo to Stakeholders

- **Confidence** is gained by **regularly demonstrable progress**
- Increases **visibility**
- Increases **product utility** by giving the customer a **concrete system** to evaluate

Demo to Stakeholders – Guidelines

- Working in vertical slices
- Early iterations will not have much built. Do demos (even if small) anyway to get into the habit of regularly reviewing your work
  - -> healthy rhythm providing feedback
- Don’t demo features that are partially done.
- Keep the demo short (<30 mins)
- The feedback you receive might be conflicting, the demo is not the place to resolve issues.
- !!!! Take notes!

Testing to the Forefront of Software Development

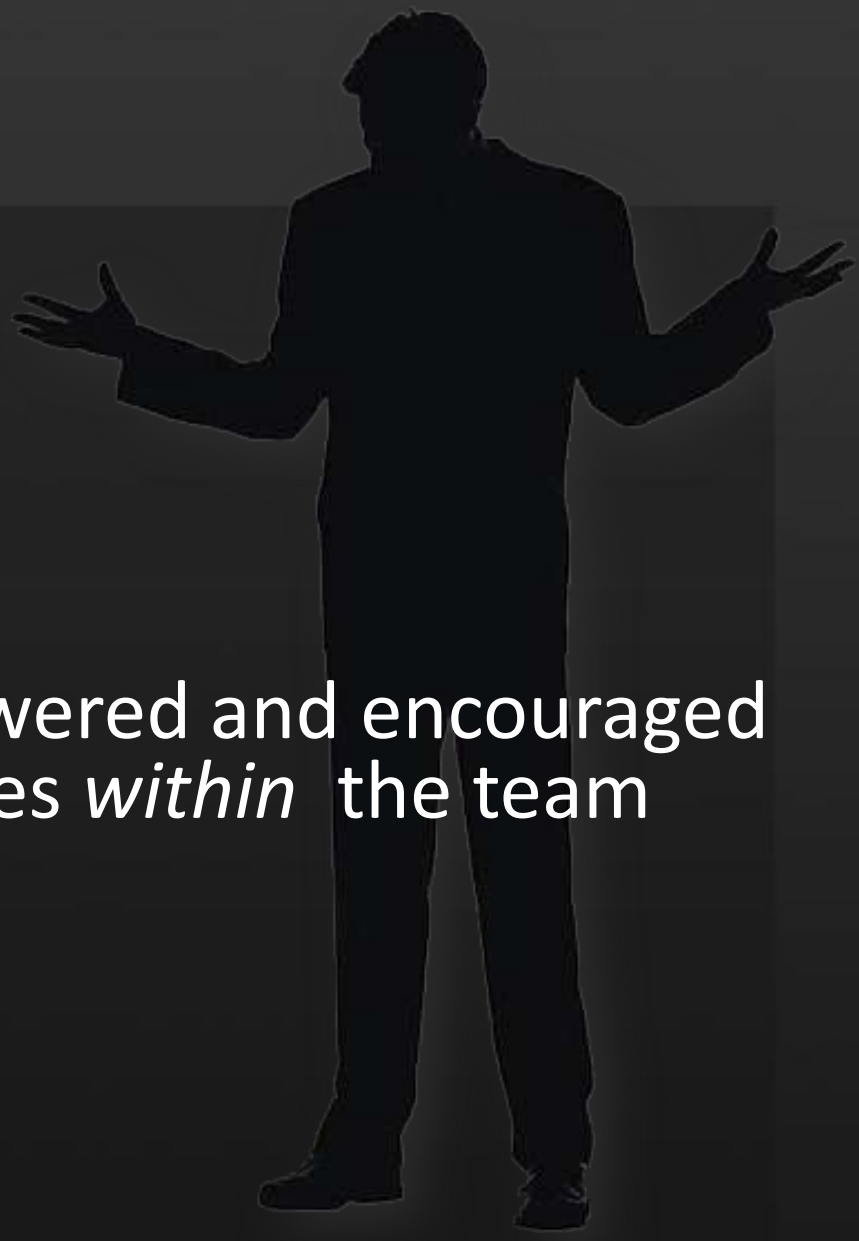
- Test artefacts are **assets** not liability
- If you are a customer, demand the supplier ships test cases (all – unit, integration, system, perf – all!)
- Focus on testing early
- Focus on producing fast and non-fragile tests – subconteneous

Process-Agnostic Practices

- Unit testing (tests are assets not liability)
- Test-Driven Development (TDD)
- Continuous Integration (CI)
- Acceptance testing (automate what makes sense)
- Iteration planning
- Daily stand-ups
- Retrospectives
- Sustainable pace

- You don’t have to be canonically agile to get benefits…
- Emphasis on learning!

Challenges



- Too many cooks
- Rewarding teams
- Team continuity

- Teams should feel empowered and encouraged to address their challenges *within* the team

PDC2008
PROFESSIONAL DEVELOPERS CONFERENCE

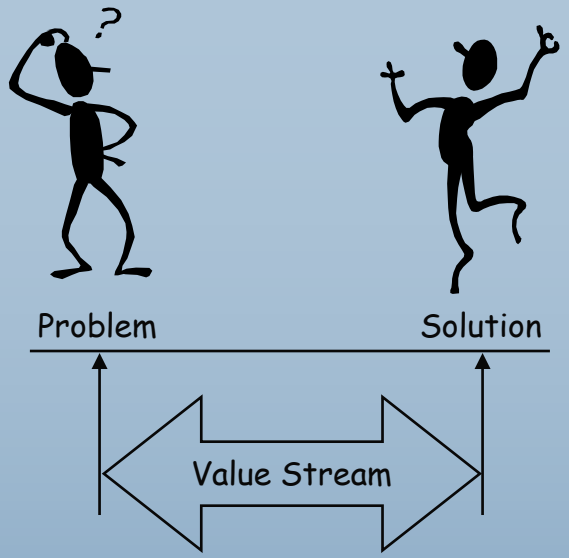
staged discussion

4 Hot Issues

PDC2008
PROFESSIONAL DEVELOPERS CONFERENCE

Value Stream Maps

Map the Value Stream



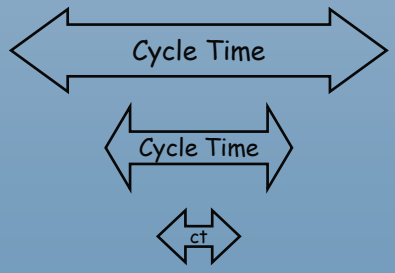
Value Stream

- ✓ The flow of activities that starts with a customer in need, and ends when that customer's need is satisfied.

Process Capability:

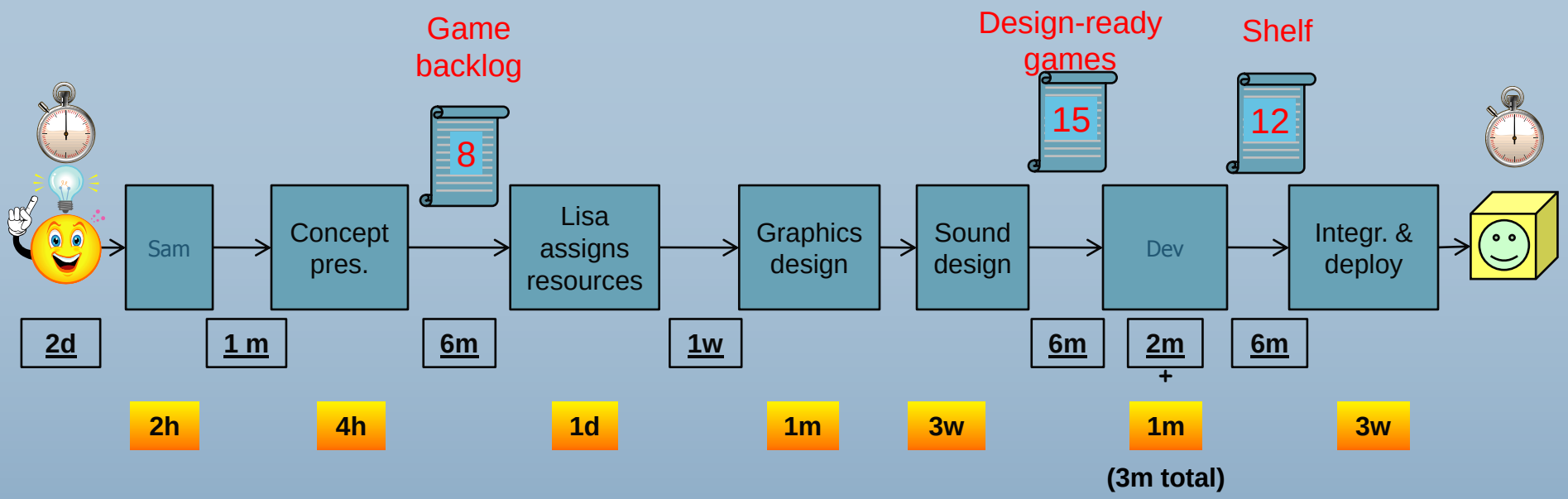
- ✓ The reliable, repeatable cycle time from customer need until that need is satisfied.

Multiple Value Streams



- Product Concept — Product Starts Delivering Value
- Feature Request — Feature in Production
- Urgent Need — Maintenance Patch Deployed

End-to-End Value Stream



Thanks to: Henrik Kniberg, of Crisp, Stockholm
Used with Permission

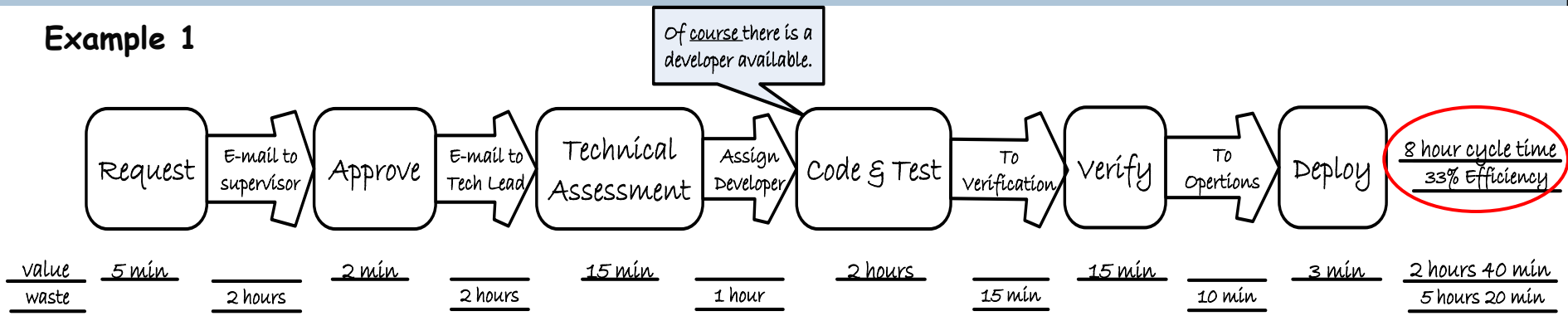
3m Value Added Time / **25 m cycle time** = 12% Process cycle efficiency

Games out of date
⇒ Missed market windows
⇒ Demotivated teams
⇒ Overhead costs

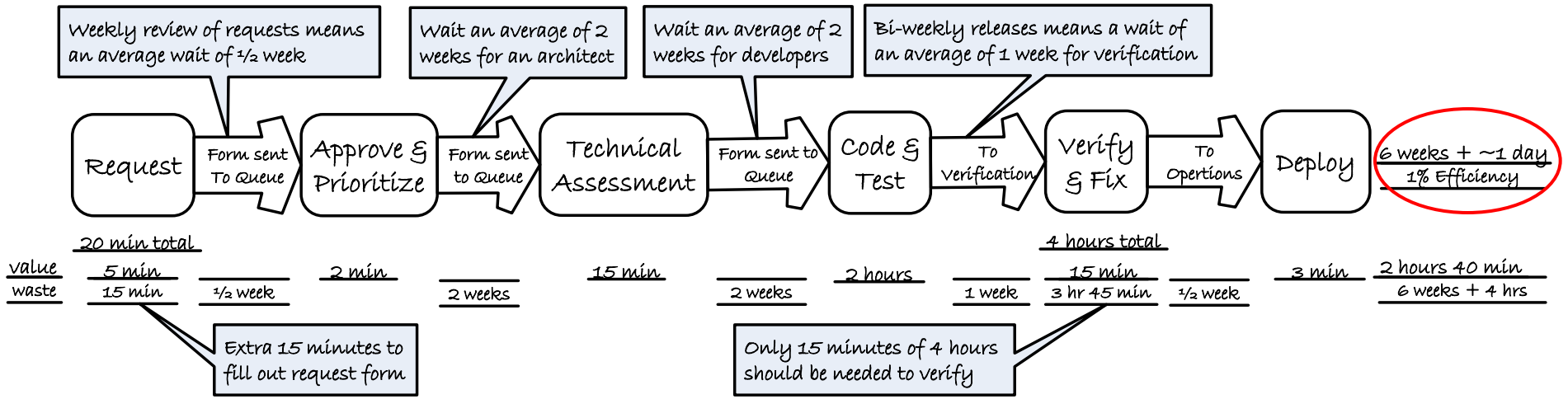
What would you do?

Value Stream Examples

Example 1

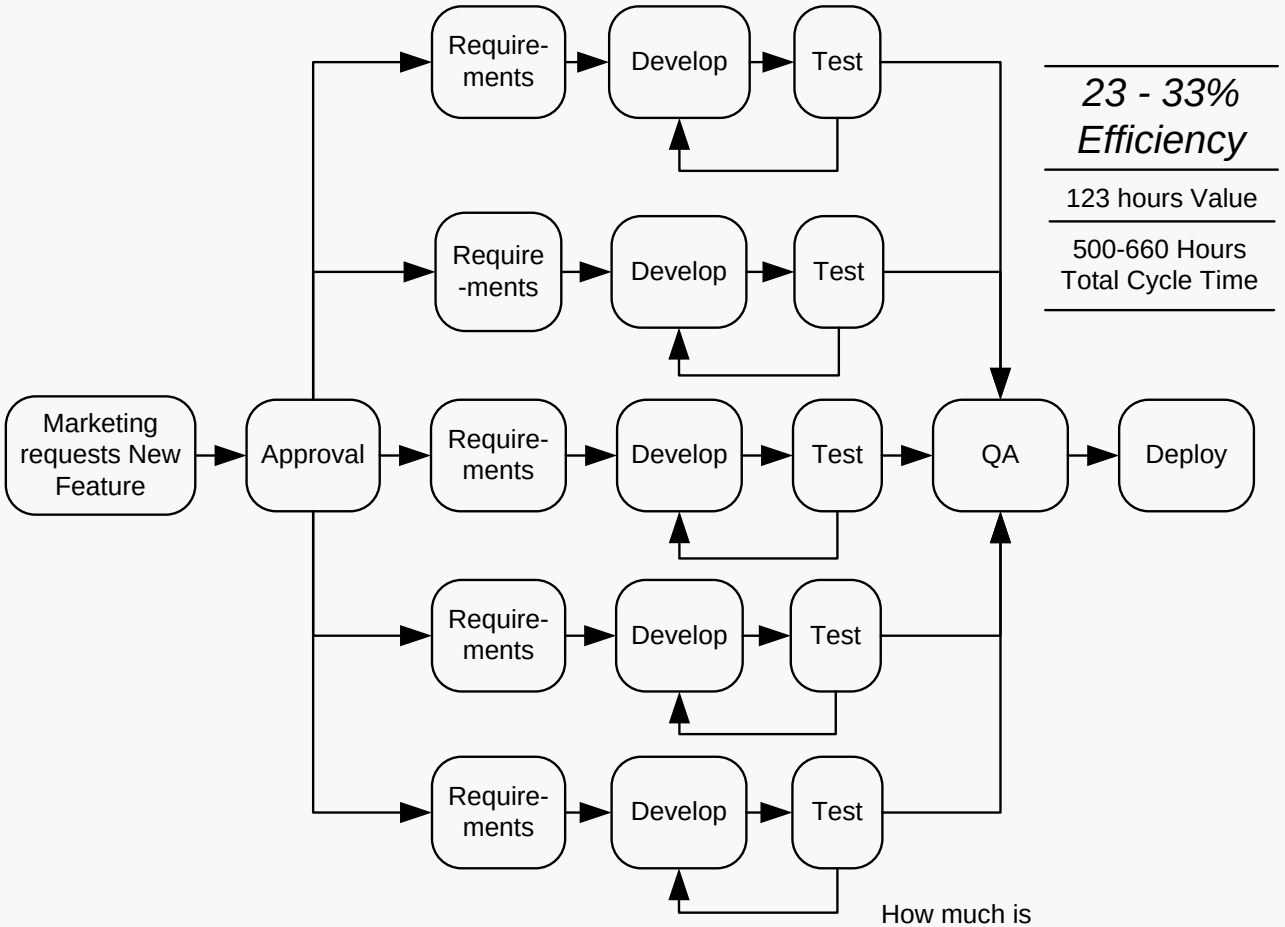


Example 2



Value Stream Examples

Example 3



value	1 hour	1 hour	2 weeks working together	How much is Value?		1 hour
waste		1 week	1 day	1 week	2-3 months to merge	½ week

PDC2008
PROFESSIONAL DEVELOPERS CONFERENCE

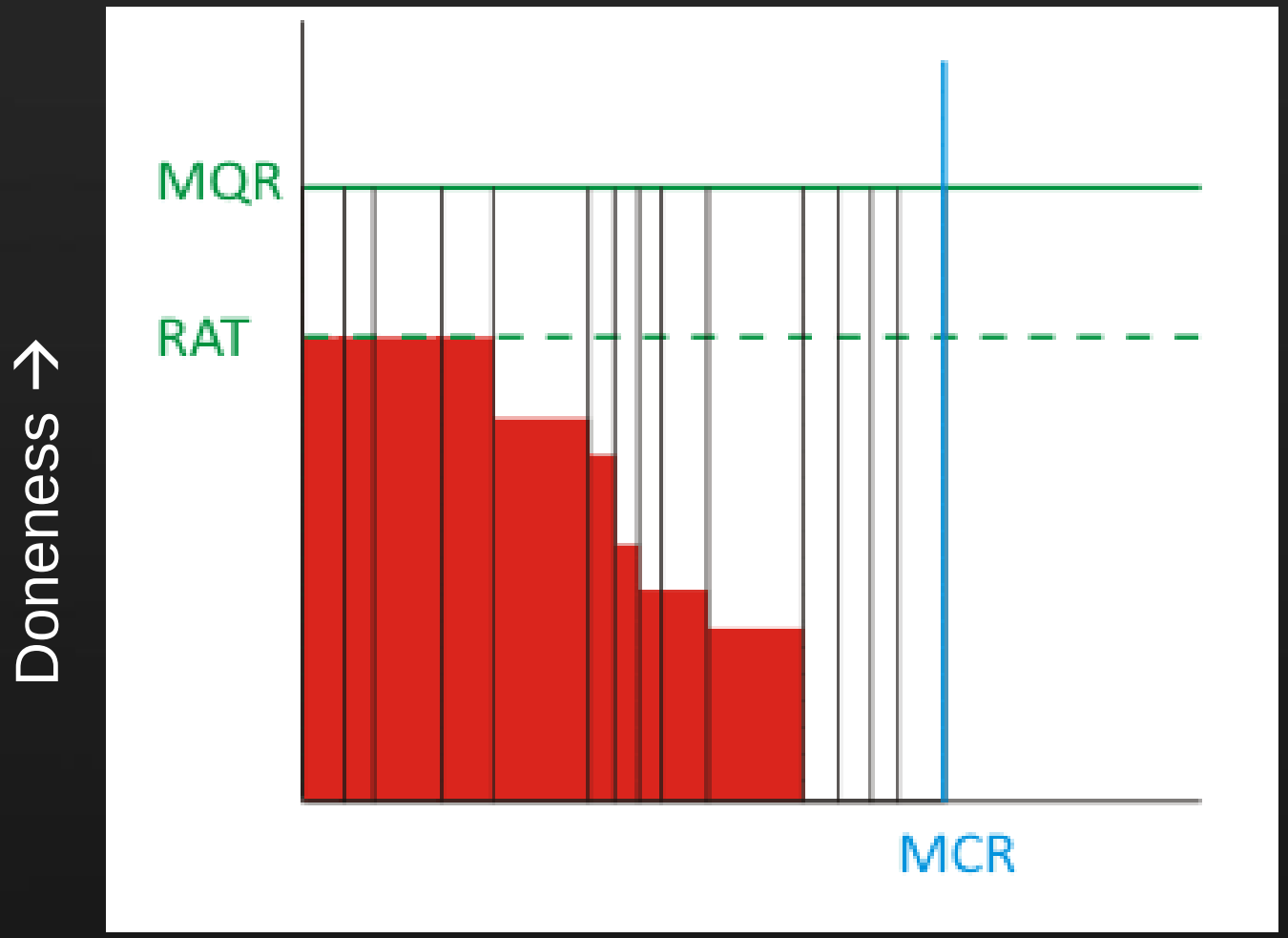
acceptance testing

How to know the software is ready for you and your customers

Acceptance Testing

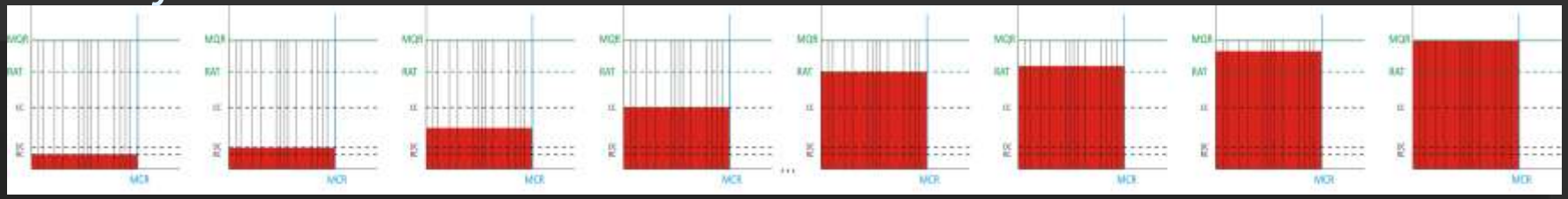
- Planned investigation by a **customer or customer proxy** to what degree the software system meets their **expectations**
- **Readiness** vs. **Acceptance**

Doneness Model

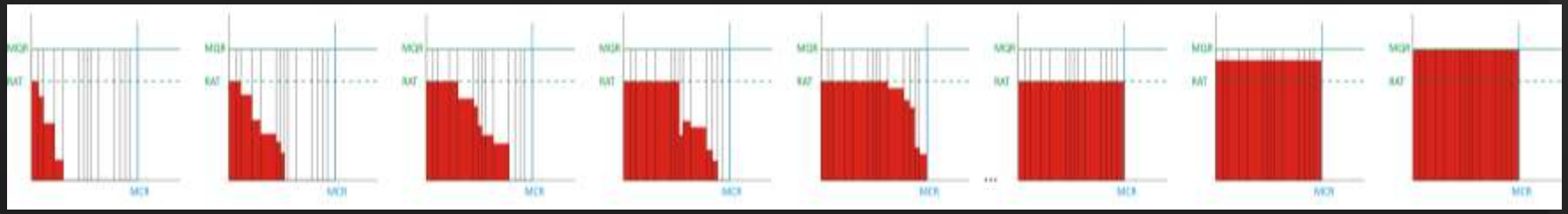


Doneness Model

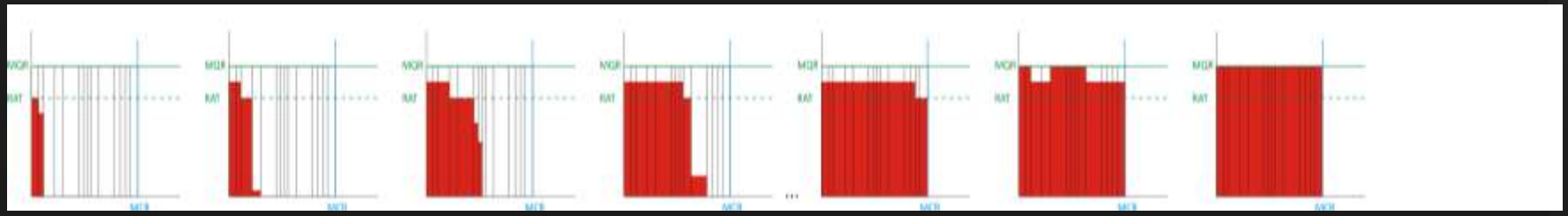
Waterfall



Incremental with cycles longer than in XP



Extreme Programming



Time

Two Perspectives on Tests

Business (Granularity of Requirement)

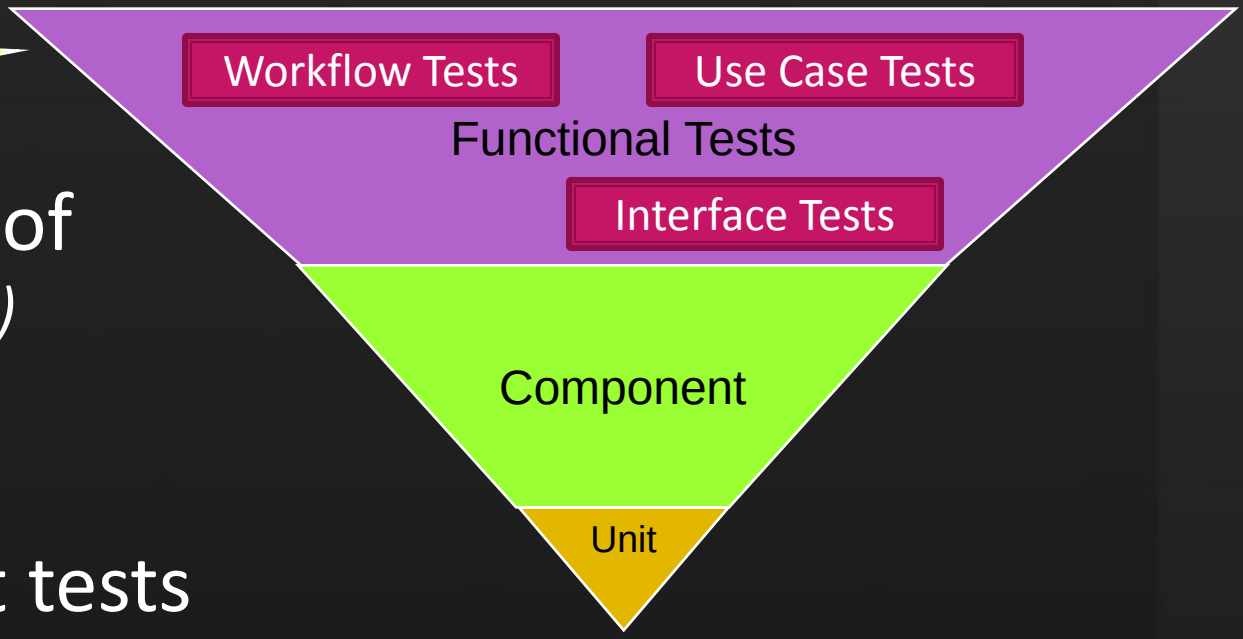
- Workflow
- Transaction / Use Case
  - Interface (e.g. UI)
- Rule (Calc, Check)

Technical (Decomposition of SUT)

- Integration
- System
- Component
- Unit

Classic Functional Test Strategy

Manual or Record & Playback

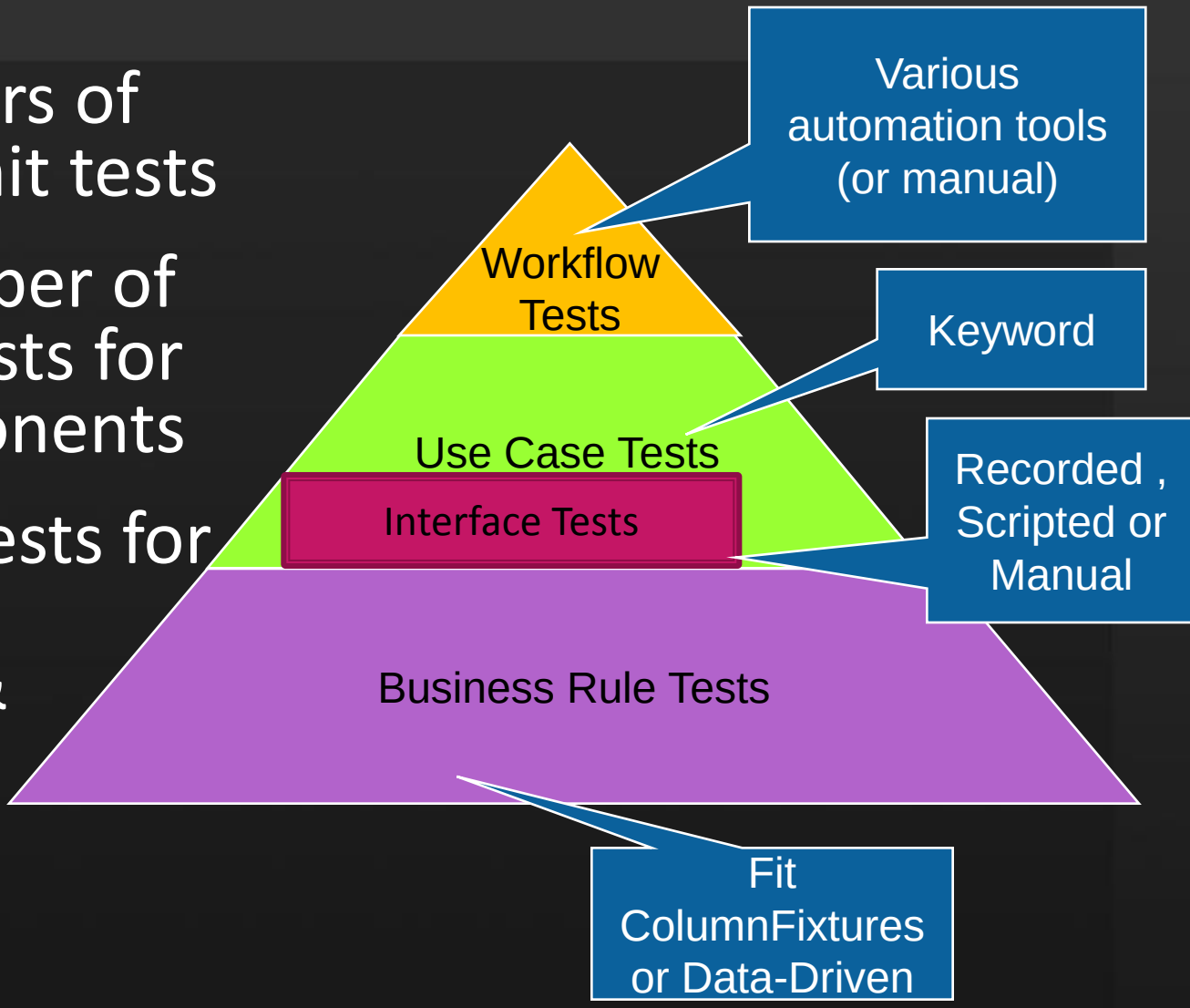


- Large numbers of *(possibly automated)* functional tests
- Very few if any automated unit tests

Typical when testing (& automation) is "QA's job"

Better Test Automation Strategy

- Large numbers of very small unit tests
- Smaller number of functional tests for major components
- Even fewer tests for the entire application & workflow



Test Automation Strategy

- Identify Goals
  - Why are we automating?
- Identify Risks
  - What risks will automation address?
- Identify types of tests & when to use
  - What kinds of tests need automation?
- Identify tools for each type of test

Test Automation Guidelines

- Record & Playback ≠ Test Automation ≠ Test Automation fetish
- Test requirements at lowest level possible
  1. Component
  2. Use Case
  3. Workflow
- Avoid testing multiple concerns together
  - UI and Logic/Rules
- Specify tests at highest possible level of abstraction & pick appropriate framework

PDC2008
PROFESSIONAL DEVELOPERS CONFERENCE

global teams

Distributed Teams

- This is the reality of software development today
- Maximize communication
  - Joint project kick off iteration
  - More formal story management
- There is only *one* team not local and remote
  - Everyone participates in daily stand-ups
- Frequent on site visits
- Time zones harder to manage than distance

Distributed Teams

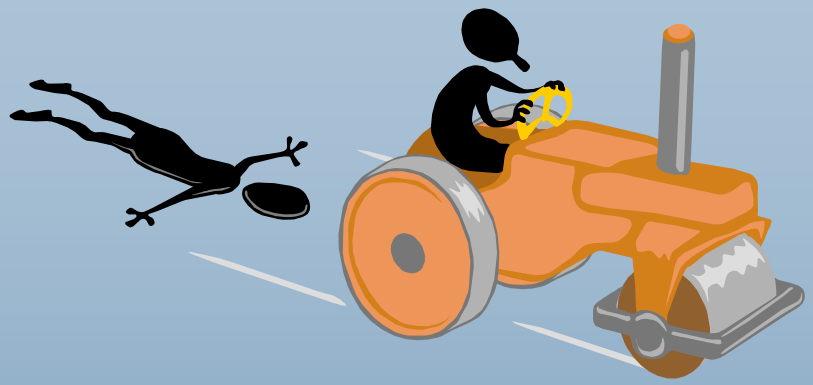
- Conway's law
- Avoid breaking user stories into tasks and then assigning tasks according to geography. Dysfunction: specialization, silos, low "bus" count
- Have all members participate in the standups via con call once in a while
- Team continuity: try to keep teams together over multiple projects
- Single system of reference/ knowledge base
  - Sharepoint or wiki or Shared OneNote or Groove

PDC2008
PROFESSIONAL DEVELOPERS CONFERENCE

agile contracts

The Apparent Problem with Two Party Interactions

Opportunism

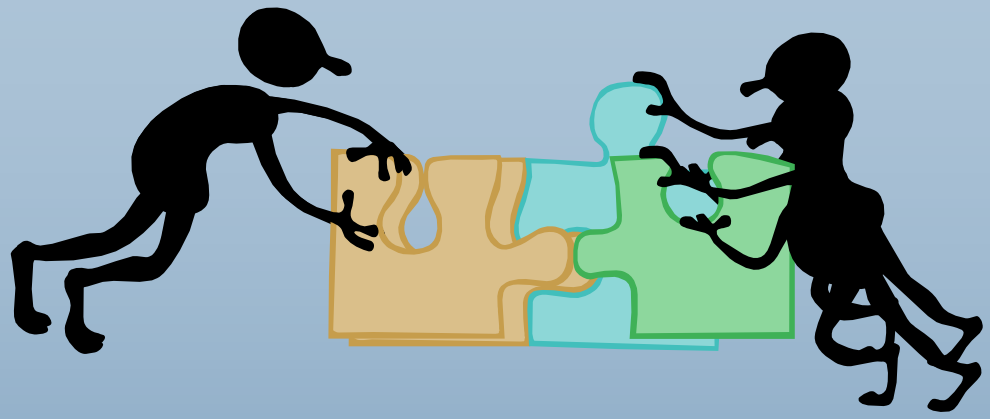


Conventional Wisdom:

- ✓Companies inevitably look out for their own interests
- ✓Contracts are needed to limit opportunistic behavior

The Real Problem with Two Party Interactions

Conflict of Interest



The Lean Approach

- ✓ Assume other party will act in good faith
- ✓ Let the relationship limit opportunism
- ✓ Use the contract to set up incentives
  - × Align the best interests of each party with the best interests of the joint venture
  - × ***Eliminate Conflicts of Interest!***

Fixed Price Contracts

Supplier is at greatest risk

- ✓ Customer has little incentive to accept the work as complete

Generally does not give the lowest cost

- ✓ Competent suppliers will include cost of risk in bid
- ✓ Creates the game of low bid with expensive change orders

Generally does not give the lowest risk

- ✓ Selection favors the most optimistic [desperate] supplier
  - × Least likely to understand project's complexity
  - × Most likely to need financial rescue
  - × Most likely to abandon the contract

Customers are least likely
to get what they really want

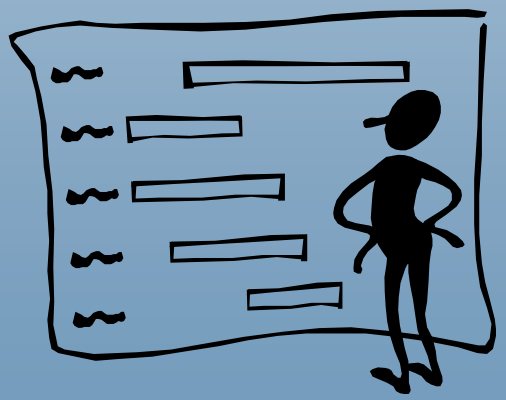
Time-and-Materials Contracts

Customer is at greatest risk

- ✓ Supplier has little incentive to complete the work
- ✓ Therefore need to control supplier opportunism

Enter: Project Control Processes

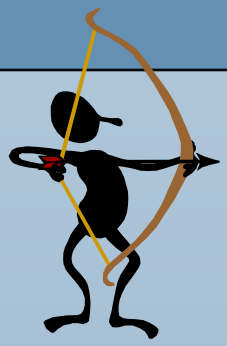
- ✓ Detailed Oversight by least knowledgeable party
- ✓ Supplier must justify every activity

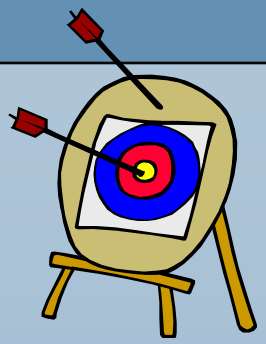


Most Project Control Processes

- ✓ Increase costs
- ✓ Do not add value
- ✓ Assume that the original plan is the optimal plan

Target Cost Contracts





Target cost

- ✓Target cost **includes all changes**
- ✓Target is the **joint responsibility** of both parties
- ✓Target cost is **clearly communicated** to workers
- ✓Negotiations occur if target cost is exceeded
  - × Neither party benefits

Workers at all levels have clear incentives to work collaboratively, compromise, and meet the target.

Remove Conflict of Interest.

Contract Format

Structure

- ✓ Start With An Umbrella or Framework Contract
- ✓ Establish a Target Cost
- ✓ Release Work In Stages
  - × Keep Stages Small
  - × Each Stage is an Iteration
- ✓ Scope Beyond the Existing Stage is Negotiable

Contract Form

- ✓ Describes the relationship, not the deliverables
- ✓ Sets up a framework for future agreements
- ✓ Provides for mediation if no agreement is reached

PDC2008
PROFESSIONAL DEVELOPERS CONFERENCE

take aways

(USE THIS SPACE FOR PRODUCT LOGOS WHEN WHITE BACKGROUND IS REQUIRED)

DELETE WHITE RECTANGLES IF NOT BEING USED

Take aways

- Welcome to the mainstream!
- Become lean
- Learn, Do, Reflect!
- Strive for both technical excellence + managerial excellence
- Adopt the empirical way
  - but watch out for dysfunctions caused by metrics
- Discover and distill right behaviors of your continual agility

Related Content

- Microsoft patterns & practices
  - msdn.microsoft.com/practices
- Mary's site:
  - poppendieck.com
- Grigori's blog:
  - blogs.msdn.com/agile

PDC2008
PROFESSIONAL DEVELOPERS CONFERENCE

agile unleashed

Panel

→ Mary Poppendieck
→ Tom Poppendieck
→ Grigori Melnik
→ Don Smith
→ Ajoy Krishnamoorthy

(USE THIS SPACE FOR PRODUCT LOGOS WHEN WHITE BACKGROUND IS REQUIRED)

DELETE WHITE RECTANGLES IF NOT BEING USED

Microsoft®

Your potential. Our passion.™