

Predictable Projects

How to get the Right Result at the Right Time

Niels Malotaux

N R Malotaux
Consultancy

+31-30-228 88 68

niels@malotaux.nl

www.malotaux.nl

Niels Malotaux

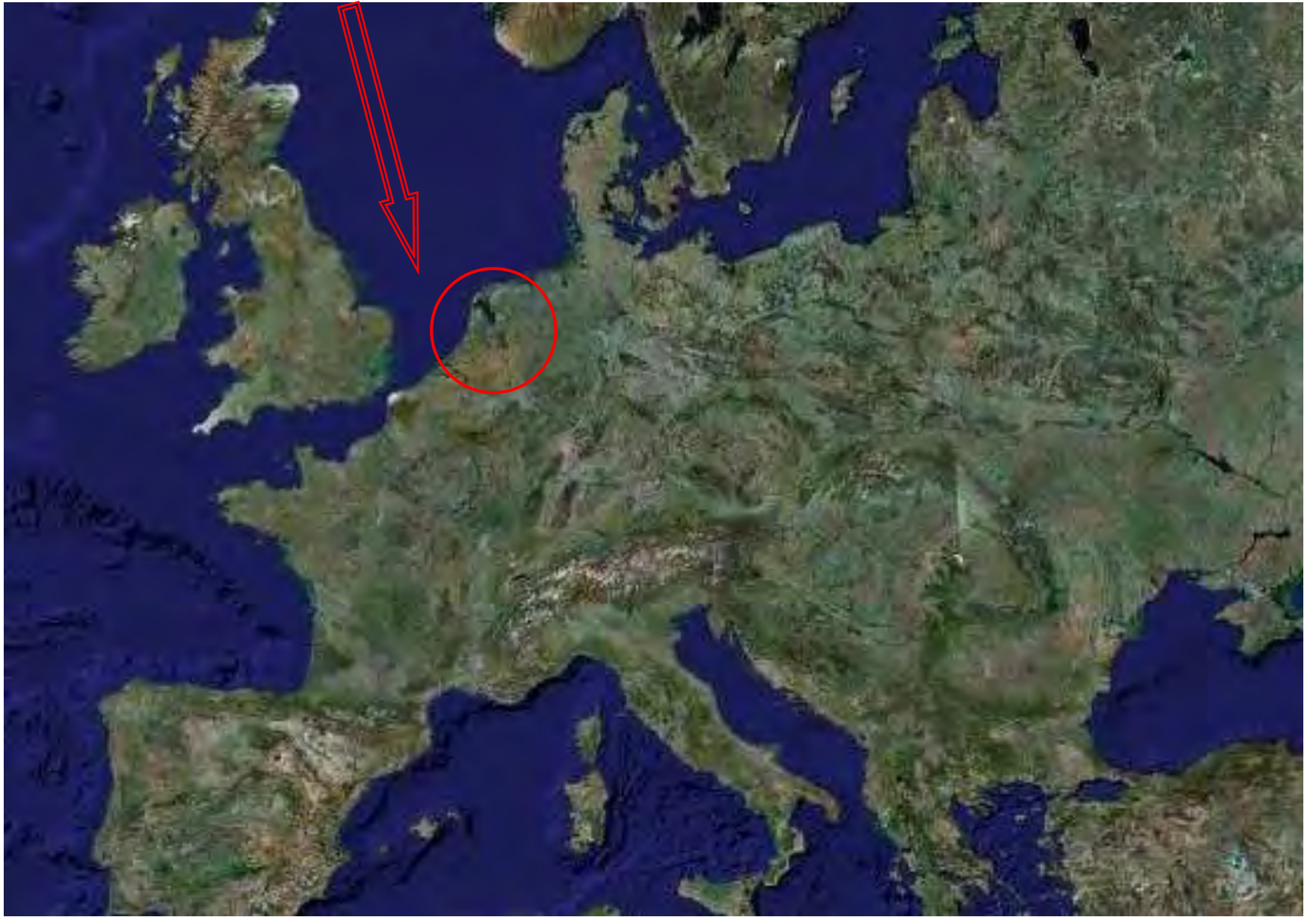
Result Management

- **Project Coach**

- Evolutionary Project Management (Evo)
- Requirements Engineering
- Reviews and Inspections

- **Researching problems in projects**
- **Finding ways for fundamentally overcoming these problems**
- **Ploughing back into projects**
- **Tuning of the results** (because theory isn't practice)

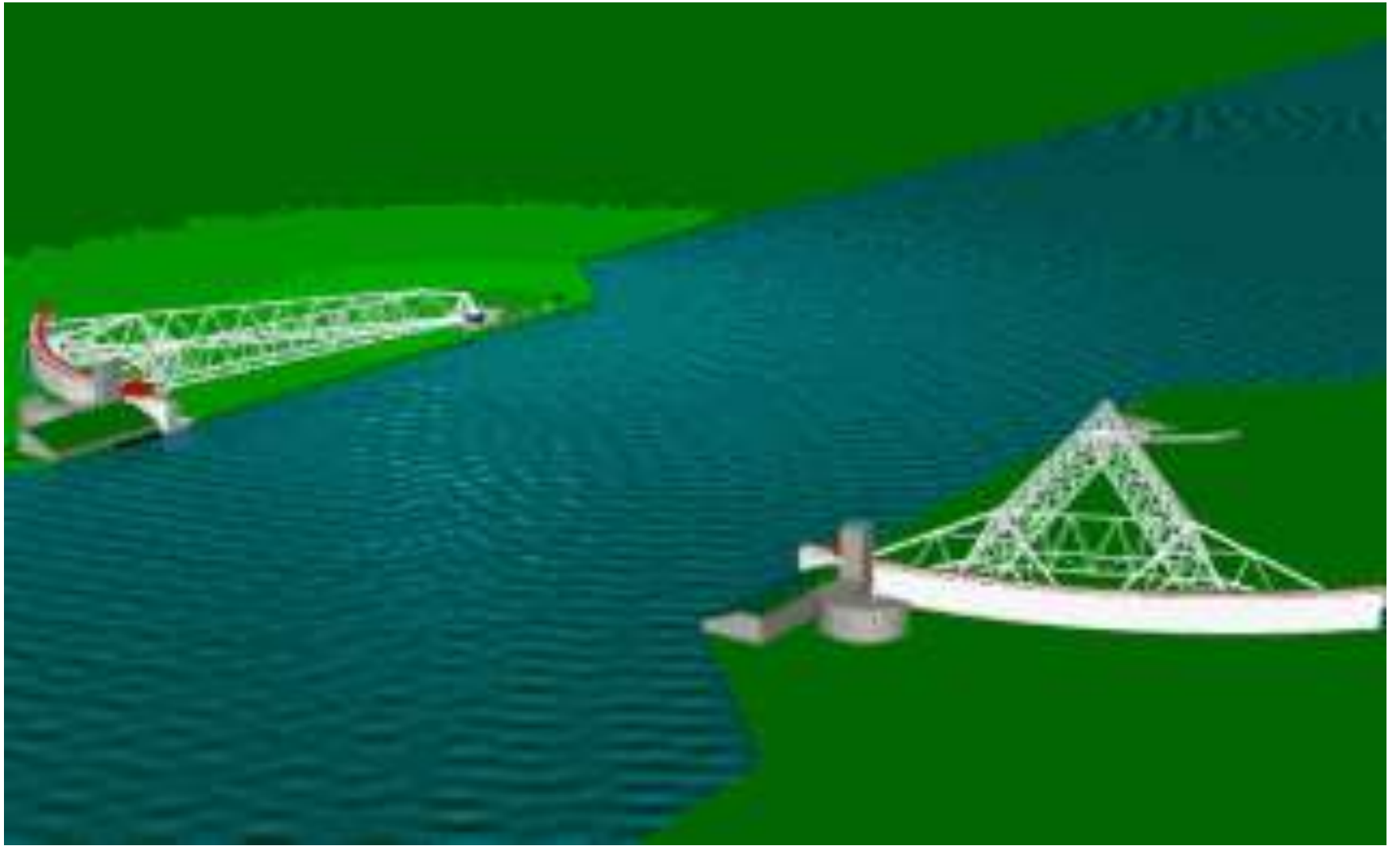


















Who are you ?

*

Projects

- **Who's working in projects?**
- **What is a project ?**
- **Who is a Project Manager ?**
- **Who is a Systems Engineer ?**
- **Who is something else ?**
- **Who has Project Result responsibility ?**

How good are you in your current work?

- **Average?**
- **Better than average?**
- **Below average?**

Predictable Projects ?

- **Any problems with projects ?**

Types of project ?

- **Process development**
- **Product development**
- **Software development**
- **Systems**
- **Systems of Systems**
- **... ?**

Time Important ?

- **Yes !**
- **Extremely yes !**
- **Yes, for our business case; no, customers don't care much**
- **Yes, it saves cost**
- **Yes, it's a requirement**
- **Yes, it's our commitment to the customer**

(previous workshop)

Things to Improve

- **Time is first priority after product quality**
- **How to make project management more efficient**
- **Systems Architecture and Design**
- **Project Management**
- **Time and Cost Management**
- **Documentation**
- **Predictable Schedule**
- **Communication**
- **Quality of requirements and final products**
- **Project quality**

(previous workshop)

Quality On Time

(previous workshop)

Successful	On Time	Need improvement
Quality and cost OK	81% overrun	Time is first priority
Normally yes, with good plan and people	Normally yes	More efficient PM
Yes	Yes	Improve Design Competence and Planning
Yes, customer happy	Yes	Time/Cost Management
Yes	No, many changes, component delay (Customers don't mind)	Documentation, Schedule
Result OK	Later (req/staff changes)	Predictable schedules
Some	Later	Schedule
No. Long project, change req → bad product quality	Maybe	Quality of requirements and results
Usually	Usually or bit later	Project quality

How to improve

- **CMMI**
- **Project planning tool**
- **Skill of team on Project Management**
- **Improve design competence**
- **Good planning**
- **Don't know**
- **Breaking down to small tasks**
- **Stable staffing**
- **Stick with requirements; better prioritizing**
- **Using PMBOK**
- **Iterative delivery & customer cooperation**

(previous workshop)

The Right Result at the Right Time

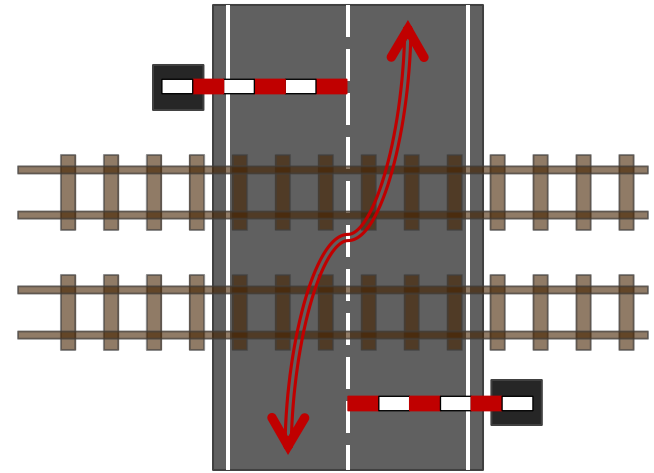
- **Do you regularly deliver the Right Result at the Right Time ?**
 - **Why not ?**
 - **Is this normal ?**
 - **Can we do something about it ?**
-
- **What is the Right Result ?**
 - **What is the Right Time ?**

Not every project is successful (at first)



- Apparently we're doing something wrong
- Otherwise projects would succeed and be on time
- Heathrow Terminal 5: “Great success !”
 - Normal people aren't interested in the technical details of a terminal
 - They only want to check-in their luggage as *easily* as possible and
 - Get their luggage back as *quickly* as possible in *acceptable condition at their destination*
 - They didn't
- One of the problems is to determine what the project (or your work in general) really is about

AHOB (Automatic Half Barrier Crossing)





ADOB

(Automatic Double Barrier Crossing)

1 train every 4 minutes

**Few years of trouble
before some stability**

At >22°C still trouble

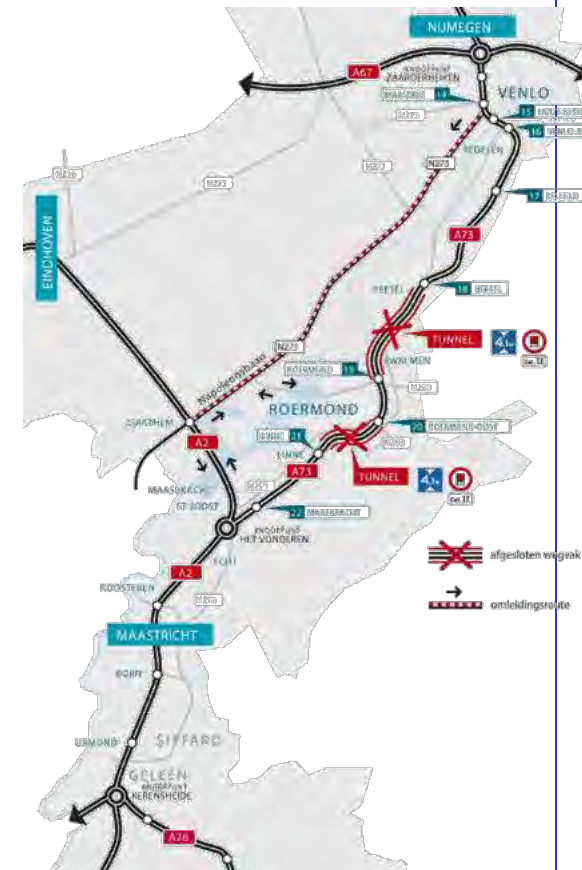
**Why it didn't work
is irrelevant**

**What we deliver
should simply work
Is that so difficult?**



Tunnels in the Netherlands

- 18-02-2008: tunnels open for traffic. A73 (42km) is now complete ! (decision 1975)
- V&V said “NO!” → Lots of trouble
- 03-03-2008: tunnels are finally open, after safety tests concluded
- 10-03-2008: tunnels are finally open, after safety tests concluded
- 06-06-2008: Coming months we're working on completion of the A73 tunnels
- 24-09-2008: start of completion in January 2009 and/or April 2009
- 24-09-2009: Completion runs as planned. Tunnels will be closed from 1 October. 1 December the tunnels will finally open
- 28-01-2010: Tunnels will have weekend closures for “regular maintenance”



Incredible Public Transport Chip-Card



Other messages:

- Invalid card
- Err. 034



The problem

- **Many projects don't deliver the right Results**
- **Many projects deliver late**

or, more positively:

- **I want my project to be more successful**
- **In shorter time**

Do we mind?

- **Does anybody mind**
 - projects being late
 - delivering inferior quality
 - costing too much ... ?

Can we afford it?

- **Can we afford**
 - projects being late
 - delivering inferior quality
 - costing too much ?

- **Finally we all pay !**
- **What are we going to do about it ?**

Goals

- **Knowing how you can optimize the Results of your daily work**
- **How to optimize the Results of your projects**
- **Creating a desire to start using this knowledge immediately**

Predictable Projects

How to Get Quality On Time

- Introduction
- Is Culture an Issue ?
- Quality On Time
- Human Behavior
- Project Life Cycles
- Evolutionary Principles
- Evolutionary Planning
- Business Case
- Stakeholders & Requirements
- Design & Architecture
- Risk
- V&V / Testing / QA
- Reviews & Inspections
- Metrics
- Some Management Issues
- Introduction Issues

Ultimate Goal of a Project

- **Delivering the Right Result at the Right Time, wasting as little time as possible** (= efficiently)

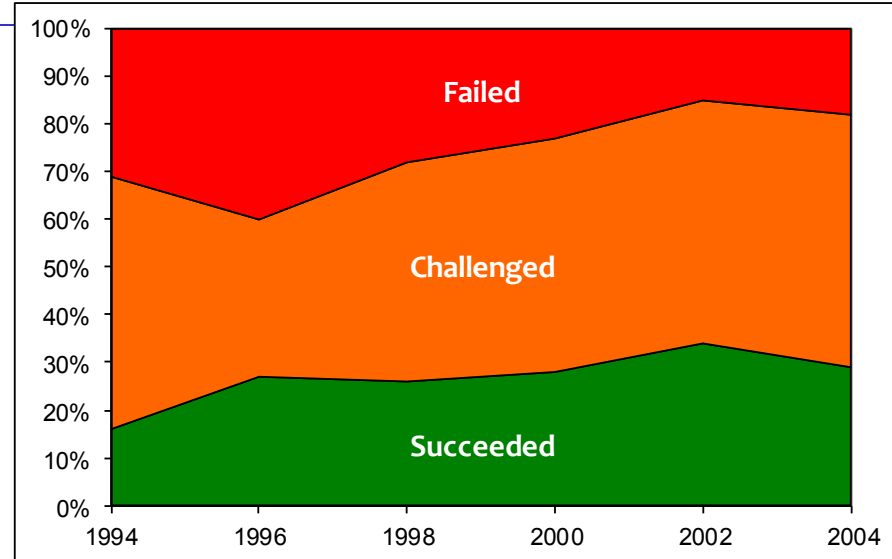
- **Providing the customer with**
 - what he needs
 - at the time he needs it
 - to be satisfied
 - to be more successful than he was without it
- **Constrained by** (win - win)
 - what the customer can afford
 - what we mutually beneficially and satisfactorily can deliver
 - in a reasonable period of time

What are you providing your customer ?

- **Who is your customer ?**
 - **What does he need ?**
 - **When does he need it ?**
 - **Will he be happy with it ?**
 - **Will he be more successful ?**
 - **Can the customer afford it ?**
 - **Is it win-win ?**
- **Providing the customer with**
 - what he needs
 - at the time he needs it
 - to be satisfied
 - to be more successful than he was without it
 - **Constrained by (win - win)**
 - what the customer can afford
 - what we mutually beneficially and satisfactorily can deliver
 - in a reasonable period of time

What's wrong with projects ?

2/3 of IT projects still fail on Quality On Time



		1994	1996	1998	2000	2002	2004	2009
Succeeded	delivered on time, on budget, with required features and functions	16%	27%	26%	28%	34%	29%	32%
Challenged	late, over budget and/or with less than the required features and functions	53%	33%	46%	49%	51%	53%	44%
Failed	cancelled prior to completion or delivered and never used	31%	40%	28%	23%	15%	18%	24%

Standish Group International

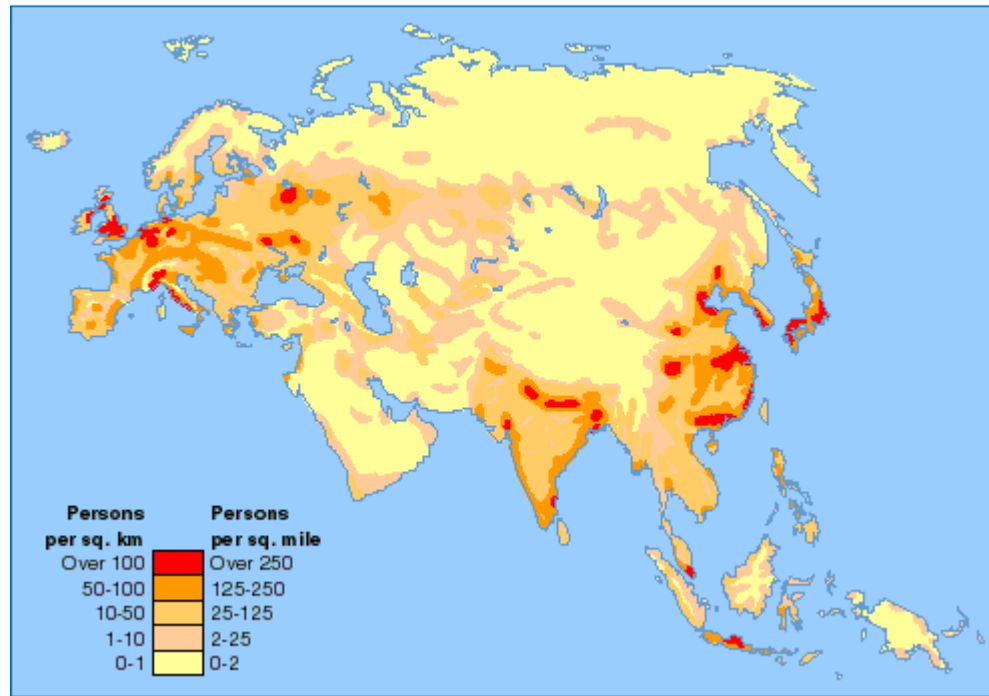
However

- “Succeeded” projects actually were late from the beginning: Management told that they multiplied “best guess” by 2.5
- Perhaps the “Failed” projects were killed early for a good reason

Top 5 success factors:

1. **Executive Support**
2. **User Involvement**
3. **Experienced Project Manager**
4. **Clear Business Objectives**
5. **Minimized Scope**

Is Culture an Issue ?



Culture

- **Latin: Cultus - adoration, worship**
- **Culture: Ingrained customs**
 - Things we learn by mimicking what we experience around us
 - Language
 - Social behavior
 - Faith, religion
 - Folklore
 - Doing what we're used to
 - We don't really know why we do it, or even that we do it; we just do it
 - Experience → intuition → culture
 - Not genetic (that would be *instinct*)
- **Once we see other cultures, we can see that our own culture isn't obvious at all; neither is theirs**
- **Still we judge others through our own cultural spectacles, whether we like it or not**

Cultural differences ?

influences on project results ?

Dutch

- open, direct, explicit, blunt
- informal
- arrogant
- preaching
- assertive
- can say no
- egalitarian, not showing wealth
- little power distance
- authority must be earned
- little brand value
- not spending more than necessary
- consensus
- win-win

Japanese ?

- ...
- ...
- ... ?

Things I heard

- **Authority**

- Boss is always right
- Teacher is always right

They are just doing their best. Lot of experience. However, are they perfect ?

- **Group is important**

- Project team is a group; organization is a group
- Self ?

- **Group is responsible**

- No personal responsibility ?
- Should we hide for responsibility ?

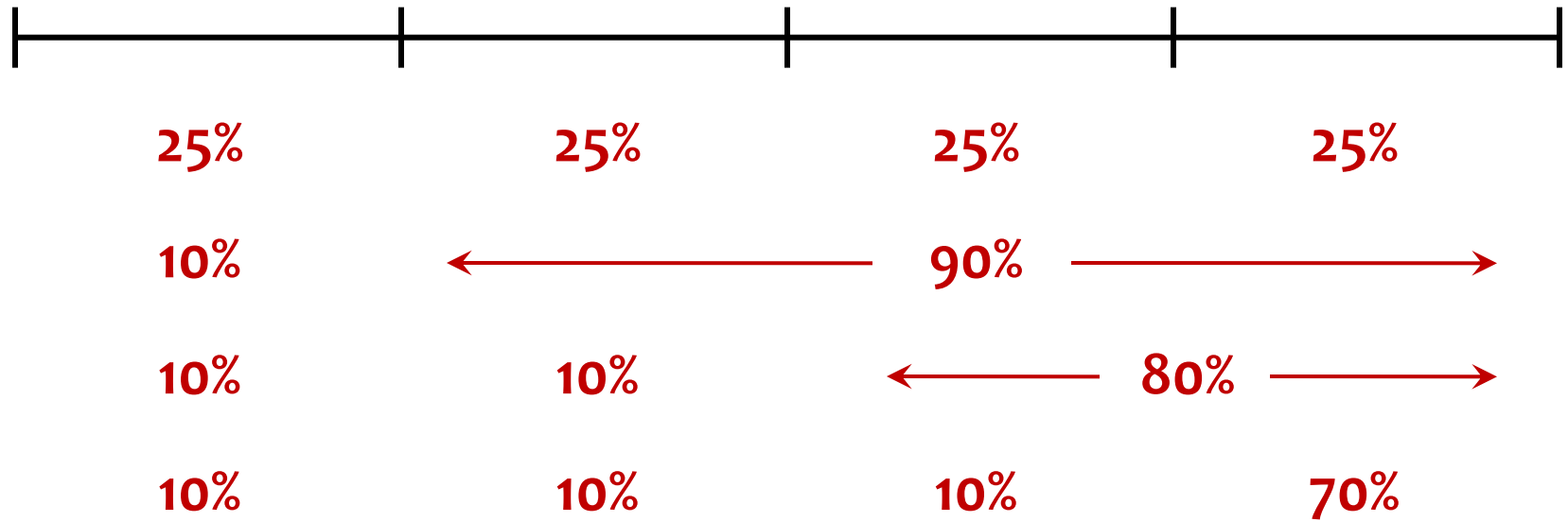
Things I heard (2)

- **Losing 'face'**
 - We are not perfect, but the customer should never find out
- **Cannot say 'No'**
 - How do you then say 'no' ?
(in Holland we say: "Yes, but ...")
- **Is that clear? - Yes**
 - 'Yes' isn't always 'Yes'
 - If you don't understand:
 - Is the teacher unclear ?
 - Am I stupid ?

The boss is always right

- Is he or she ?
- Afraid for losing 'face' ?
- How about losing face *invisibly* ?
(you don't say it, but you know)
- Would you like that if you were a boss ?
- How do we tell, without losing 'face' ?
- Should we ?
- Is it my culture ?

4 week project



Is culture a risk for projects ?

If we want to be winners

- **People make mistakes**
- **We are people**
- **We make mistakes**
- **Mistakes cause problems**
- **We don't want problems**

- **Let's uncover our mistakes as quickly as possible, so that we can do something about it**
- **Let's help each other**
- **We cannot help each other, if we don't know**

Quality on Time

The Right things at the Right time

Quality On Time

- **Whatever we do in a project,
at a certain moment there should be a Result**



- **How do we get the Right Result at the Right Time?**
- **Or shorter: Quality On Time**
- **What the Customer needs, when he needs it,
to earn more than we need**

Quality On Time

- **What is Quality?**
- **What is On Time?**

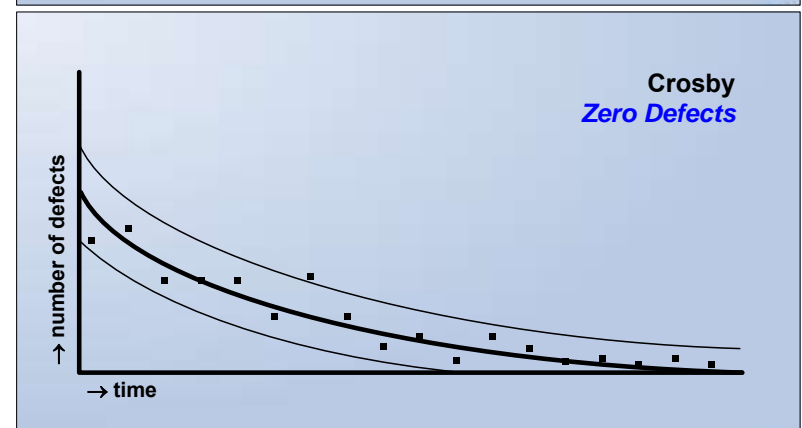
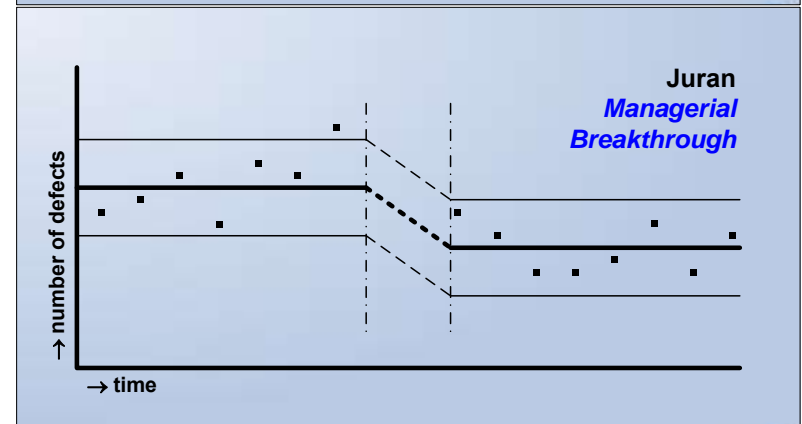
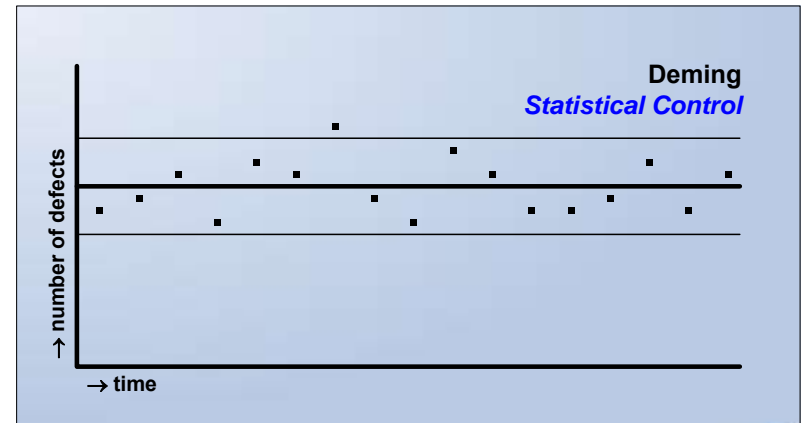
Quality - the Right Results

- I know it when I see it ...?
- Should be *measurable*
- Should be *predictable*
- But ...
ultimately they must like it when they see it

Quality guru's

- **Shewhart** - Economic Control of Quality 1930
- **Deming** - Japan 1950, Out of the crisis 1986
- **Juran** - Japan 1954, Quality handbook 1951
- **Crosby** - Zero Defects 1961, Quality is Free 1979
- **Imai** - Kaizen 1986, Gemba Kaizen 1997

Deming - Juran - Crosby



Deming

- **Quality comes not from inspection (V&V), but from *improvement of the production process***
- **Inspection does not improve quality, nor guarantee quality**
- **It's too late**
- **The quality, good or bad, *is already in the product***
- ***You cannot inspect quality into a product***

Absolutes of Quality

- **Conformance to requirements**
- **Obtained through prevention**
- **Performance standard is zero defects**
- **Measured by the price of non-conformance (PONC)**

Philip Crosby, 1970

- **The purpose is customer success (not customer satisfaction)**

Added by Philip Crosby Associates, 2004

The Absolutes of Quality Management™

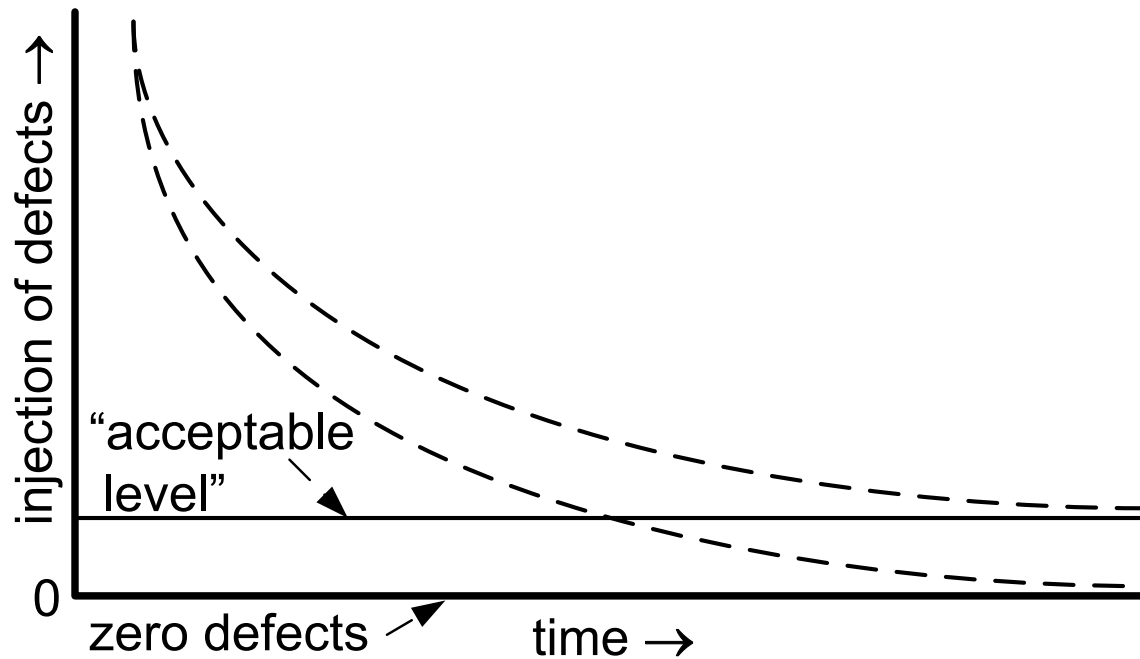
- 1 Quality has to be defined as conformance to requirements, not as goodness.
- 2 The system for causing quality is prevention, not appraisal.
- 3 The performance standard must be Zero Defects, not "that's close enough."
- 4 The measurement of quality is the Price of Nonconformance™, not indexes.
- 5 The purpose of quality is to create customer success, not customer satisfaction.

Philip Crosby / Associates™

Quality Means Profitability

Is Zero Defects possible?

- **Zero Defects is an asymptote**

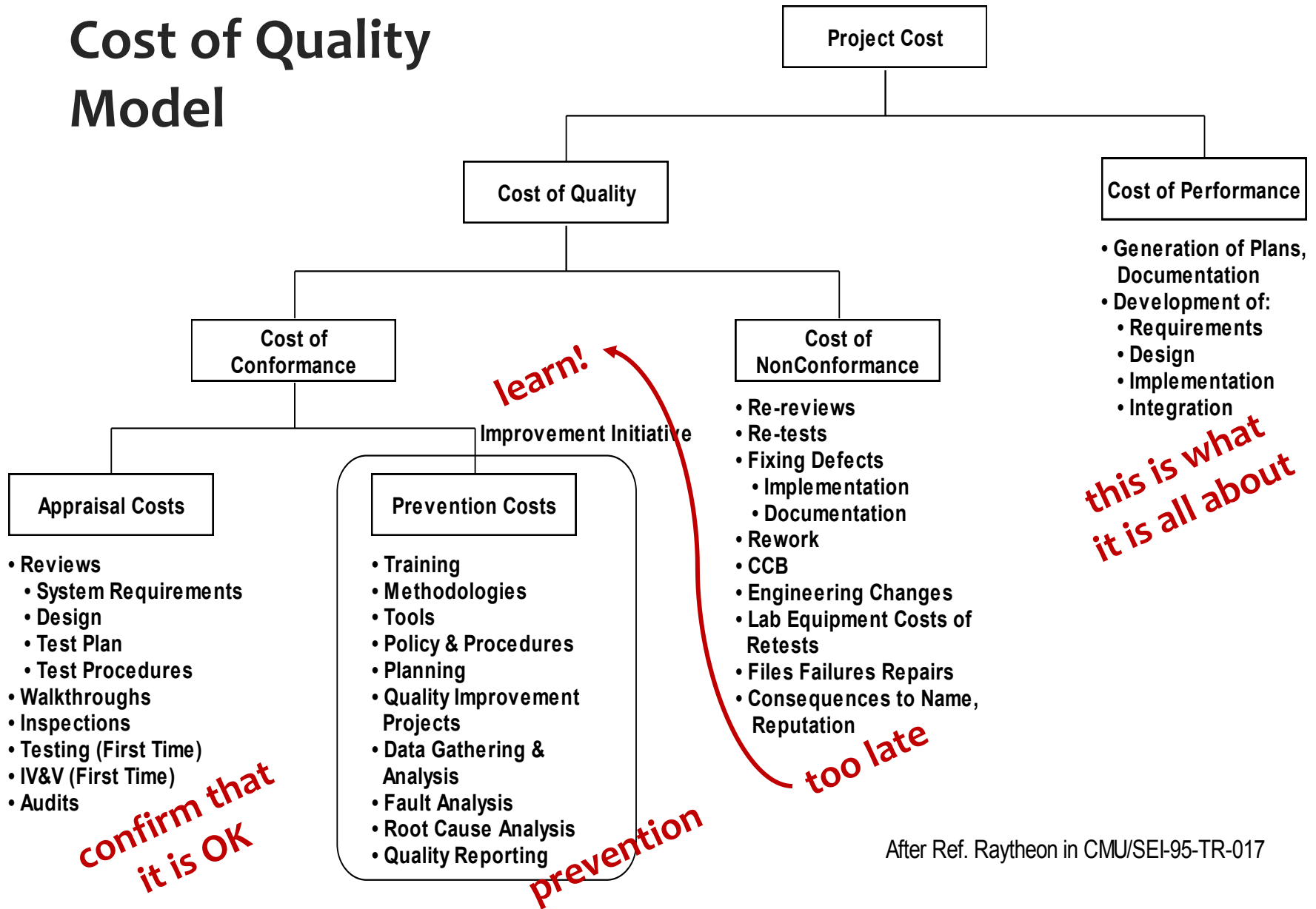


- **When Philip Crosby started with Zero Defects in 1961, errors dropped by 40% almost immediately**

Attitude

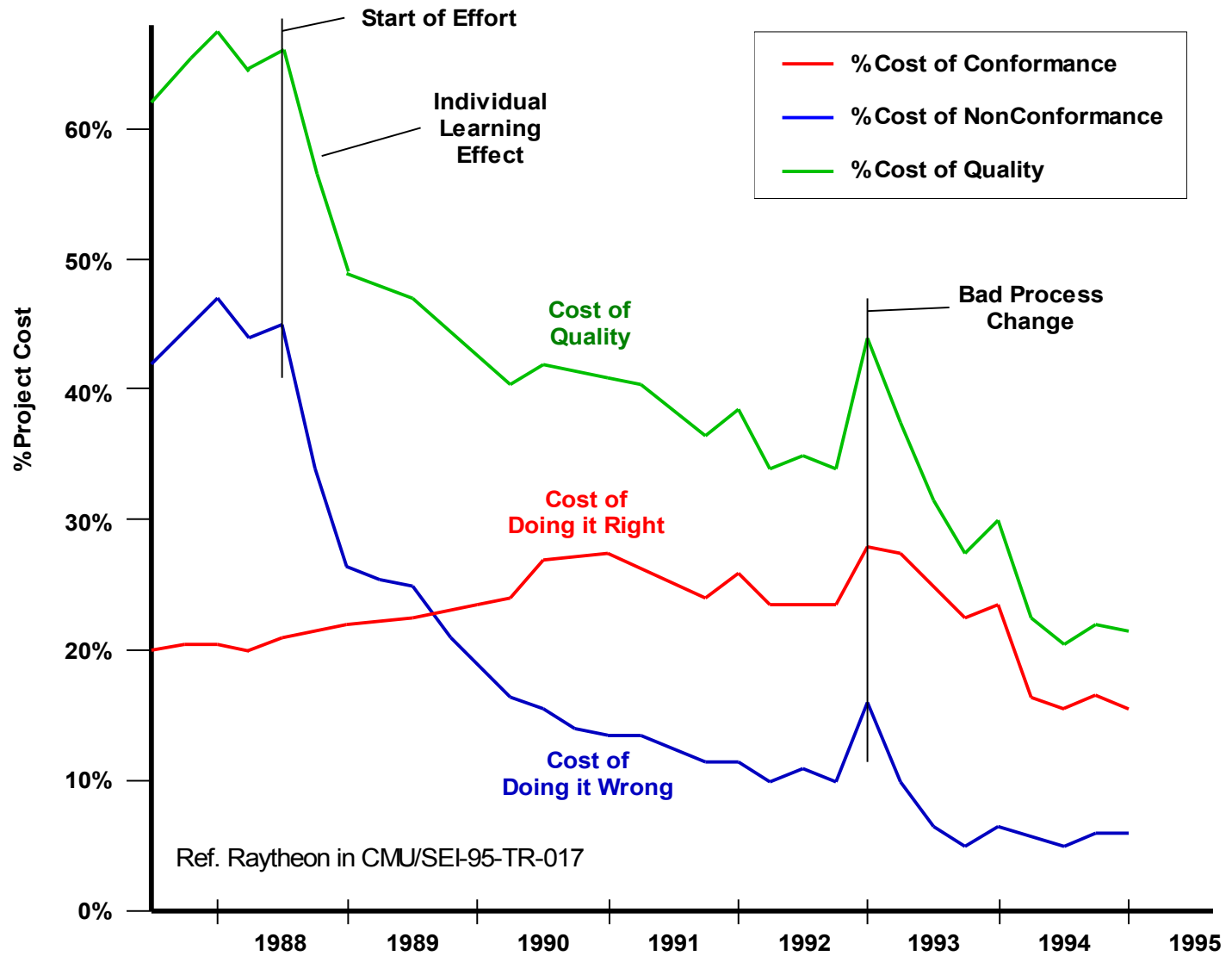
- **As long as we think Zero Defects is impossible, we will keep producing defects**
- **From now on, we don't want to make mistakes any more**
- **We feel the failure (if we don't feel failure, we don't learn)**
- **If we deliver a result, we are sure it is OK and we'll be highly surprised when there proves to be a defect after all**
- **We do what we can to improve (continuous improvement)**

Cost of Quality Model



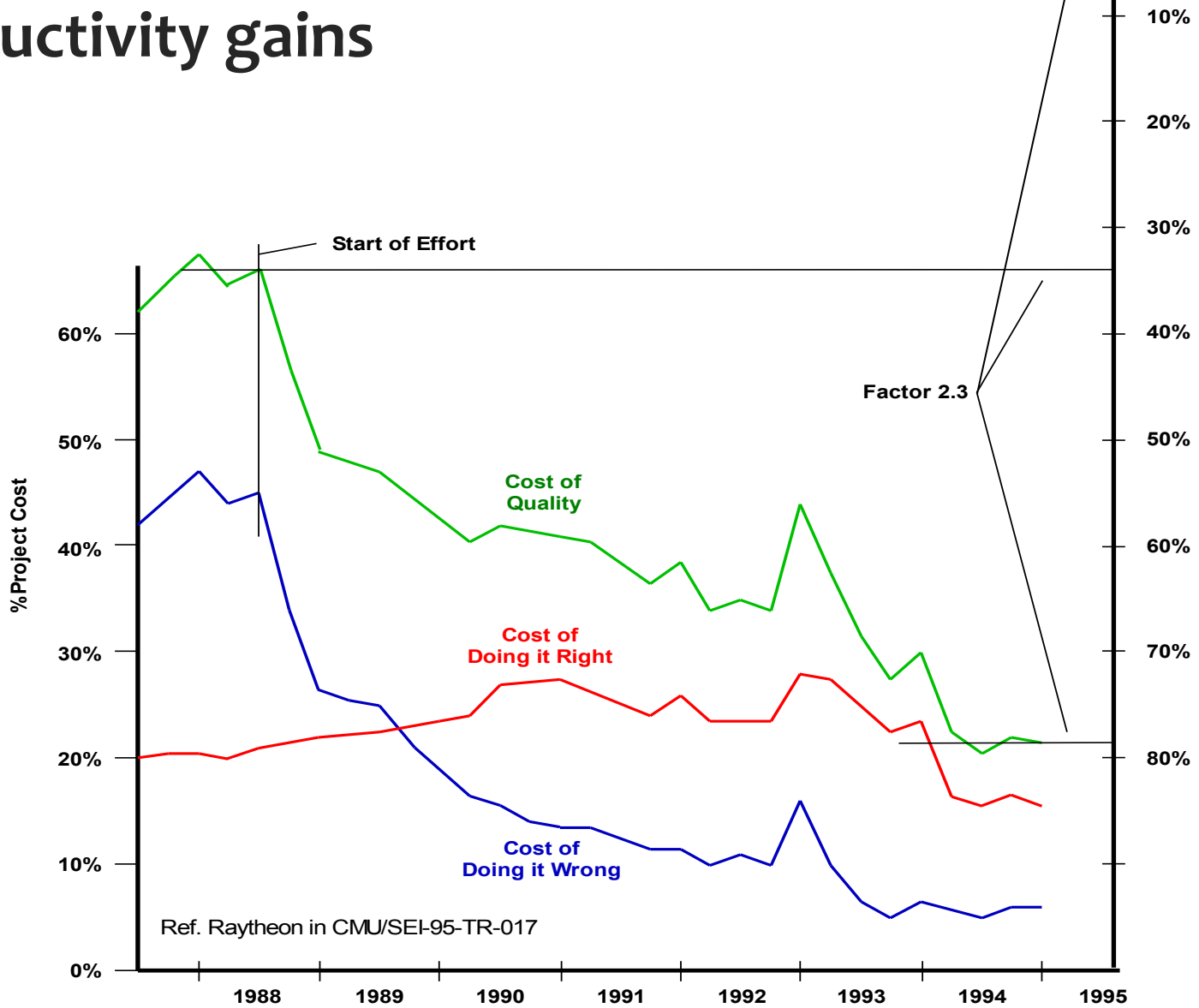
After Ref. Raytheon in CMU/SEI-95-TR-017

Cost of Quality



Ref. Raytheon in CMU/SEI-95-TR-017

Productivity gains



Peter Drucker

**Quality in a service or product is not what you put into it
It is what the client or customer gets out of it**

On Time

- **Yesterday?**
- **Before the next exhibition?**
- **Managers dream?**
- **Time to market?**
- **Time to profit?**

**Compromise between what is needed
and what is possible**

Are you Serious about Time?

- **Is Time Important ?**
- **Do you mind time ?**
- **Does your boss mind time ?**
- **Does your customer mind time ?**
- **Are you always on time ?**

Is it difficult to be on time ?

- **Did anyone miss a plane ?**
- **What did you feel ?**
- **Why did it happen ?**
- **Did it happen again ?**

Time as a Requirement

- **Delivery Time is a Requirement, like all other Requirements**
- **How come most projects are late ???**
- **Apparently all other Requirements are more important than Delivery Time**

- **Are they really?**



Fallacy of 'all' requirements

- “We’re done when all requirements are implemented”
- Isn’t delivery time a requirement ?
- Requirements are always *contradictory*
- Perception of the requirements
- Who’s requirements are we talking about ?
- Do we really know the *real* requirements ?
- Are customers able to define requirements ?
 - Customers specify things they do not need
 - And forget things they do need
 - They’re even less trained in defining requirements than we are
- What we think we have to do should fit the available time
- Use the Business Case

Will your current project be on time ?

- Was your previous project successful *and* on time ?
- Will your current project be successful *and* on time ?
- How do you know ?

**If our previous project was late,
our current project will also be late**

unless we do things *differently* and *better*

**If we don't learn from history,
we are doomed to repeat it**

**Projects don't have to be late
They deserve better**

But we're not Project Managers !

- **What caused the project being late ?**
- **Could we have prevented the project being late ?**
- **Was delivery time important ?**
- **Was delivery time a requirement ?**
- **Were all other requirements really more important ?**

What could we have done to save time?

*

Who's Responsible for the Result of the Project ?

- **The Project Manager is responsible for delivering the right result at the right time**
- **The Project Workers work and decisions determine the result and the time it is delivered**
- **This makes everybody in the project implicitly as responsible as Project Management**

Types of Project Management

1. **There is no project leader**
2. **He does not know, others don't know or nobody knows what it means**
3. **Project follower:
Hopes it will get on track eventually**
4. **Project leader: vision, strategy, scenario's, first time right, zero defects, time to market: makes it happen**

**Projects without project leader fail
(even one-person projects !)**

Projects with more than one project leader also fail

Architect ↔ Project Manager

- **Architect: Master Builder**
- **Architect is the conductor of the Product**
- **Project Manager is the conductor of the Project**
- **There is only one captain on the ship:
the Project Manager**
- **Test Manager is the conductor of the Test Process**
- **Systems Engineer is a kind of Architect**

PM and SE/Architect are like Conductors



- **Project Manager is the conductor of the *Project***
 - Vision and techniques to organize the project
- **Systems Engineer/Architect is the conductor of the *Product***
 - Vision and techniques to realize the product
- **However, there should be only *one captain on the ship***

Systems Engineers

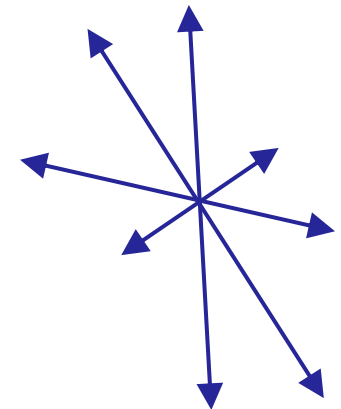
- **Other Engineers (?)**

- Silo thinking
- Sub-optimizing
- Gold plating (hobbies)
- Little attention to interfaces
- Projects are always *multidisciplinary*



- **Systems Engineers**

- Multi-dimensional thinking
- Optimizing design decisions over all dimensions
- Whole life-cycle (cradle to cradle)
- Balancing requirements
- Including delivery time
- All disciplines → *interdisciplinary*



Multidisciplinary ↔ Interdisciplinary

- **Tension between**
 - Technologically possible
 - Economically profitable
 - Socially and psychologically acceptable
 - All kinds of disciplines needed for a good solution
- **Multidisciplinary**
 - Many disciplines work in the project
 - Optimize solution in their own domain
- **Interdisciplinary**
 - Many disciplines work *together* in the project
 - Overall-optimizing
 - First *developing the problem* before developing the solution

Causes of Delay



- **Some typical causes of delay are:**

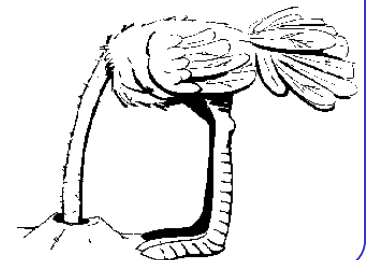
- Developing the wrong things
 - Unclear requirements
 - Misunderstandings
 - No feedback from stakeholders
 - No adequate planning
 - No adequate communication
 - Doing unnecessary things
 - Doing things less cleverly
 - Waiting (before and during the project)
 - Changing requirements
 - Doing things over
 - Indecisiveness
 - Suppliers
 - Quality of suppliers results
 - No Sense of Urgency
 - Hobbying
 - Political ploys
 - Boss is always right (culture)
- **Excuses, excuses: it's always "them". How about "us" ?**
 - **A lot of delay is avoidable and therefore unjustifiable**

The challenge

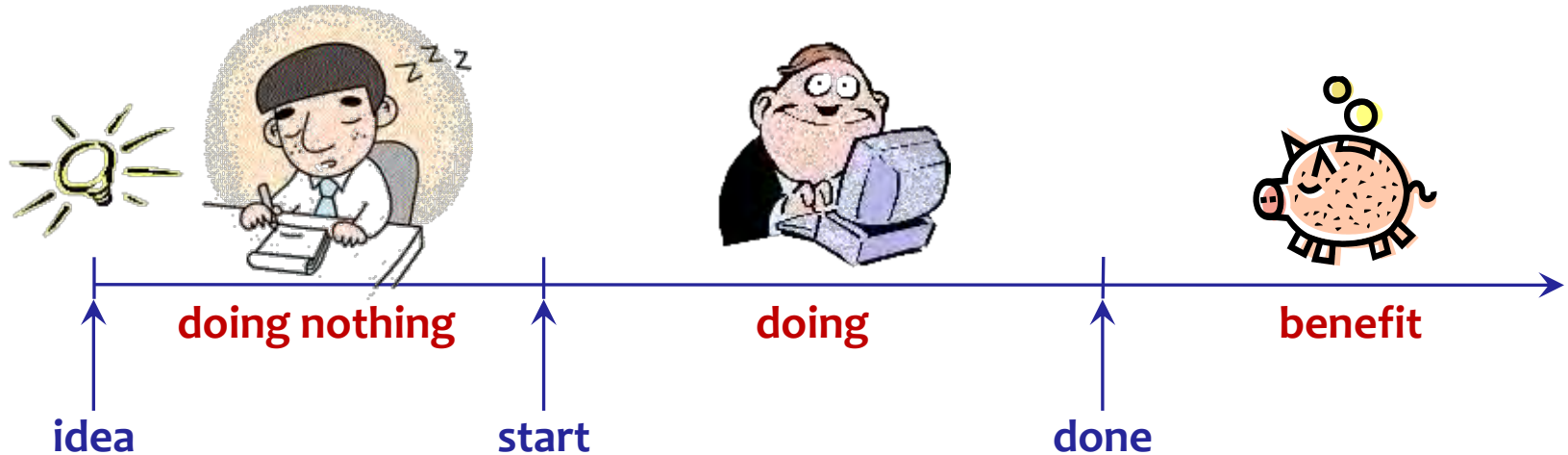
- Getting and keeping the project under control
- Never to be late
- If we are late, we *failed*
- No excuses when we're not done at the FatalDay
- Not stealing from our customer's (boss) purse
- The only justifiable cost is the cost of developing the right things at the right time
- The rest is *waste*
- Would we enjoy producing waste ?

FatalDay

- FatalDay is the last moment *it shall be there*
- After the FatalDay, we'll have real trouble if the Result isn't there
- Real Option Theory says that we should do things as late as possible, but not later
 - As late as possible, having the most up-to-date information to decide what to do
 - Not later: the option has expired; it has no value any more
- Count backwards from the FatalDay to know when we should have started
- If that's before now, what are we going to do about it, because *failure is not an option*



Project ROI



Return on Investment (ROI)

- + **Benefit of doing** - huge (otherwise other projects would be more rewarding)
- **Cost of doing** - project cost, usually minor compared with other costs
- **Cost of doing nothing** - every day we start later, we finish later
- **Cost of being late** - lost benefit

Time to market

- 5000 products per year \approx 20 products per day
- € 5000 per product
- Profit € 500 per product
- Profit € 10.000 per day

**Every day we start later, we'll be done a day later
and miss € 10.000**

Cost of one day of delay

- **Do you know how much you cost per day?**

Note: that's not what you get !

- **New electronic measuring instrument**

- 40 people in Oregon, US
- 8 people in Bangalore, India

- **US\$ 40,000 per day for the project**

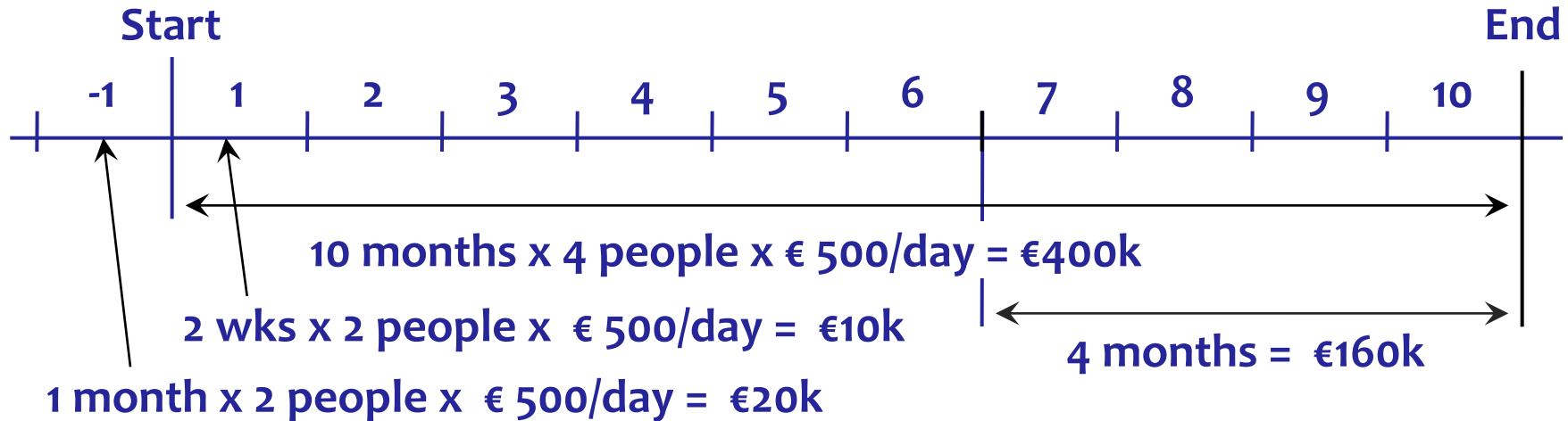
- **Plus US\$ 30,000 per day for lost benefit**

- **Total: US\$ 70,000 per day for every day of (unnecessary) delay**

- **0th order estimations are good enough**



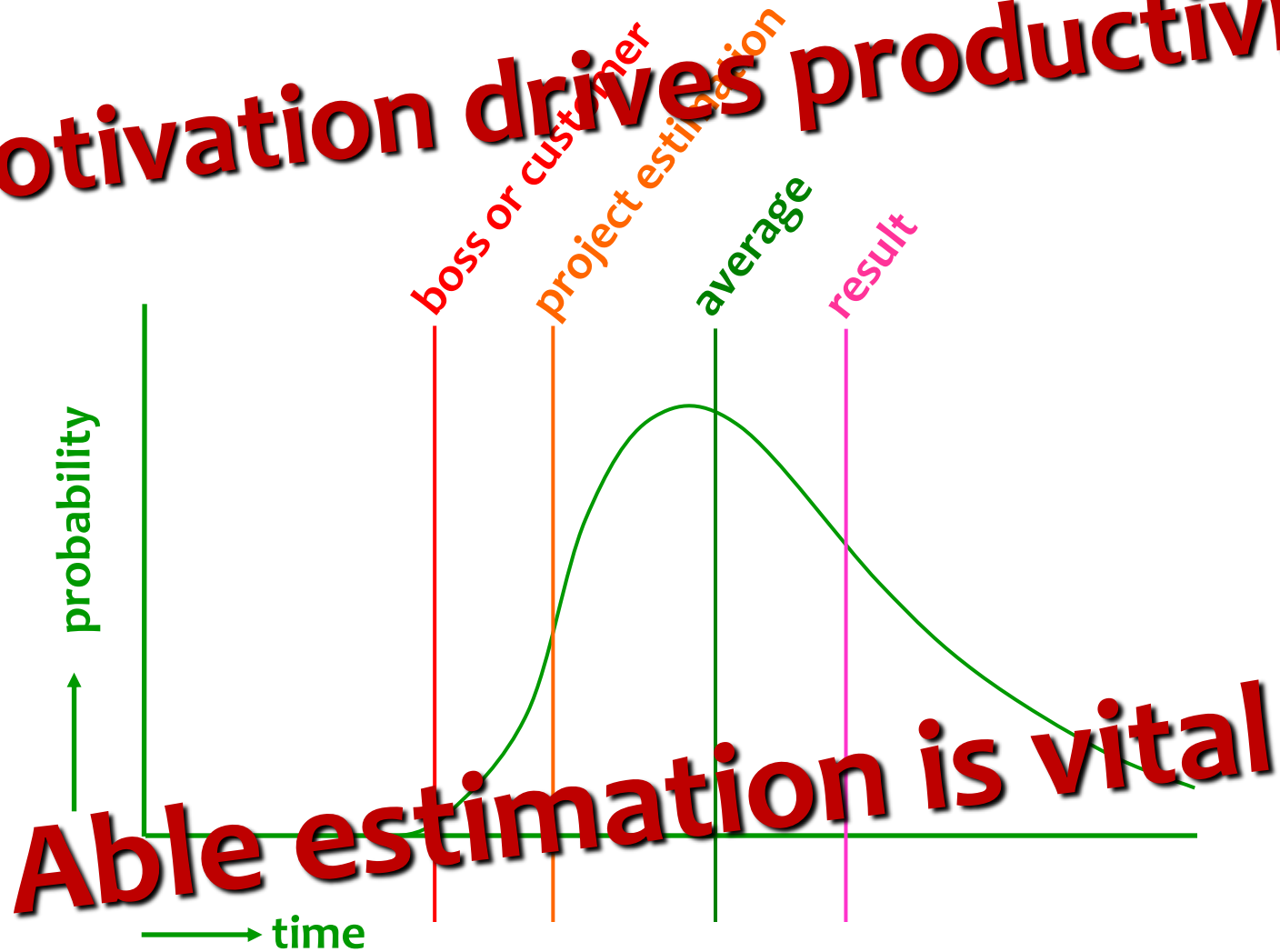
The Cost of Time



- We can save 4 months by investing €200k → “That’s too much !”
 - It’s a *nicer* solution - Let’s do 2 weeks more research on the benefits
 - What are the expected revenues when all is done? → €16M/yr (1.3M/mnd)
 - So 2 weeks extra doesn’t cost €10k, but rather €16M/24 = €670k
 - And saving 4 months brings €16M/3 = €5M extra
- Invest that €200k *NOW* and don’t waste time !

Lead time

Motivation drives productivity



Estimation Exercise



Are you an optimistic or a realistic estimator?

Let's find out !

Project:

Multiplying two numbers of 4 figures

How many seconds would you need to complete this Project?

Is this what you did?

Defect rate

- **Before test ?**
- **After test ?**

Alternative Design (*how to solve the requirement*)

Another alternative design

*There are usually more,
and possibly better solutions
than the obvious one*

What was the real requirement?

Assumptions, assumptions ...

Better assume that many assumptions are wrong.

Check !

Elements in the exercise

- **Estimation, optimistic / realistic**
- **Interrupts**
- **Test, test strategy**
- **Defect-rate**
- **Design**
- **Requirements**
- **Real Requirements**
- **Assumptions**

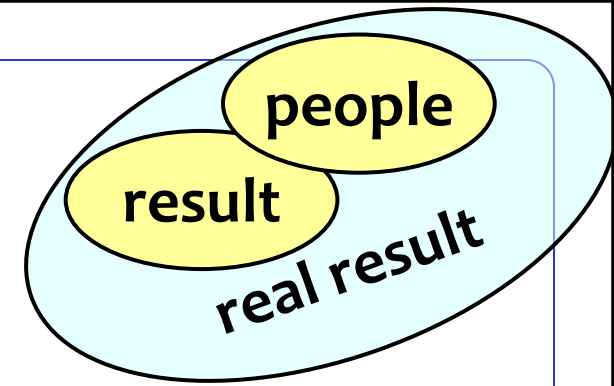
Human Behavior

Human Behavior

- **Systems are conceived, designed, implemented, maintained, used, and tolerated (or not) by people**
- **People react quite predictably**
- **However, often differently from what we intuitively think**

- **Most project process approaches (PMI, INCOSE, as well as developers)**
 - **ignore human behavior,**
 - **incorrectly assume behavior,**
 - **or decide how people should behave (ha ha)**
- **To succeed in projects, we must study and adapt to real behavior rather than assumed behavior**
- ***Even if we don't agree with that behavior***

Is Human Behavior a risk?



- **Human behavior is a risk for the success of the system**
 - When human behavior is incorrectly modeled in the system
 - Not because human users are wrong
- **Things that can go wrong**
 - Customers not knowing well to describe what they really need
 - Users not understanding how to use or operate the system
 - Users using the system in unexpected ways
 - Incorrect modeling of human transfer functions within the system: ignorance of designers of systems engineers
- **Actually, the humans aren't acting unpredictably**
 - Because it happens again and again
 - Human error results from physiological and psychological limitations of humans

Human Behavioral Inhibitors



- **No Sense of Urgency**
- **Indifference**
- **(lack of) Discipline**
- **Intuition**
- **Fear of Uncertainty**
- **Fear of Perceived Weakness**
- **Fear of Failure**
- **Perceived lack of time**
- **(lack of) Zero Defects attitude**
- **Ignorance**
- **Incompetence**
- **Politics**

People responsible for success

- **During the project**
 - Can still influence the performance of the project
 - First responsibility of the Project Manager
 - Actually responsibility of the whole development organization
- **After the project, once the system is out there**
 - No influence on the performance of the system any more
 - System must perform autonomously
 - So the performance must be there *by design*
 - Including appropriate interface with humans
 - Responsibility and required skill of Systems Engineering

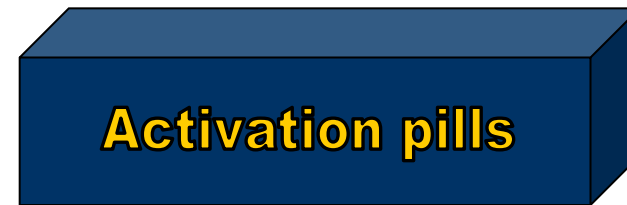
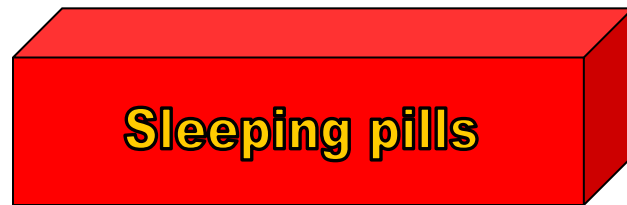
Discipline

- **Control of wrong inclinations**
 - **Even if we know how it should be done ...**
(if nobody is watching ...)
 - **Discipline is very difficult**
 - **Romans 7:19**
 - The good that I want to do, I do not ...
-
- **Helping each other** (watching over the shoulder)
 - **Rapid success** (do it 3 weeks for me...)
 - **Making mistakes** (provides short window of opportunity)
 - **Openness** (management must learn how to cope)

Intuition

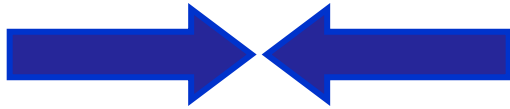
- **Makes you react on every situation**
- **Intuition is fed by experience**
- **It is free, we always carry it with us**
- **We cannot even turn it off**
- **Sometimes intuition shows us the wrong direction**
- **In many cases the head knows, the heart not**
- **Coaching is about redirecting intuition**

Is intuition wrong, or is the design wrong ?

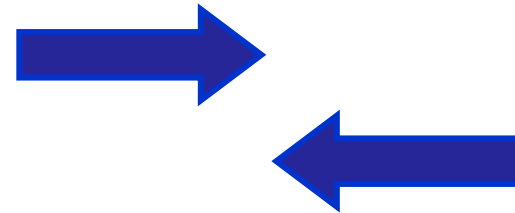


Communication

- Talking as near as possible along each other



To each other



Along each other

- Don't *assume* we understand: *check* !

Communication

- **Traffic accident: witnesses tell *their* truth**
- **Same words, different concepts**
- **Human brains contain rather fuzzy concepts**
- **Try to explain to a colleague**
- **Writing it down is explaining it to paper**
- **If it's written it can be discussed and changed**
- **Vocal communication evaporates immediately**
- **E-mail communication evaporates in a few days**

Perception



- **Quick, acute, and intuitive cognition** (www.M-W.com)
- **What people say and what they do is not always equal**
- **The head knows, but the heart decides**
- **Hidden emotions are often the drivers of behavior**
- **Customers who said they wanted lots of different ice cream flavors from which to choose, still tended to buy those that were fundamentally vanilla**
- **So, trying to find out what the real value to the customer is, can show many paradoxes**
- **Better not simply believe what they say: check!**

Responsibility

- **Taking responsibility - commitment**
- **Getting responsibility - empowerment**
- **Understanding responsibility**
- **Giving back responsibility**

Culture



- **It failed because of the existing culture**
(no good excuse !)
- **Culture is the *result* of how people work together**
- **Culture can't be changed** (“we must change the culture”)
- **Culture can change**
- **By doing things differently**

It can't be done, *they* don't allow it

- **If the success of your project is being frustrated by**
 - dogmatic rules
 - amateur managers

it's no excuse for failure of your project

- If you don't really get the responsibility (empowerment)
 - If you cannot continue to take responsibility
- **Return the responsibility**
- **At the end of your project it's too late**
at the FatalDate any excuse is irrelevant
- **You knew much earlier**



People oppose change !



- People are not against change
- People (sub-consciously) don't like **uncertainty**
- Any project changes something and thus introduces uncertainty
- People can cope with uncertainty for a short time

Excuses, excuses, excuses ...

- We have been thoroughly trained to make excuses
- We always downplay our failures

- At the Fatal Day, any excuse is in vain: we failed
- Even if we “couldn’t do anything about it”
- Failure is a very hard word. That’s why we are using it !
- No pain, no gain
- We never say: “You failed”, better: “We failed”
 - After all, we didn’t help the person not to fail
 - “Lose face” is not only typical Asian

Ignore the first reaction

- **If you show something is wrong**
- **Even if the person agrees, first you'll get:**

**“Yes, but ... bla bla” or,
“That’s because ... bla bla”**

- **We have been trained from childhood to make excuses**
- **Ignore the bla bla**
- **Wait for the next reaction**

Logical thinking is not always better

- **Intuitive decision is often good**
- **Logical thinking feeds the sub-consciousness**
- **Sub-consciousness needs some time**

Real Options

- Option to make or abandon a decision
- The later you make the decision, the more information you can have about it
- Options have value until expiration
- On expiration the value has disappeared
- Just in Time delivery
- Start feeding your sub-consciousness in due time, to decide just in time

Accept Human Psychology

- **Why don't they practice what we preach?**
(Humphrey 1999)
- **They don't practice what we preach !**

What now?

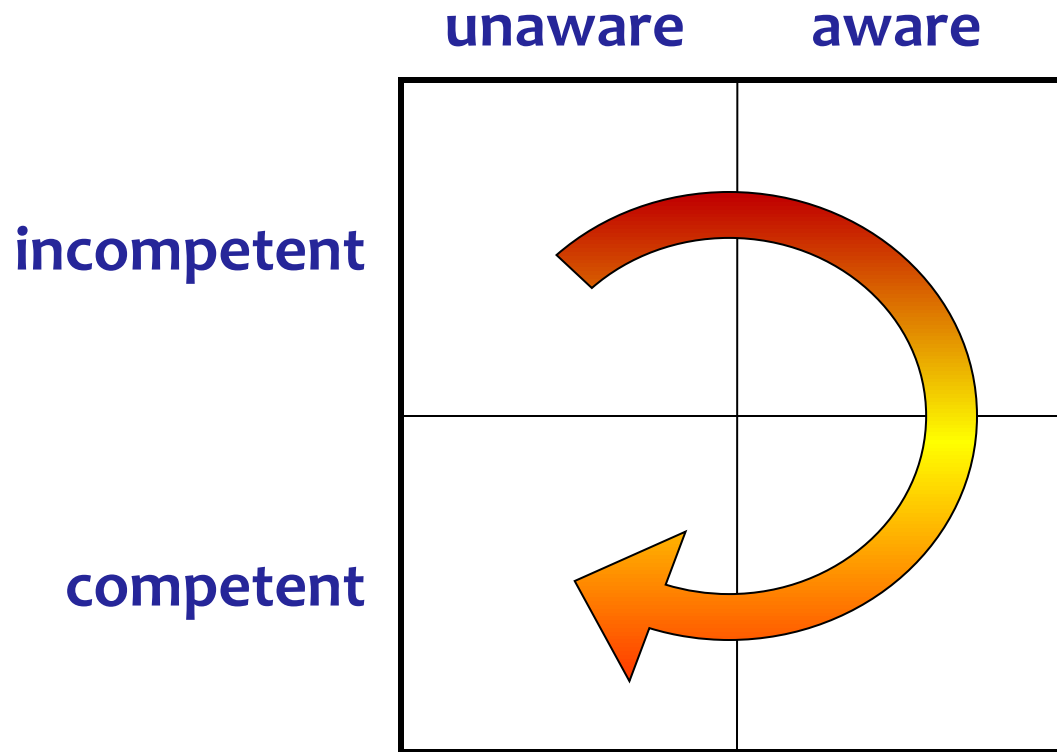
Ready in January

Sense of Urgency

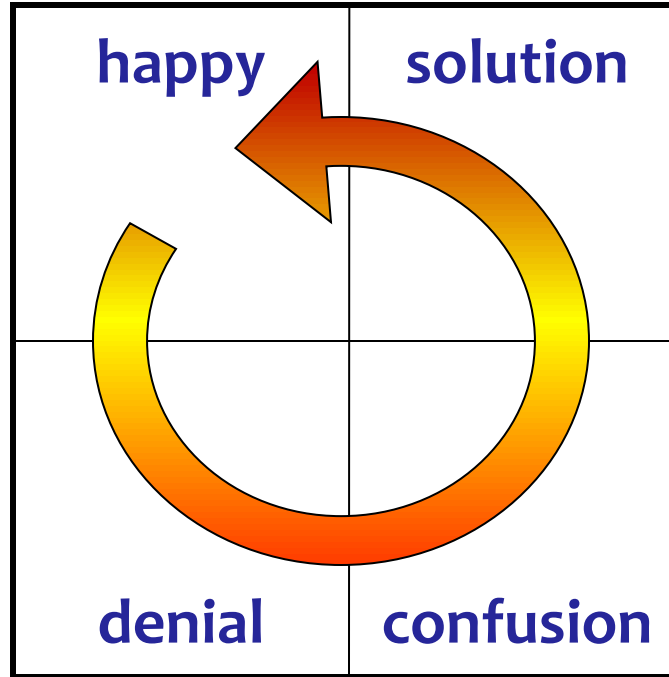
- **Stick to your agreement**
 - Can you do that?
 - Yes
 - When is it done?

Be as explicit as needed

Competence square



The problem of problem denial



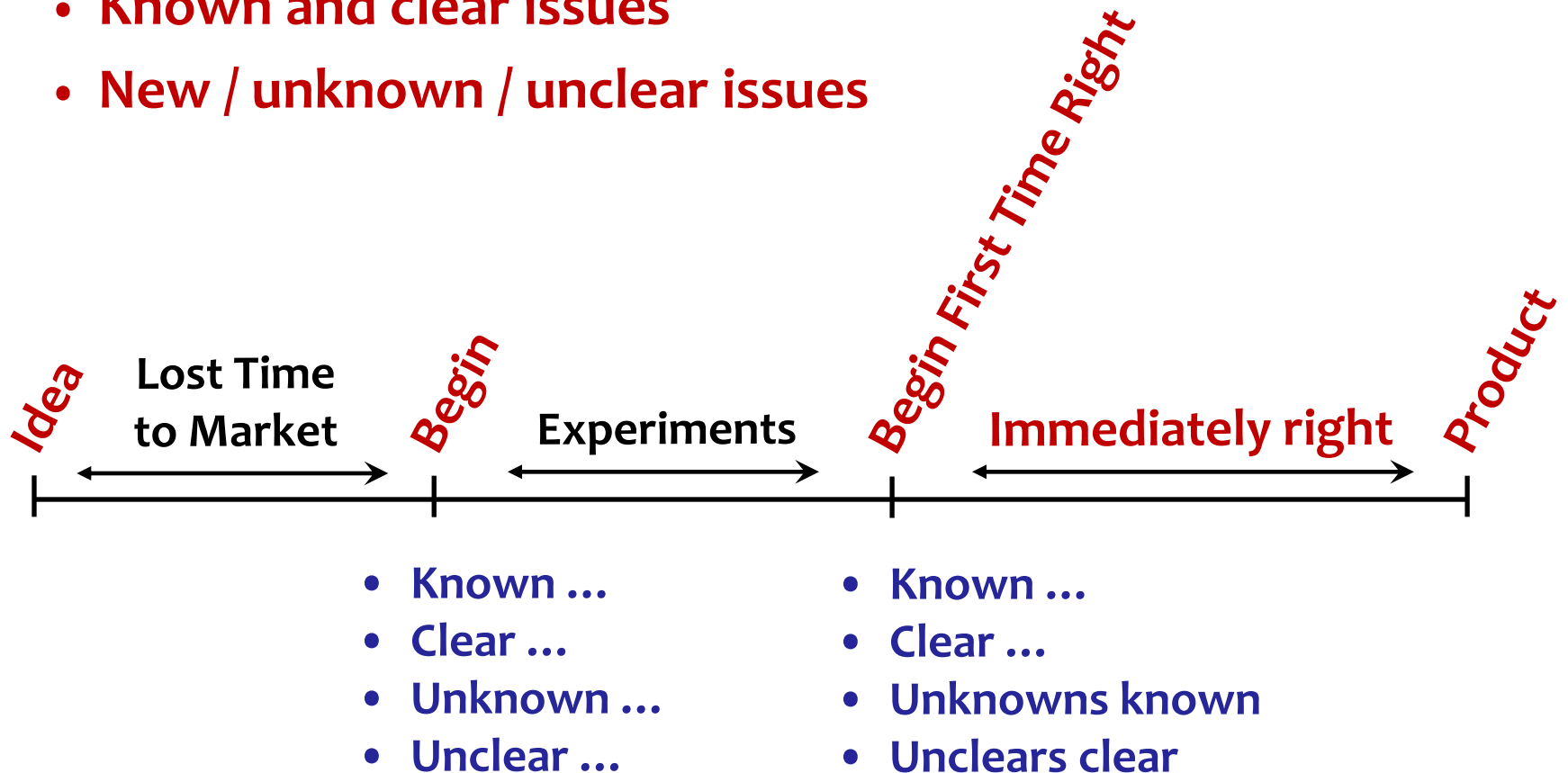
What do we do first ?

- **Known and clear issues**
- **New / unknown / unclear issues**



First Time Right

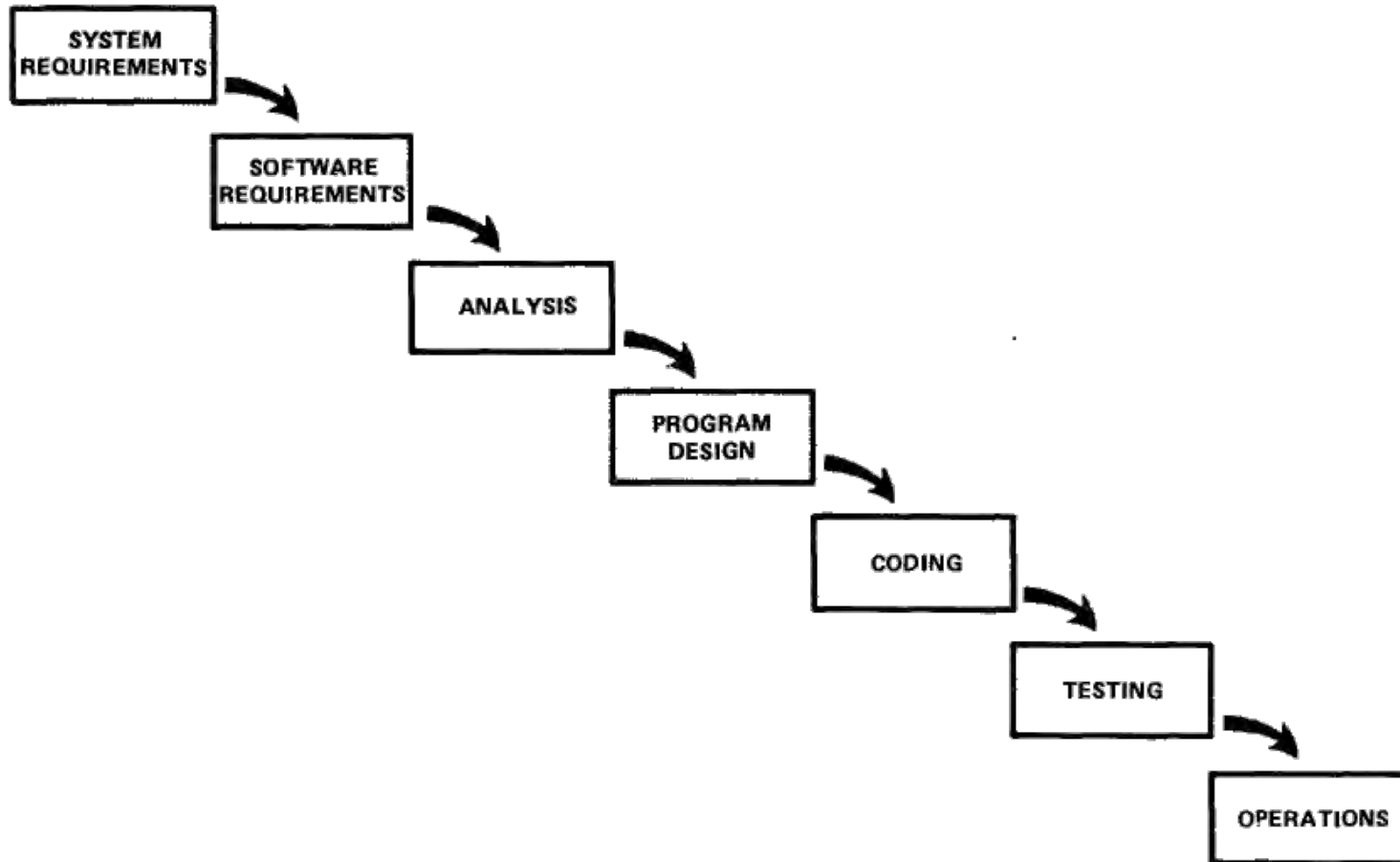
- Known and clear issues
- New / unknown / unclear issues



Project Life Cycles

Waterfall ?

Winston Royce 1970



This is what Royce wanted to say

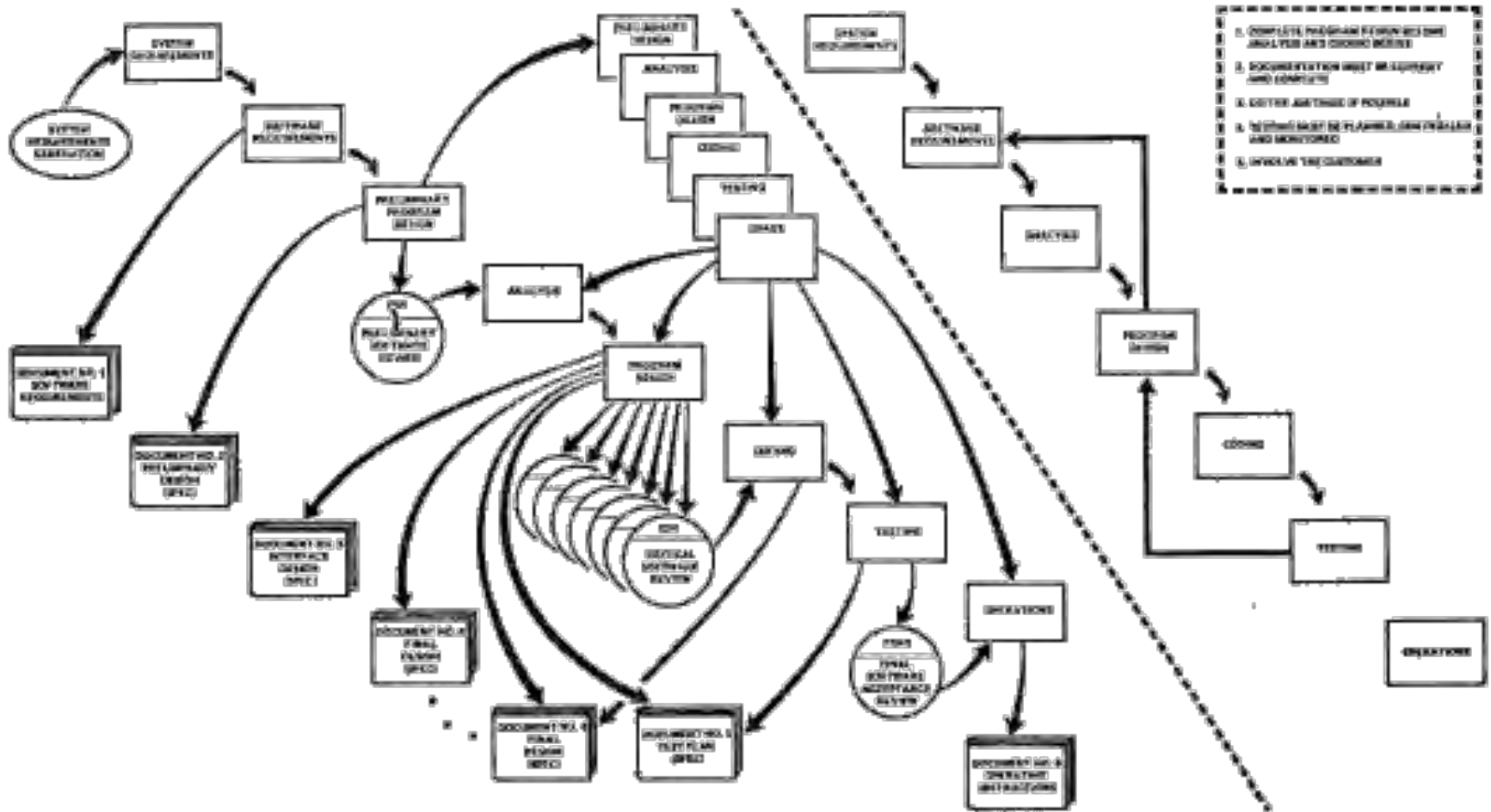


Figure 10. Summary

When can we use waterfall ?

- Requirements are completely clear, nothing will change
- We've done it many times before
- Everybody knows exactly what to do
- We call this *production*

- In your projects:
 - Is everything completely clear ?
 - Will nothing change ?
 - Does everybody know exactly what to do ?
 - Are you sure ?
- Even most production doesn't run smoothly the first time

How management likes it



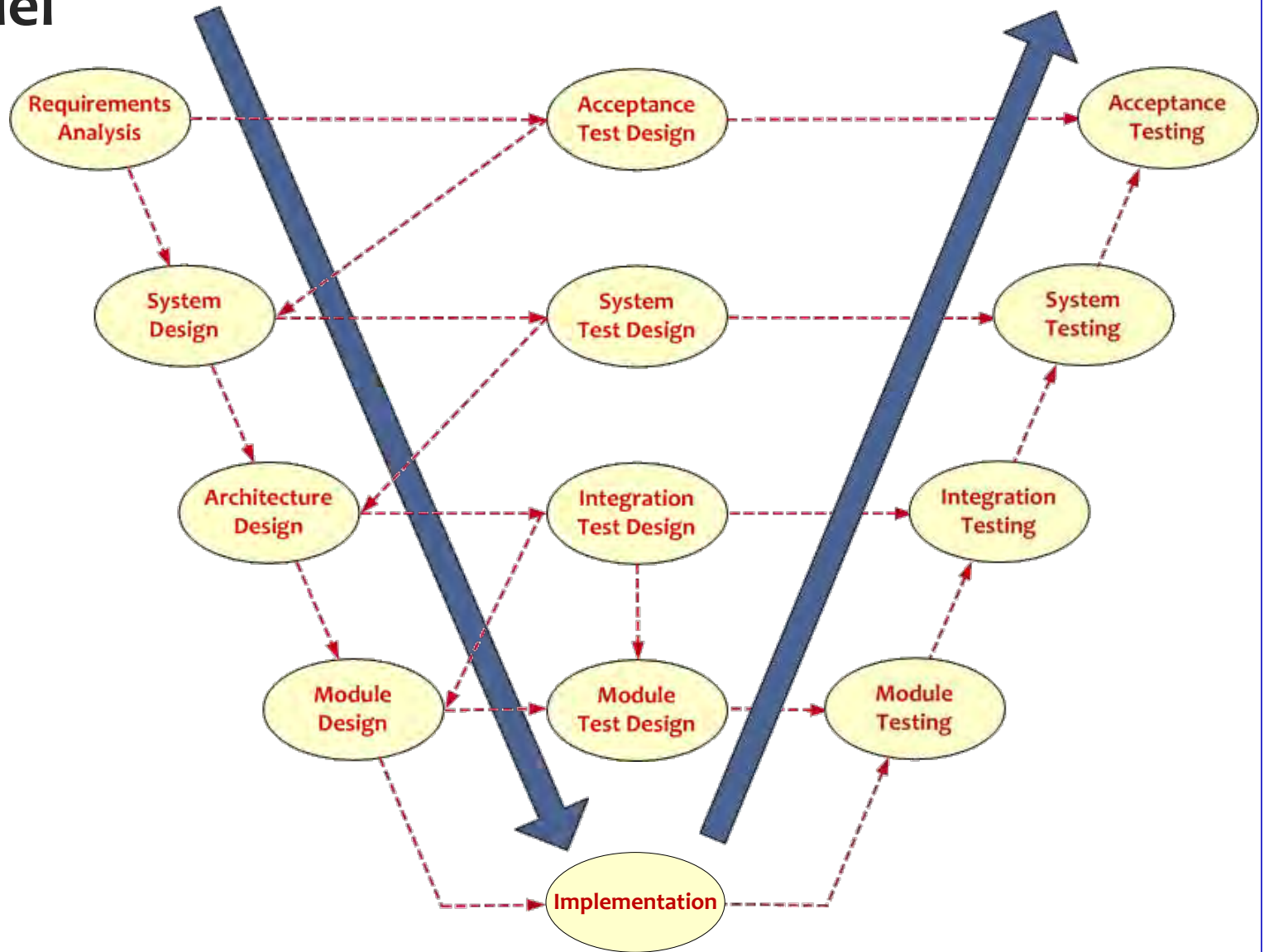
Start
Project

We can
do it

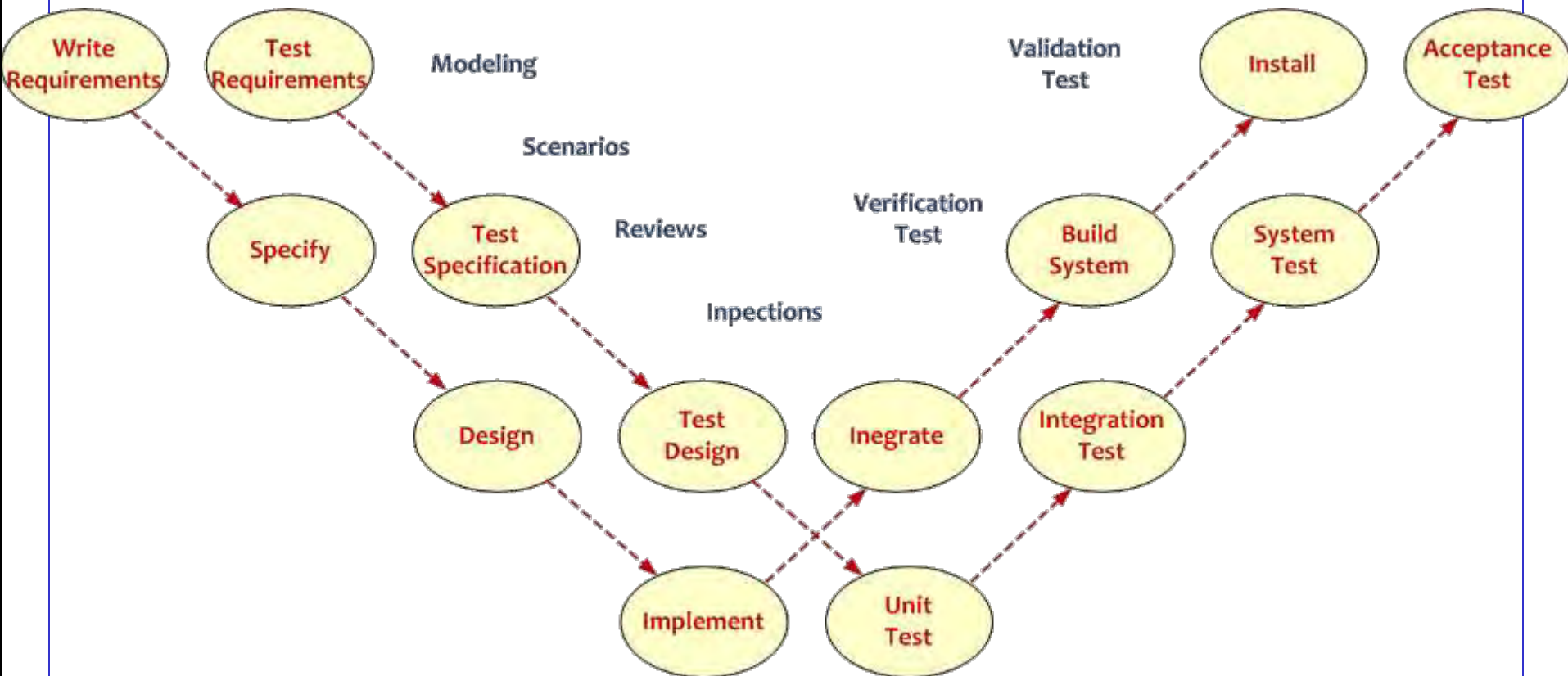


We did it

V-Model



W-model



All Models are wrong

Some are useful

Evolutionary Principles

No cure - no pay

- **If what we do doesn't deliver a positive ROI, there is no money to pay our salary**
- **So, better do not do things that do not deliver ROI**
- **Do you dare to work on a no-cure-no-pay basis ?**

Value

- **Value is what makes the customer more successful and happy than before**
- **We're in the game of**
 - **Optimizing Value**
 - **Eliminating Non-Value**
 - **Not causing problems**
 - **At the lowest cost**

Perceived value

- **What we perceive as value**
- **What the users perceive as value**
- **What the customer perceives as value**
- **What the stakeholders perceive as value**

- **May be different from real value**
- **Better assume that a lot of our assumptions are wrong**

- **Still, value means different things to different stakeholders**
- **If we want to be successful, we may have to find the best compromise**

The head and the hart

- **There is often a paradox between what the mind tells and what the body does** (logic ↔ emotion)
- **So, we shouldn't just do what the customer says, but rather find out what he really needs** and still wants
- **If we base our perception of the requirements on what the customer says** (Waterfall, Agile), **we're probably developing a great solution to the wrong problem**

Murphy's Law

- **Whatever can go wrong, will go wrong**
- **Should we accept fate ??**

Murphy's Law for Professionals:

Whatever can go wrong, will go wrong ...

Therefore:

**We should actively check all possibilities that can go wrong
and *make sure that they cannot happen***

Preflection, foresight, prevention

Insanity is doing the same things over and over again and hoping the outcome to be different (*let alone better*)

Albert Einstein 1879-1955, Benjamin Franklin 1706-1790, it seems Franklin was first

Only if we change our way of working, the result may be different

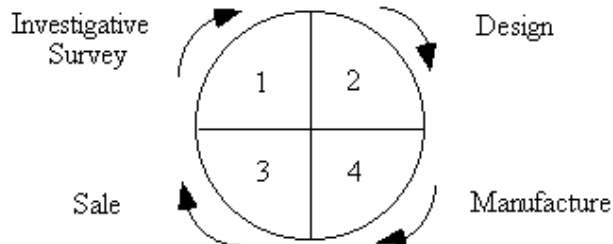
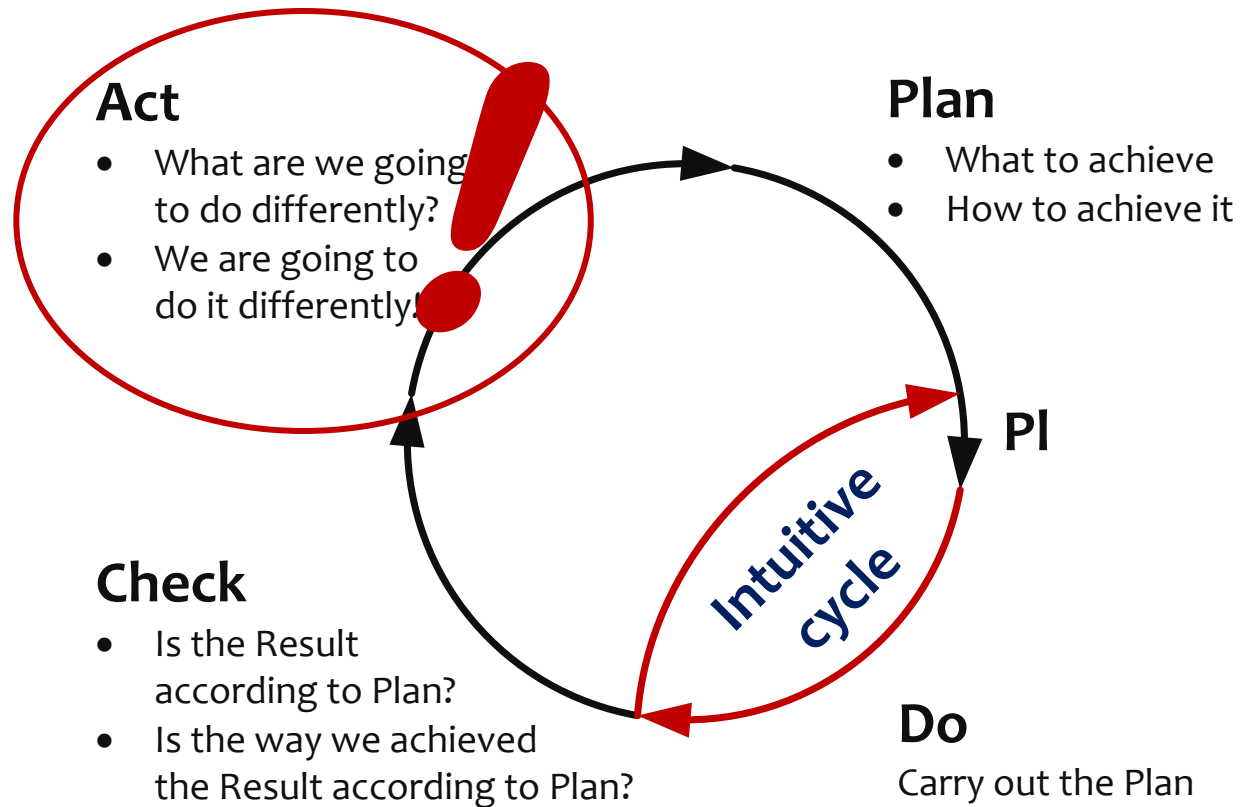
- **Hindsight is easy, but reactive**
- **Foresight is less easy, but proactive**
- **Reflection is for hindsight and learning**
- **Preflection is for foresight and prevention**

Only with *prevention* we can save precious time

This is used in the Deming or *Plan-Do-Check-Act* cycle

The essential ingredient: the PDCA Cycle

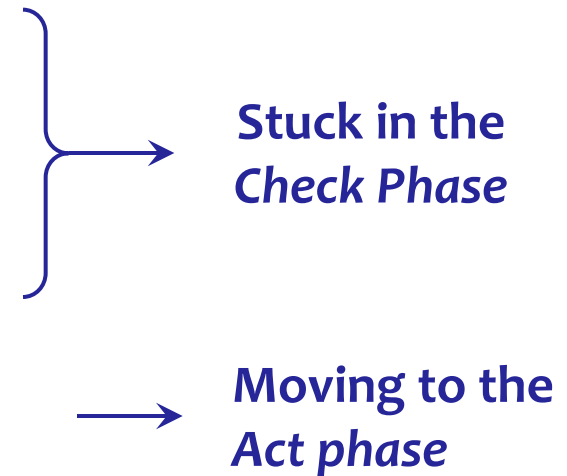
(Shewhart Cycle - Deming Cycle - Plan-Do-Study-Act Cycle - Kaizen)



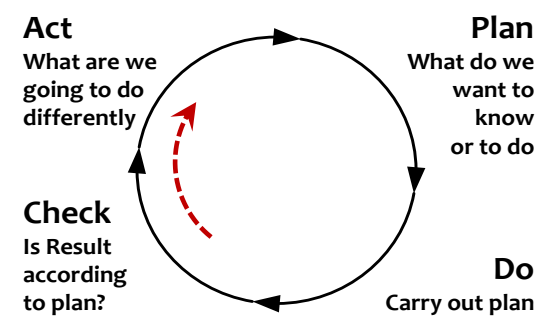


It can't be done - it must be done

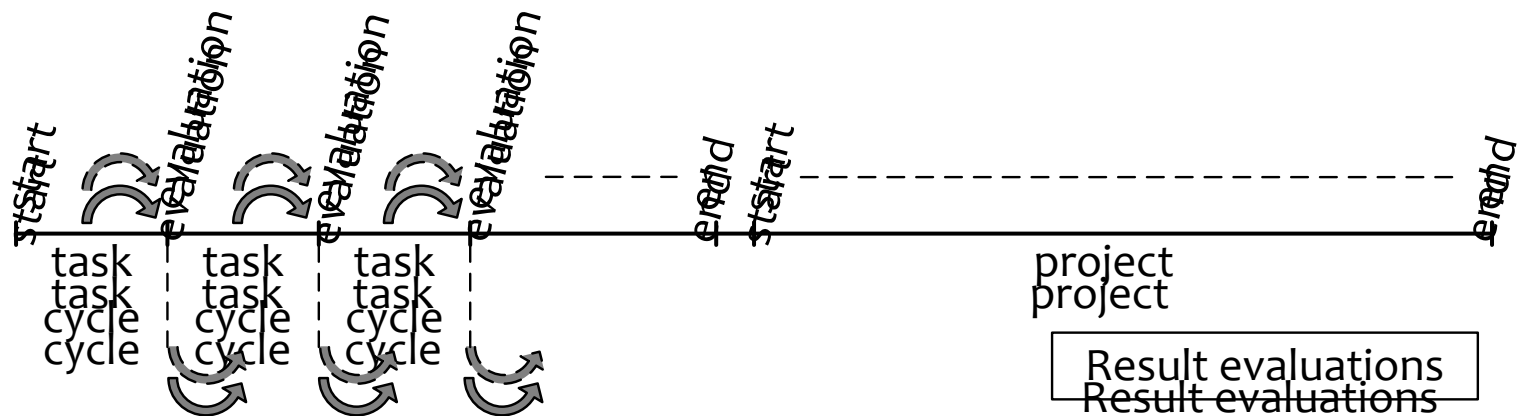
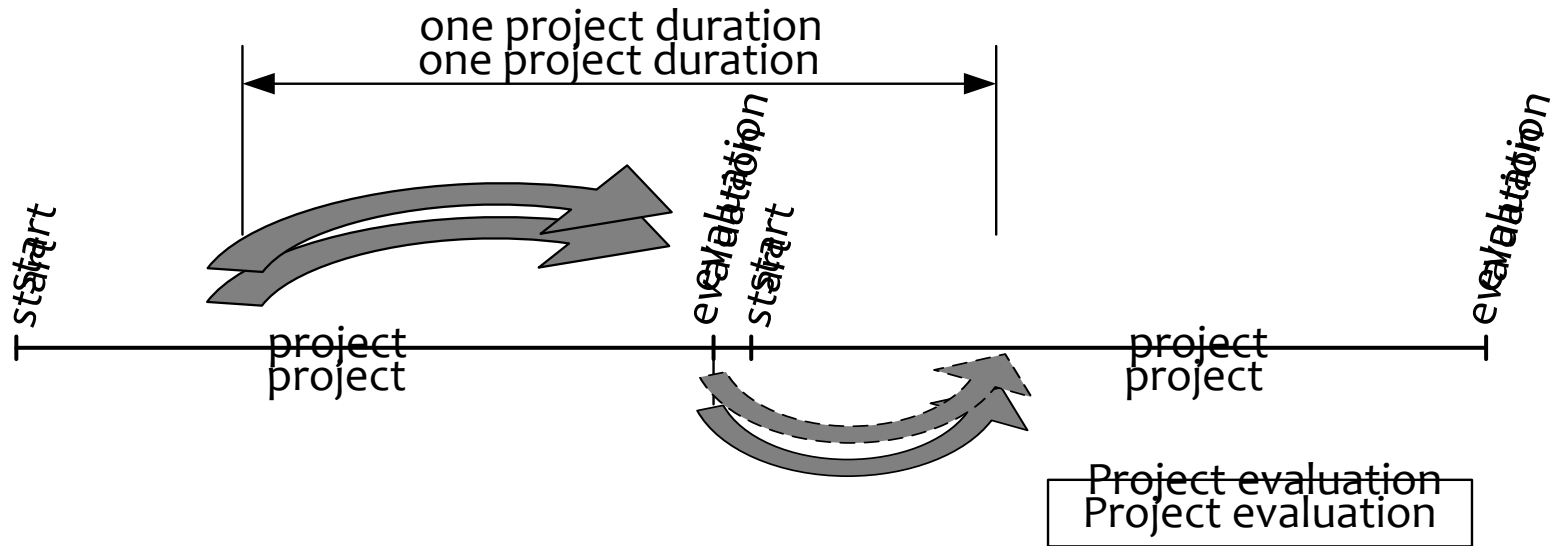
- **It can't be done**
 - Management doesn't allow it
 - "They" won't do it
 - It's impossible
- **It must be done**
 - How are we going to do it



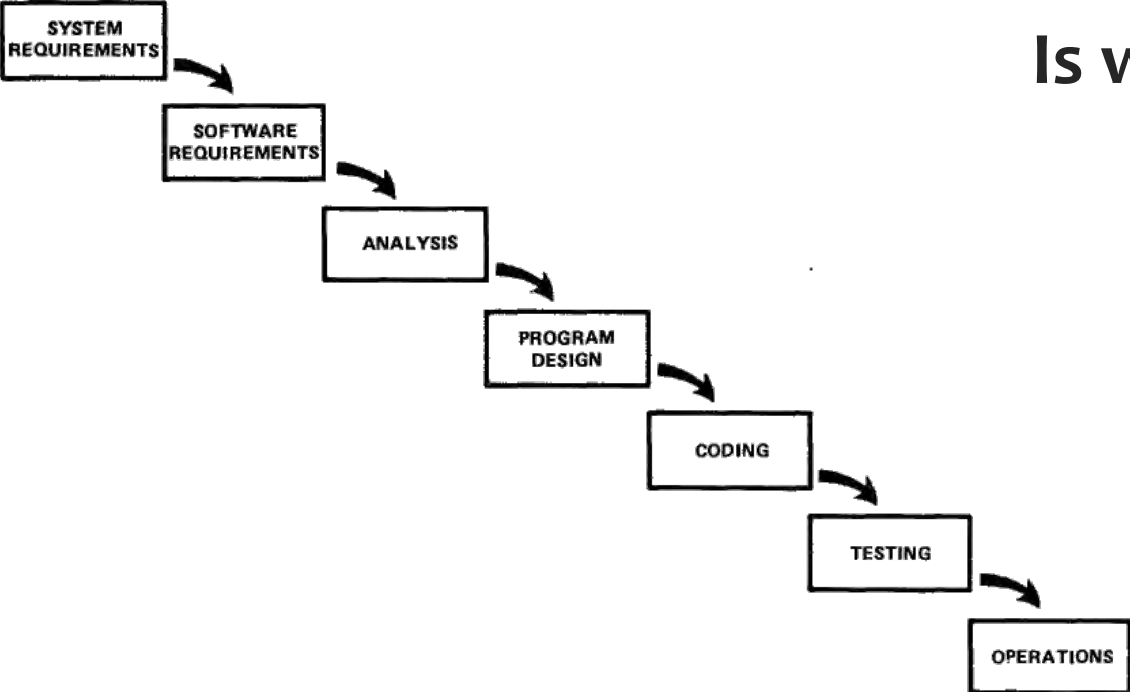
- **We don't let others make us fail**



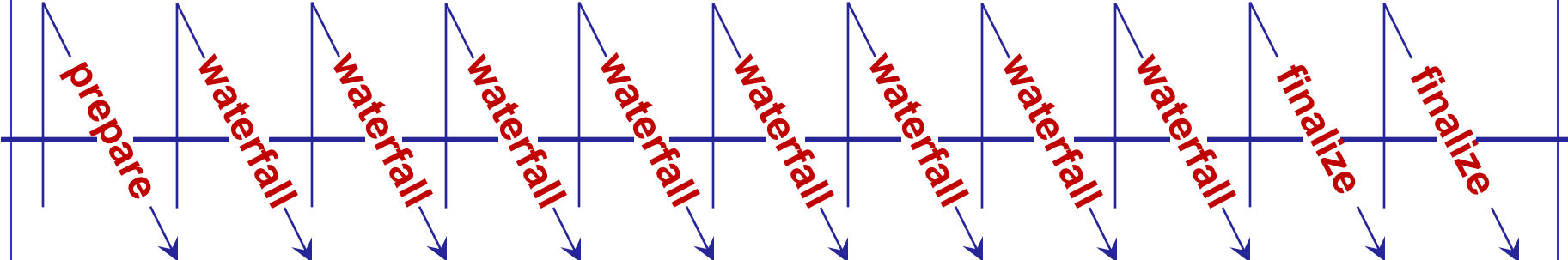
Project evaluations



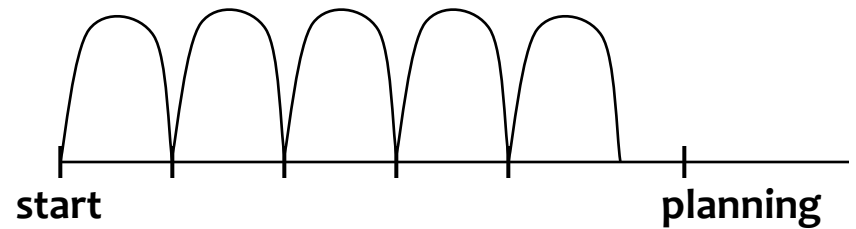
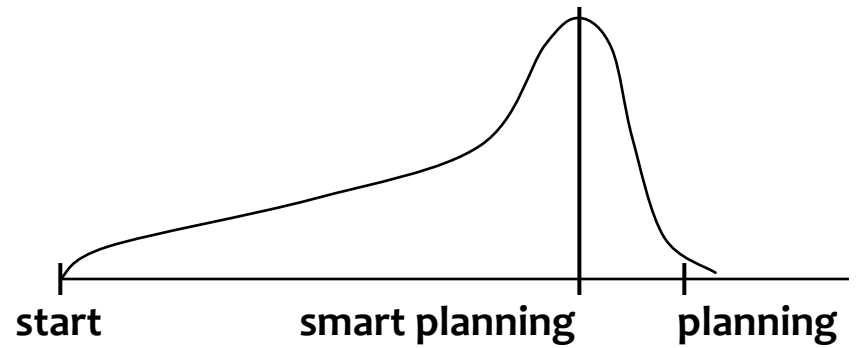
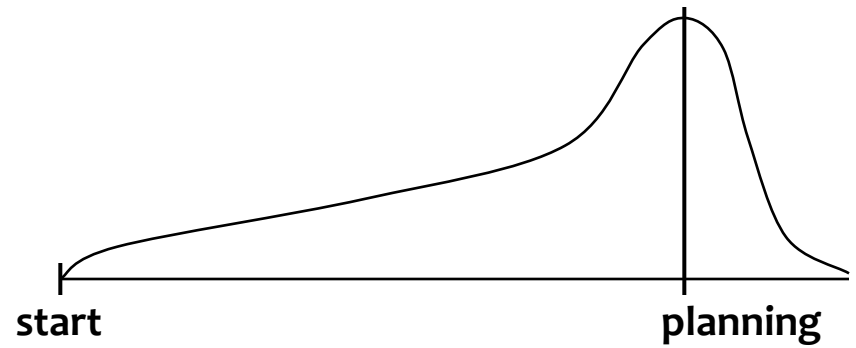
Is waterfall wrong ?



cycle 1 2 3 4 5 n-1 n



Development cycles



Knowledge how to achieve the goal

If we

- Use very short Plan-Do-Check-Act cycles
- Constantly selecting the most important things to do

then we can

- Most quickly learn what the real requirements are
- Learn how to most effectively and efficiently realize these requirements

and we can

- Spot problems quicker, allowing more time to do something about them

Act

- What are we going to do differently?
- We are going to do it differently!

Plan

- What to achieve
- How to achieve it

Check

- Is the Result according to Plan?
- Is the way we achieved the Result according to Plan?

Do

Carry out the Plan



doing the
right things



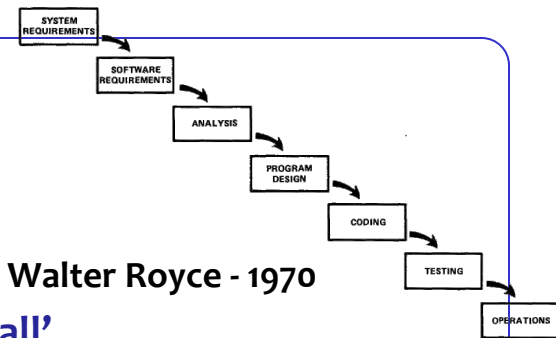
doing the
right things
right

Known for decades

- **Benjamin Franklin** (1706-1790)
 - Waste nothing, cut off all unnecessary activities, plan before doing, be proactive, assess results and learn continuously to improve
- **Henry Ford** (1863-1947)
 - My Life and Work (1922)
 - We have eliminated a great number of wastes
 - Today and Tomorrow (1926)
 - Learning from waste, keeping things clean and safe, better treated people produce more
- **Toyoda's (Sakichi, Kiichiro, Eiji)** (1867-1930, 1894-1952, 1913-)
 - Jidoka: Zero-Defects, stop the production line (1926)
 - Just-in-time – flow – pull
- **W. Edwards Deming** (1900-1993)
 - Shewart cycle: Design-Produce-Sell-Study-Redesign (Japan – 1950)
 - Becoming totally focused on quality improvement (Japan – 1950)
Management to take personal responsibility for quality of the product
 - Out of the Crisis (1986) - Reduce waste
- **Joseph M. Juran** (1904-2008)
 - Quality Control Handbook (1951, Japan – 1954)
 - Total Quality Management – TQM
 - Pareto Principe
- **Philip Crosby** (1926-2001)
 - Quality is Free (1980)
 - Zero-defects (1961)
- **Taiichi Ohno** (1912-1990)
 - (Implemented the) Toyota Production System (Beyond Lange-Scale Production) (1988)
 - Absolute elimination of waste - Optimizing the TimeLine from order to cash
- **Masaaki Imai** (1930-)
 - Kaizen: The Key to Japan's Competitive Success (1986)
 - Gemba Kaizen: A Commonsense, Low-Cost Approach to Management (1997)

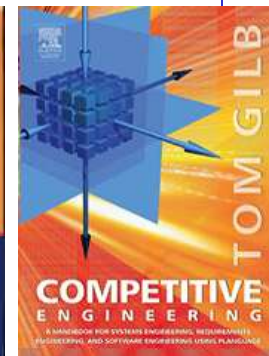
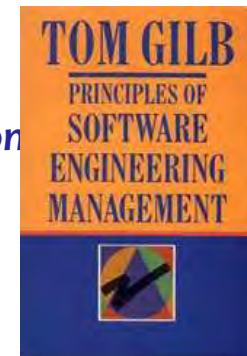
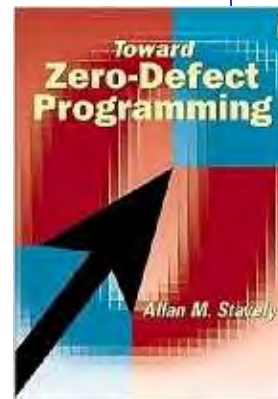
Eliminating Waste
Not doing what
doesn't yield value





There is nothing new in software too

- **Managing the development of large software systems** - Walter Royce - 1970
 - Famous “Waterfall document”: figure 2 showed a ‘waterfall’
 - Text and other figures showed that Waterfall doesn’t work
 - Anyone promoting Waterfall doesn’t know or didn’t learn from history
- **Incremental development** - Harlan Mills - 1971
 - Continual Quality feedback by Statistical Process Control (Deming !)
 - Continual feedback by customer use
 - Accommodation of change - Always a working system
- **Cleanroom software engineering** - Harlan Mills - 1970’s
 - Incremental Development - Short Iterations
 - Defect prevention rather than defect removal
 - Statistical testing
 - 10-times less defects at lower cost
 - Quality is cheaper
- **Evolutionary Delivery - Evo** - Tom Gilb - 1974, 1976, 1988, 2005
 - Incremental + Iterative + *Learning and consequent adaptation*
 - Fast and Frequent Plan-Do-Check-Act
 - Quantifying Requirements - Real Requirements
 - Defect prevention rather than defect removal



If we know, why do projects still fail ?

Cobb's Paradox:

**"We know why projects fail,
we know how to prevent their failure
-- so why do they still fail?"**

**Martin Cobb
Treasury Board of Canada Secretariat
Ottawa, Canada**

1989

Evo

Act

- What are we going to do differently?
- We are going to do it differently!

Plan

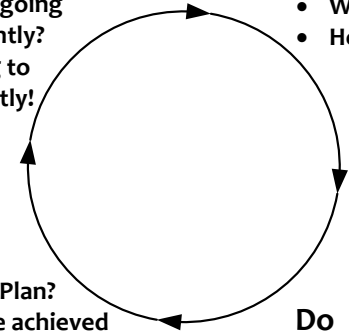
- What to achieve
- How to achieve it

Check

- Is the Result according to Plan?
- Is the way we achieved the Result according to Plan?

Do

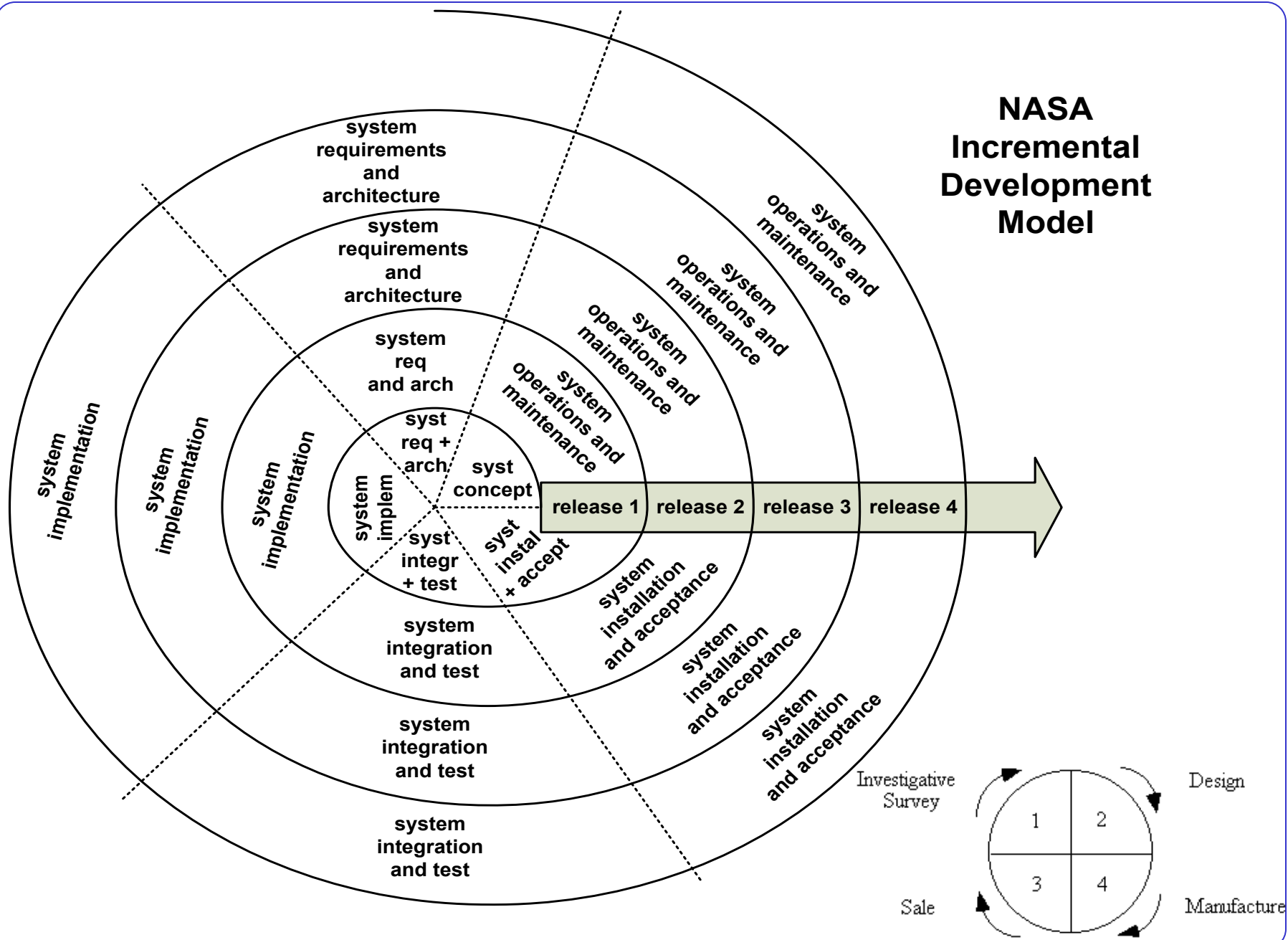
Carry out the Plan



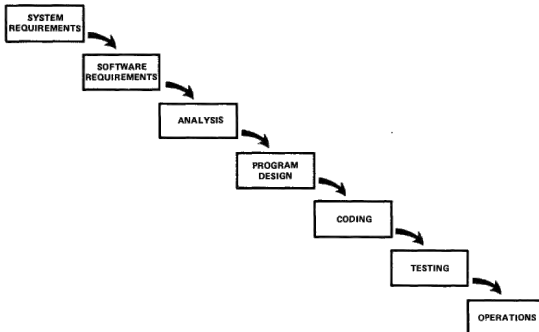
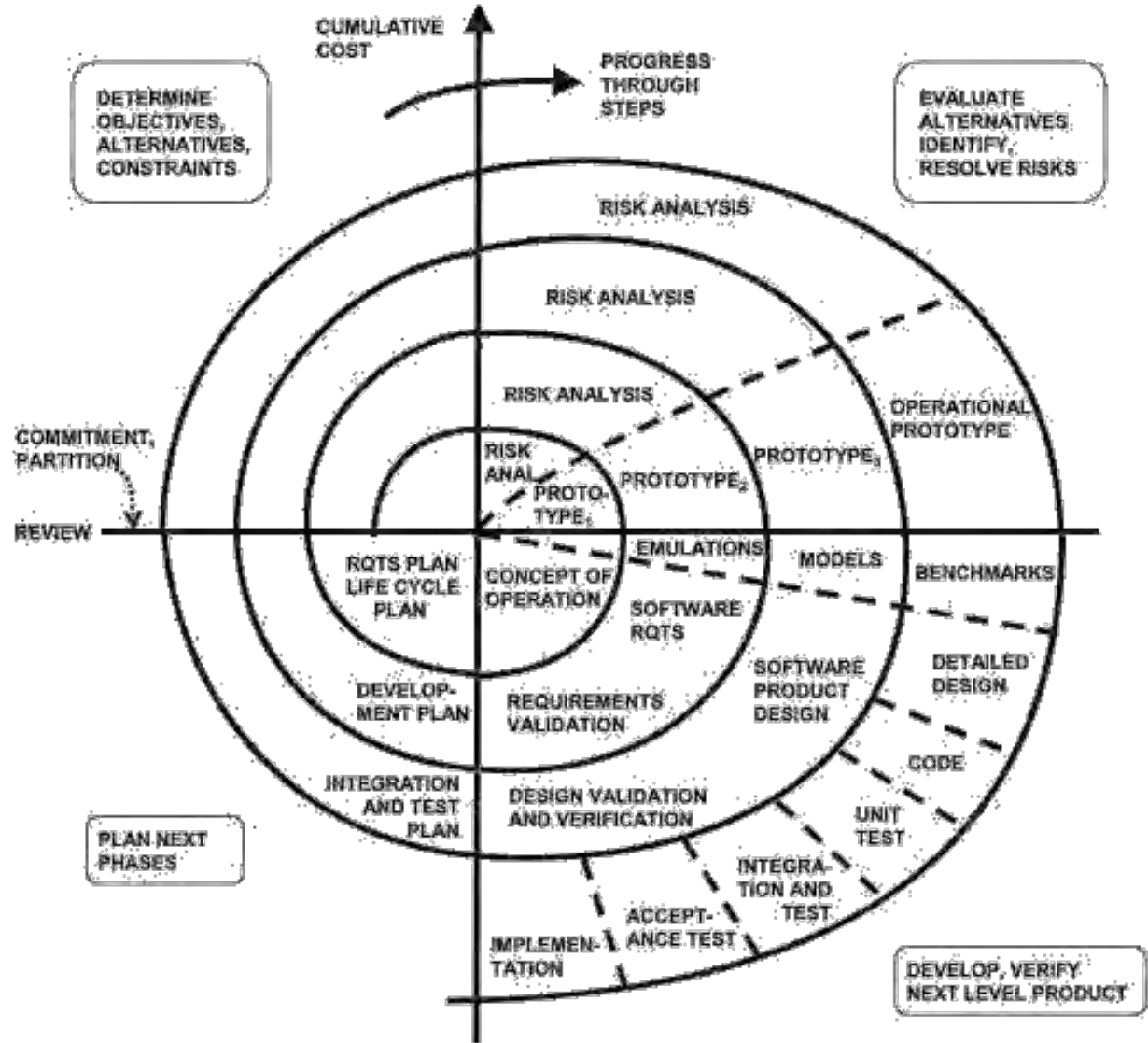
- **Evo (short for Evolutionary...) uses PDCA consistently**
- **Applying the PDCA-cycle actively, deliberately, rapidly and frequently, for *Product*, *Project* and *Process*, based on ROI and highest value**
- **Combining Planning, Requirements- and Risk-Management into *Result Management***
- **We know we are not perfect, but the customer shouldn't be affected**
- **Evo is about *delivering* Real Stuff to Real Stakeholders doing Real Things**
- **Projects seriously applying Evo, routinely conclude successfully on time, or earlier**

“Nothing beats the Real Thing”

NASA Incremental Development Model



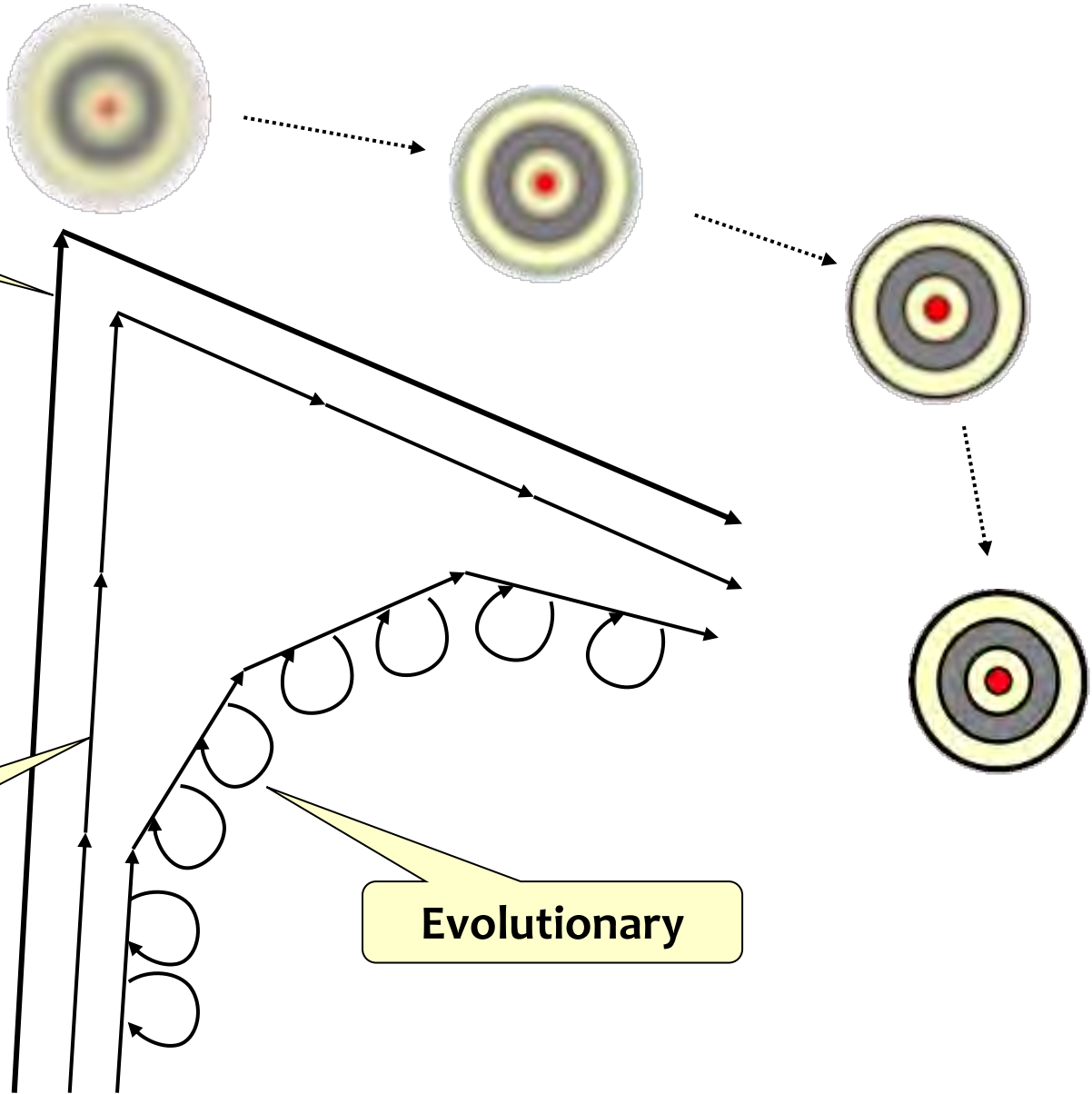
Spiral Process model (Boehm 88)



**Waterfall,
Big-Bang**

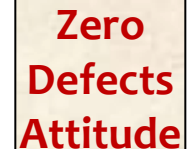
Incremental

Evolutionary



Evolutionary Project Management (Evo)

- **Plan-Do-Check-Act**
 - The powerful ingredient for success
- **Business Case**
 - *Why* we are going to improve *what*
- **Requirements Engineering**
 - *What* we are going to improve *and what not*
 - *How much* we will improve: quantification
- **Architecture and Design**
 - Selecting the optimum compromise for the conflicting requirements
- **Early Review & Inspection**
 - Measuring quality while doing, learning to prevent doing the wrong things



Zero
Defects
Attitude

Evo Project Planning

- **Weekly TaskCycle**
 - Short term planning
 - Optimizing estimation
 - Promising what we can achieve
 - Living up to our promises
- **Bi-weekly DeliveryCycle**
 - Optimizing the requirements and checking the assumptions
 - Soliciting feedback by delivering Real Results to *eagerly waiting* Stakeholders
- **TimeLine**
 - Getting and keeping control of Time: Predicting the future
 - Feeding program/portfolio/resource management

Evolutionary Planning

TaskCycle
DeliveryCycle

To-do lists

- **Are you using to-do lists?**

→ **EXERCISE**

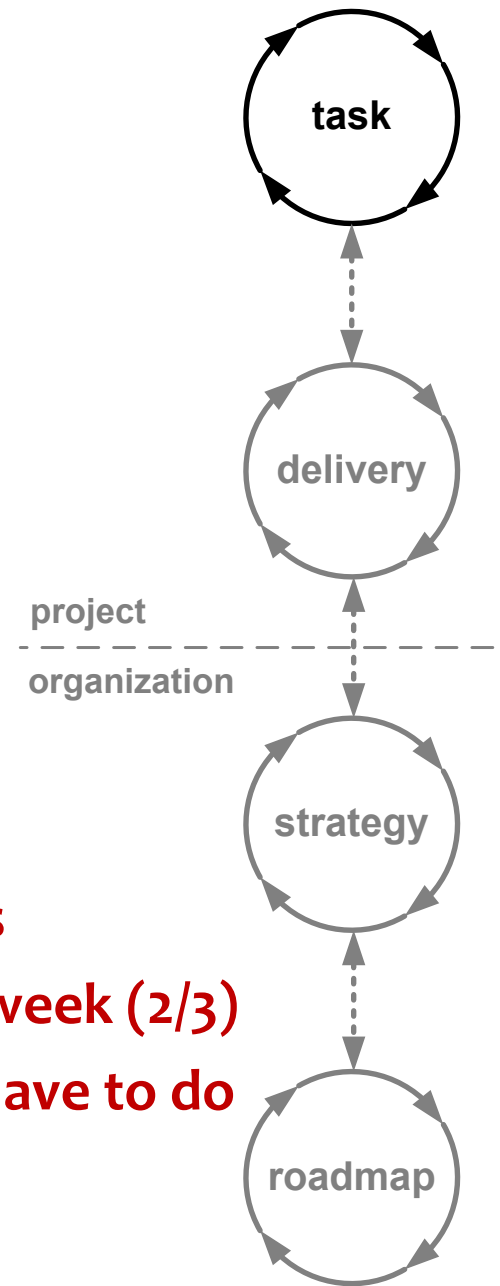
- Did you add effort estimates?
- Does what you have to do fit in the available time ?
- Did you check what you can do and what you cannot do?
- Did you take the consequence?

- **Evo:**

- Because we are short of time, we better use the limited available time as best as possible
- We don't try to do better than *possible*
- To make sure we do the best possible, we *choose* what to do in the limited available time. We don't just let it happen randomly

Evo Planning: Weekly TaskCycle

- Are we **doing** the right things, in the right order, to the right level of detail for now
- Optimizing estimation, planning and tracking abilities to better predict the future
- Select highest priority tasks, never do any lower priority tasks, never do undefined tasks
- There are only about 26 plannable hours in a week (2/3)
- In the remaining time: do whatever else you have to do
- Tasks are always done, 100% done



Effort and Lead Time

- **Days estimation → lead time (calendar time)**
- **Hours estimation → effort**

- **Effort variations and lead time variations have different causes**
- **Treat them differently and keep them separate**
 - **Effort: complexity**
 - **Lead Time: time-management**
 - **(effort / lead-time ratio)**

Every week we plan

- How much time do we have available
- $\frac{2}{3}$ of available time is net plannable time
- What is most important to do
- Estimate effort needed to do these things
- Which most important things fit in the net available time (default 26 hr per week)
- What can, and are we going to do
- What are we **not** going to do

$\frac{2}{3}$ is default start value
this value works well in development projects

Task _a	2	↑	
Task _b	5		
Task _c	3		
Task _d	6		do
Task _e	1		
Task _f	4		
Task _g	5		26
<hr/>			
Task _h	4	↓	
Task _j	3		do
Task _k	1		not

Making best use of limited available time

- **If the work is done, the time is already spent**
- **If we still have to do the work, we can decide**
 - What is really important
 - What is less important
 - What we must do
 - What we can do
 - What we are going to do
 - What we are not going to do
- **Therefore we plan first, in stead of finding out later**
- **We cannot work in history**

Estimation

- **Changing from Optimistic to Realistic**
- **Only works if we are Serious about Time**

Sense of Urgency

At the end of the week

**Immediate
consumption
of metrics**

- **Was all planned work really done?**

If a Task was not completed, we have to learn:

- **Time spent but the work not done? → effort estimation problem**
Discuss what the causes may be and decide how to change your estimation habits
- **Time not spent? → time management problem**
 - Too much distraction
 - Too much time spent on other (poorly-estimated) Tasks
 - Too much time spent on unplanned Tasks

Discuss what the causes may be and decide how to improve (Check and Act)

- **Conclude unfinished Tasks after *having dealt with the consequences* ← immediate metrics consumption !**
 - Feed the disappointment of the “failure” into your intuition mechanism
 - Define new Tasks, with estimates, and put on the Candidate Task List
 - Declare the Task finished after having taken the consequences
- **Continue with planning the Tasks for the next week**

Meetings

- **Do you have weekly project meetings ?**
- **Pitfalls**
 - Not reaching set goals
 - One to ones, others waiting
 - Example: status round (“round of excuses”)
 - Example: detailed discussion
 - Discussing less important subjects for too long
- **Meetings are very costly** (ROI?)
 - Try the meeting-meter
number of people * average hourly rate: show \$\$ ticking

Weekly 3-Step Procedure

- **Individual preparation**
 - Conclude current tasks
 - What to do next
 - Estimations
 - How much time available
- **Modulation with / coaching by Project Management**
 - Status
 - Priority check
 - Feasibility
 - Commitment and decision
- **Synchronization with group (team meeting)**
 - Formal confirmation
 - Concurrency
 - Learning
 - Helping
 - Socializing

Today
6 mei 2004 wk 19

Project
Dino-QUA

Delivery
4

Other work

TaskCycle
Future

TaskType
Code

Priority
0

Who
-

hrs
hr (=Timebox!)

Plan Reviewer
-

done (Checks)
 100% done

Hours of	0	total
-	0	OK
in Cycle	0	not OK
Fut	0	

TaskSheet Results Checks Project and Delivery Tasks Cycle and Delivery Timing Printing Edit/New

Task Name
Hoe gaan we exporteren doen

Cycle
-

Other work

Task cycle due date
-

Delivery Nr
4

Delivery Name
Delivery 4

Delivery Due
21 mei 2004 wk 21

The TaskSheet is used to focus on what the task really is about.

Task Description

Validation (how to check that the requirements are met)

Functional Requirements (what the result of this task should be)

Implementation Ideas (solution direction ideas)

Performance Requirements (how well the result should do the what)

Planning (to make sure task is done on time)

Constraints (what not)

Unclears (anything that is still unclear)

ID	Project	Delivery	Cycle	Task cycle due date	Pri	Who	hrs	Done	TaskName
59	Dino-QUA	Delivery 4	Fut		0	-			Hoe gaan we exporteren doen
58	Dino-QUA	Delivery 4	Fut		0	-			Hoe gaan we importeren doen?
212	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	18		Documentatie SPS, SCM-BDB
220	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	6		Samples importeren
211	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	4		Conversie aanpassen n.a.v. Hans van der Meij
214	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	4	Arian	10		Export blokken maken
215	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	2		Checkbox toevoegen voor export-blokken
216	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	2		Backsupport toevoegen met Ronald
217	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	2		Backsupport toevoegen met Arian
218	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	6		Uitzoeken rechts uitvullen van kolommen bij sample, subsample
219	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	6		Maken Process dialog
210	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	2		Conversie aanpassen voor Omrekenfactor koppeling
200	Dino-QUA	Delivery 4	12	4 jun 2003 wk 23	5	Niko	4	OK	parameterformulier voor analyserapport met tabbladen
201	Dino-QUA	Delivery 4	12	4 jun 2003 wk 23	5	Arian	3	OK	Aanpassingen Monsterscherm doorvoeren (nieuwe velden)
104	Dino-QUA	Delivery 5	13	4 jun 2003 wk 23	5	Niko	4	OK	Uitbreiden dataset delimit QUAS CALMITY en communicatie...

TaskSheet

(this is the think part of **First Think - Then Do**)

- **Task description**
- **Requirements for this task to be used as reference for verification:**
 - **Functions (what should be the result of this task?)**
 - **Qualities (how well should the results be)**
 - **Constraints (e.g. what not to do)**
- **What activities must be done to realize the requirements stated?**
- **Implementation details (how am I going to implement it)**
- **Verification approach - test design**
- **Planning**
- **Is everything really clear?**
- **Have this document (and related docs, if any) reviewed**
- **Clarify any unclears until everything is clear and agreed with the reviewer**
- **Do the work**

TaskSheet: More time needed?

- **Some people think they need more time for the Task if they “must fill in” the TaskSheet**
- **If you feel you “must fill in”: *Don't do it!***
- **If you think you need more time: add more time**
- **You will need this information during the Task anyway, so you should want it and**
- **It should *save* time**

Analysis Tasks

- **I don't know...**
 - That's an Analysis Task!
 - How much time are you going to give yourself?
- **To find out something we do not know**
 - Use short TimeBox
 - Documented at the end of the TimeBox:
 - What do we know now
 - What do we not yet know
 - What should we know more
 - Which New Tasks can we define?
 - Estimation and priority of these tasks defined
- **Typically Architecture and Design issues!**

Today

6 mei 2004 wk 19

Project

Dino-QUA

Delivery

4

 Other work

TaskCycle

12

TaskType

Code

Priority

5

Who

Arian

hrs

3 hr (=Timebox!)

Plan Reviewer

done (Checks)

OK 100% done

Hours of	26	total
Arian	26	OK
in Cycle	12	0 not OK

TaskSheet

Results

Checks

Project and Delivery

Tasks Cycle and Delivery

Timing

Printing

Edit/New

Done

Proposed New Tasks: with estimations!

Still to do

ID	Project	Delivery	Cycle	Task cycle due date	Pri	Who	hrs	Done	TaskName
59	Dino-QUA	Delivery 4	Fut		0	-			Hoe gaan we exporteren doen
58	Dino-QUA	Delivery 4	Fut		0	-			Hoe gaan we importeren doen?
212	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	18		Documentatie SPS, SCM-BDB
220	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	6		Samples importeren
211	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	4		Conversie aanpassen n.a.v. Hans van der Meij
214	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	4	Arian	10		Export blokken maken
215	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	2		Checkbox toevoegen voor export-blokken
216	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	2		Backsupport toevoegen met Ronald
217	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	2		Backsupport toevoegen met Arian
218	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	6		Uitzoeken rechts invullen van kolommen bij sample, subsample
219	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	6		Maken Process dialog
210	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	2		Conversie aanpassen voor Omrekenfactor koppeling
200	Dino-QUA	Delivery 4	12	4 jun 2003 wk 23	5	Niko	4	OK	parameterformulier voor analyserapport met tabbladen
201	Dino-QUA	Delivery 4	12	4 jun 2003 wk 23	5	Arian	3	OK	Aanpassingen Monsterscherm doorvoeren (nieuwe velden)
194	Dino-QUA	Delivery 5	13	4 jun 2003 wk 23	5	Niko	4	OK	Uitsiden detecteren Dino-QUA CALINITZ en communicatie...

Management Questions on Tasks

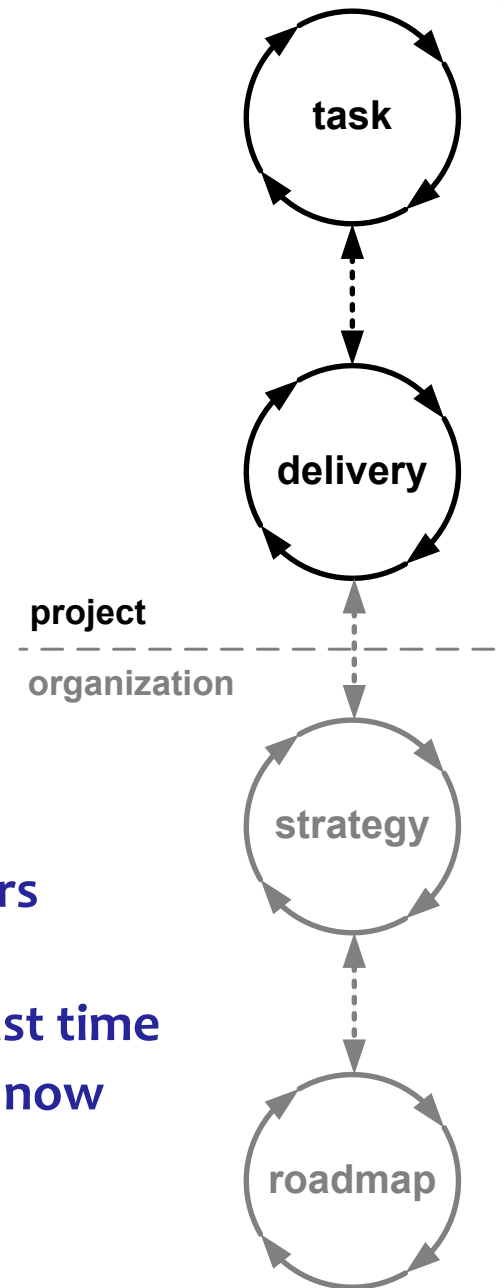
The screenshot displays the Evo Administrator software interface. The top window title is "Evo Administrator V1.12 - 18 Apr 2004." The main window is titled "TaskSheet" and contains a form for defining a task. The form includes fields for Project (Dino-QUA), Delivery (4), Task Cycle (Future), Task Type (Code), Priority (0), and Who. It also has sections for Functional Requirements, Performance Requirements, Constraints, Validation, Implementation Ideas, Planning, and Unclears. A table at the bottom lists tasks with columns for ID, Project, Delivery, Cycle, Task cycle due date, Pri, Who, hrs, Done, and TaskName.

ID	Project	Delivery	Cycle	Task cycle due date	Pri	Who	hrs	Done	TaskName
59	Dino-QUA	Delivery 4	Fut		0	-			Hoe gaan we exporteren doen
58	Dino-QUA	Delivery 4	Fut		0	-			Hoe gaan we importeren doen?
212	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	18		Documentatie SPS, SCM-BDB
220	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	6		Samples importeren
211	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	4		Conversie aanpassen n.a.v. Hans van der Meij
214	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	4	Arian	10		Export blokken maken
215	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	2		Checkbox toevoegen voor export-blokken
216	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	2		Backsupport toevoegen met Ronald
217	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	2		Backsupport toevoegen met Arian
218	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	6		Uitzoeken rechts uitvullen van kolommen bij sample, subsample
219	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Ronald	6		Maken Process dialog
210	Dino-QUA	Delivery 7	13	11 jun 2003 wk 24	5	Niko	2		Conversie aanpassen voor Omrekenfactor koppeling
200	Dino-QUA	Delivery 4	12	4 jun 2003 wk 23	5	Niko	4	OK	parameterformulier voor analyserapport met tabbladen
201	Dino-QUA	Delivery 4	12	4 jun 2003 wk 23	5	Arian	3	OK	Aanpassingen Monsterscherm doorvoeren (nieuwe velden)
104	Dino-QUA	Delivery 6	13	11 jun 2003 wk 24	5	Arian	4	OK	Uitvoeren datareductie QUA, CALIBTY en genereren testrapporten

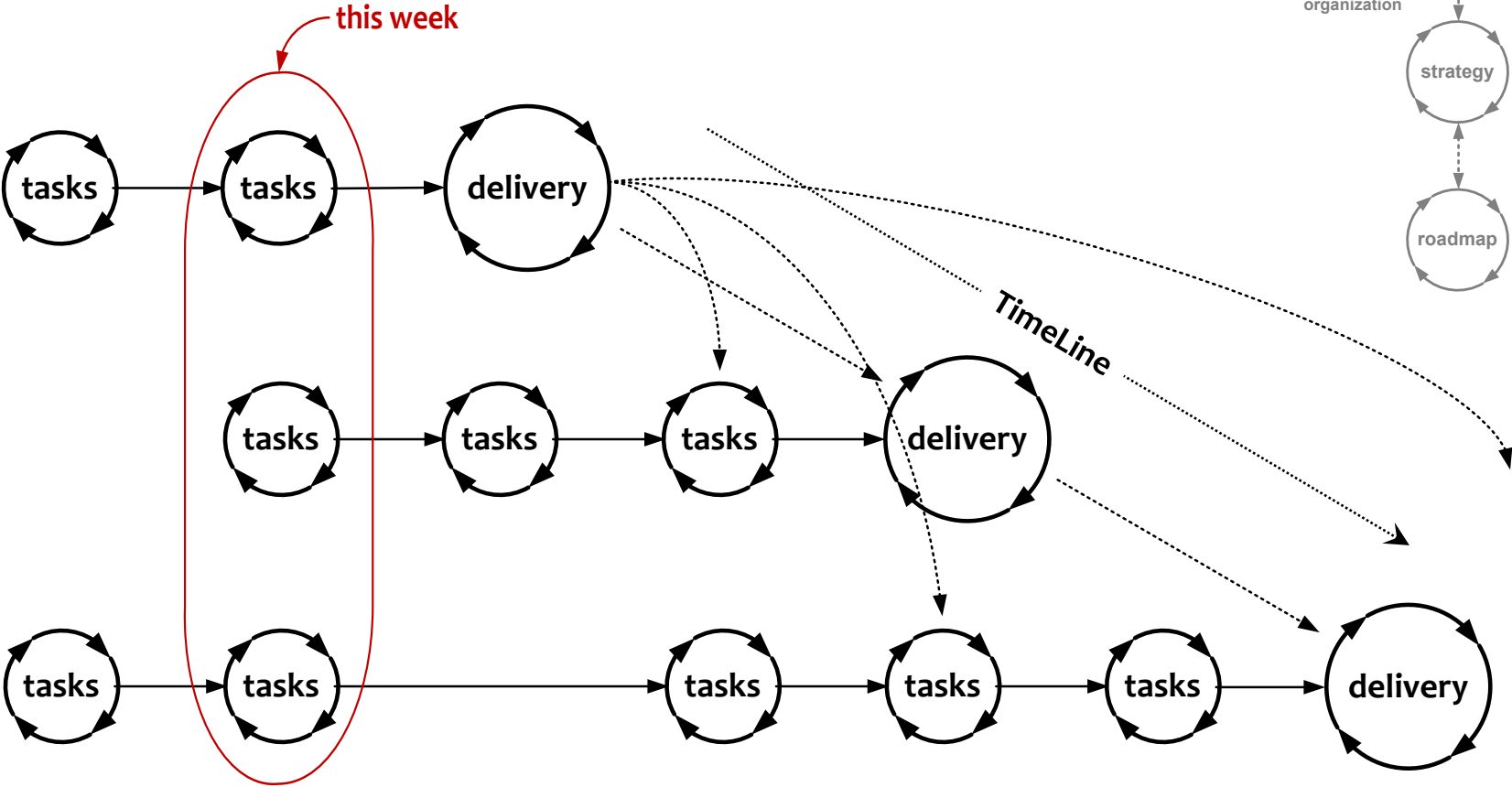
- Is the Project under Control ?
- Show me !
 - No “holes” in OK’s
 - All available plannable time planned
 - TaskSheets used
 - Results used
 - Prompt explanation in case of discrepancies

DeliveryCycle

- **Are we delivering the right things, in the right order to the right level of detail for now**
- **Optimizing requirements and checking assumptions**
 1. What will generate the optimum feedback
 2. We deliver only to eagerly waiting stakeholders
 3. Delivering the juiciest, most important stakeholder values that can be made in the least time
 - What will make Stakeholders more productive now
- **Not more than 2 weeks**



Tasks feed Deliveries



Task Cycle ↔ Delivery Cycle

Doing

Delivering

the right things, in the right order to the right level of detail

Optimizing

Estimation,
planning, tracking

Requirements,
assumptions

Selecting

Highest priority tasks

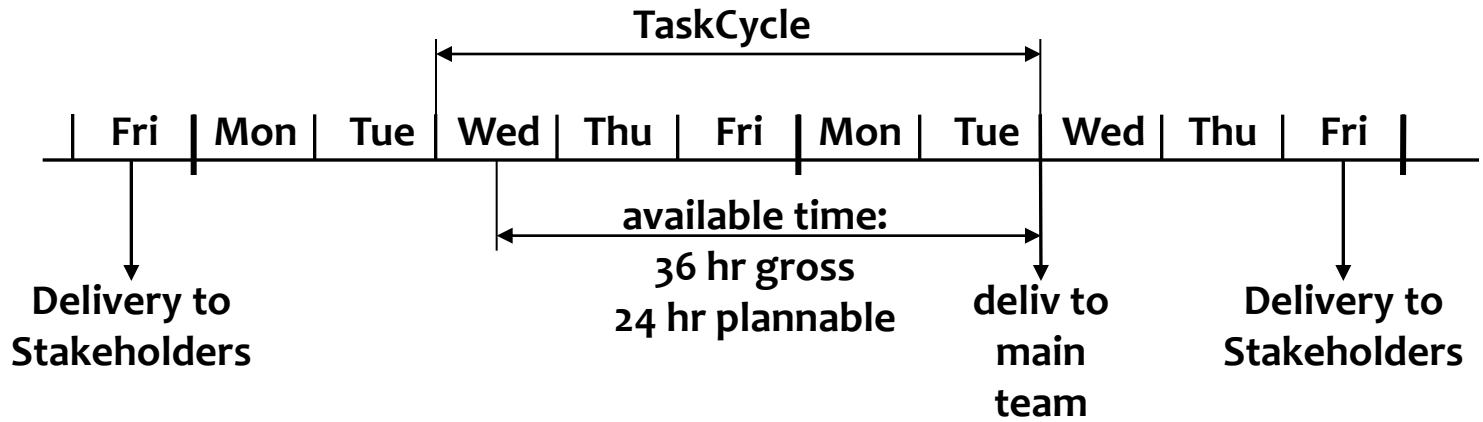
Most important values

≤ 1 week

≤ 2 weeks

Always done, 100% done

Designing a Delivery



Serge (ProjLead)

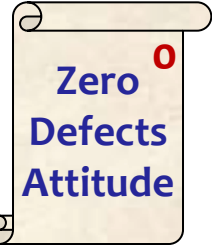
MbWA	3
Planning nxt wk	3
Work for deliv	4
-	6
-	2
-	1
-	5
Total	24

Gregory

Draft design	6
Finish design	6
Work for deliv	3
-	1
-	2
-	2
-	3
-	5
-	6
XMLa	4
XMLb	4
Total	42

Gregory (later)

Draft design	0
Finish design	0
...	
Repair deliv	0
...	
Jerome	
XMLa	3
XMLb	3
...	



TaskCycle Exercise

- How much time do you have available
- $\frac{2}{3}$ of available time is net plannable time
- What is most important to do (make list)
- Estimate effort needed to do these things
- Which most important things fit in the net available time (default 26 hr)
- What can you do, and what are you going to do
- What are you *not* going to do
- Why ?

Task _a	2	↑	
Task _b	5		
Task _c	3		
Task _d	6		do
Task _e	1		
Task _f	4		
Task _g	5		26
<hr/>			
Task _h	4	↓	
Task _j	3		do
Task _k	1		not

Agile, but will we be on time ?

- Organizing the work in very short cycles
- Making sure we are doing the right things
- Doing the right things right
- Continuously optimizing (what not to do)
- So, we already work more efficiently

but ...

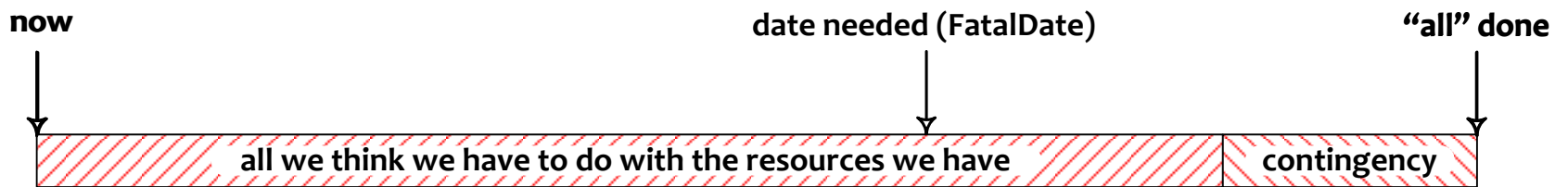
- How do we make sure the whole project is done on time ?

Evolutionary Planning

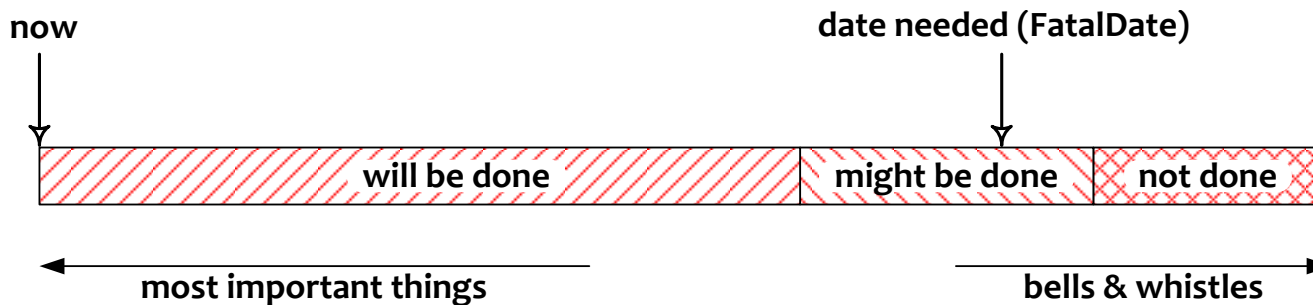
TimeLine

TimeLine

What the customer wants, he cannot afford



Standard Projects



Evo

Two options

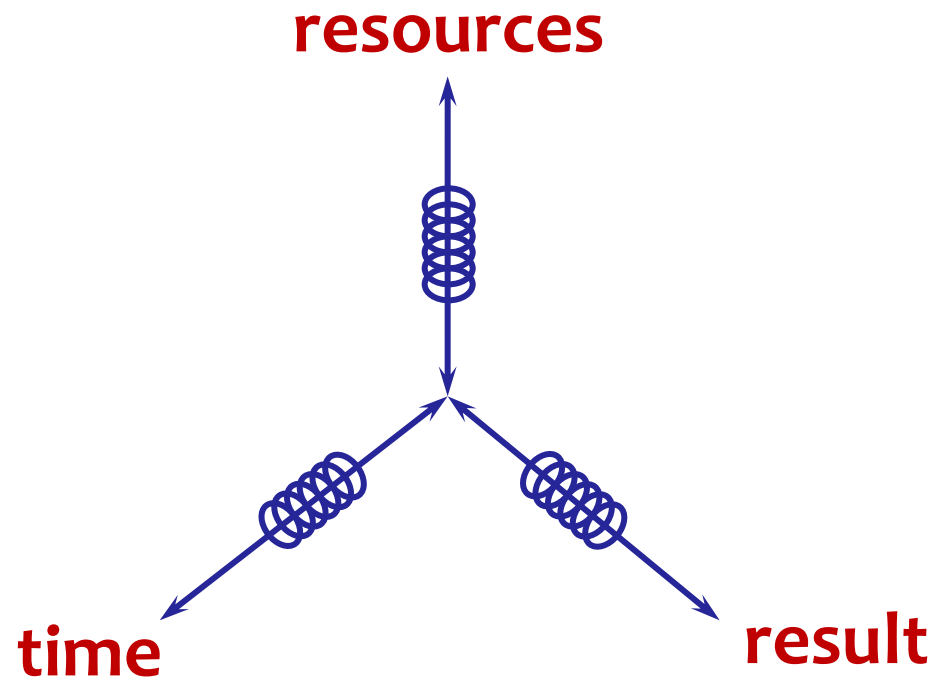
1. **Conventional option**
At the fatal day we'll tell we didn't succeed
2. **Evo option**
We already know we won't succeed, so we can tell it now,
then together we can decide what to do

Which option do you want?

Quality On Time is also being honest as soon as you can

The challenge is to find out as soon as you can

Dependencies



Priorities

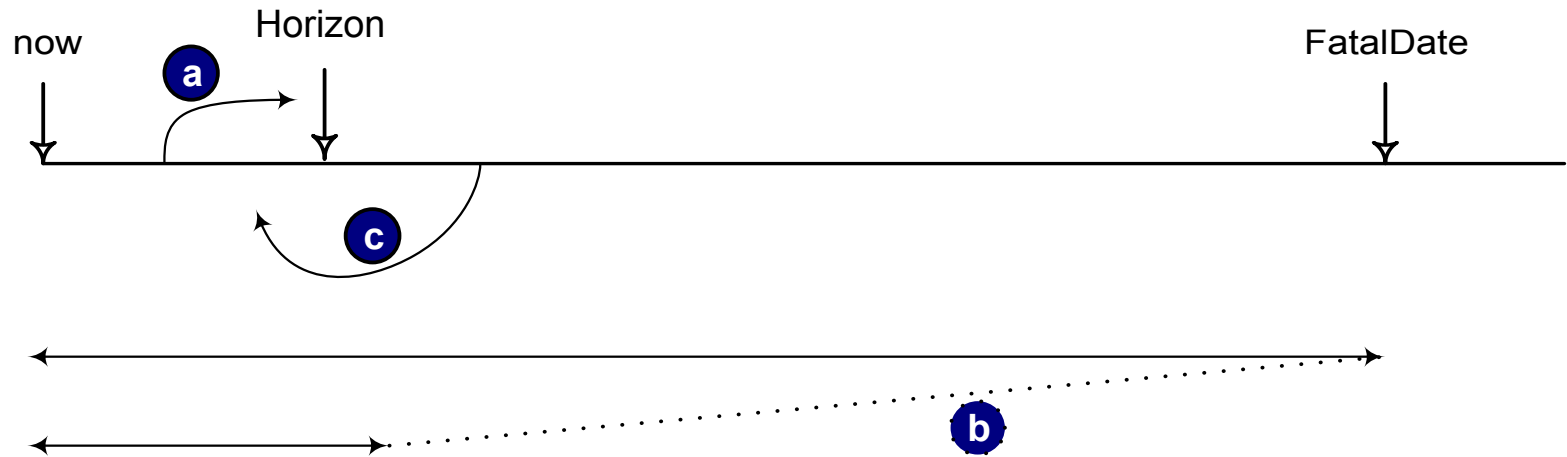
Better 80% 100% done, than 100% 80% done

Let it be the most important 80%

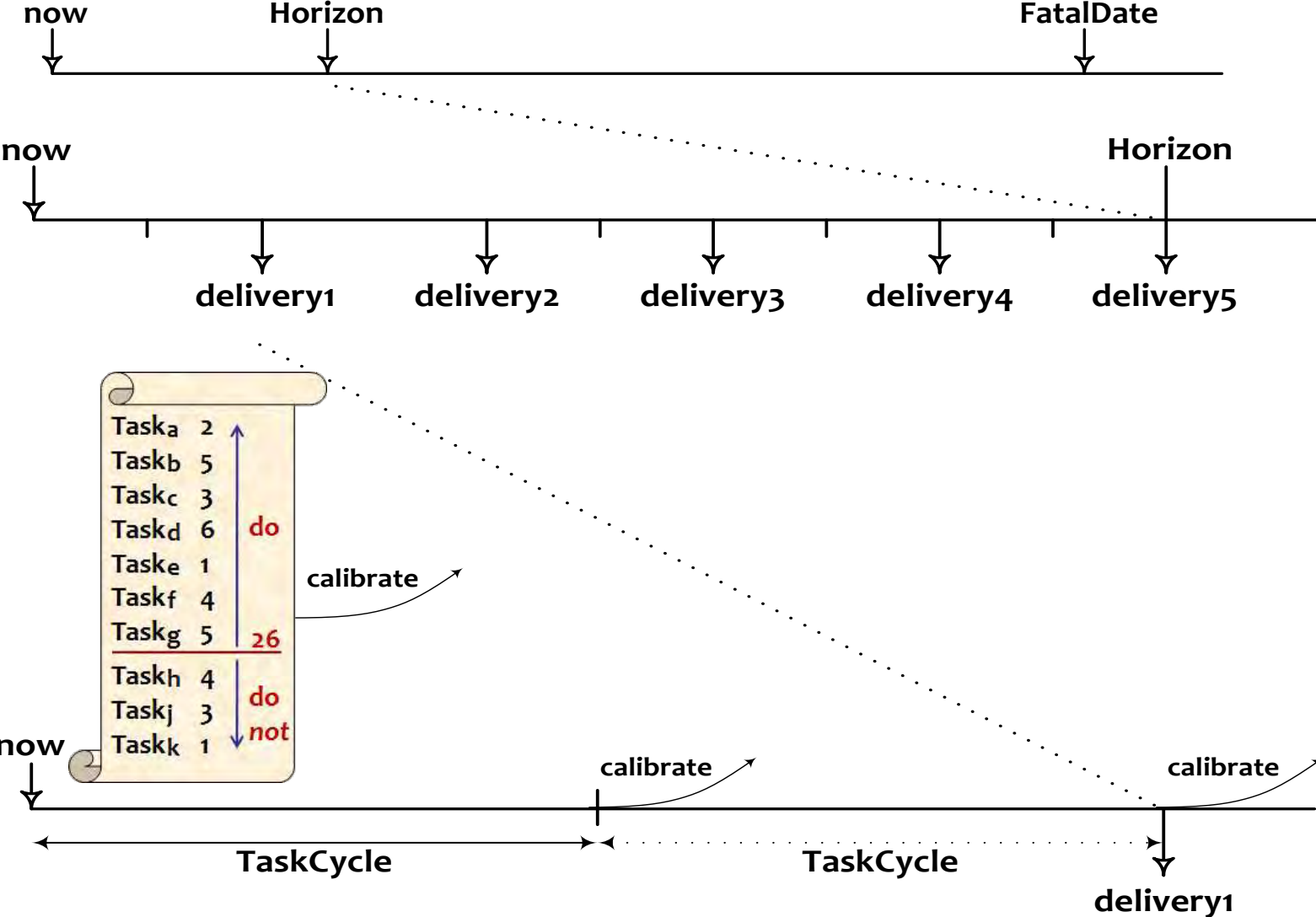
If it easily fits ...



Setting a Horizon

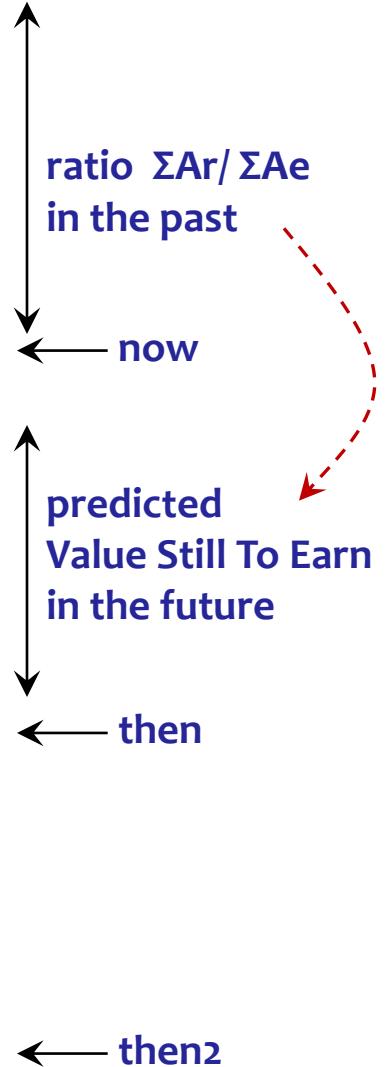


Result to Tasks and back



Calibration

Activity	Estimate	Real
Act1	Ae1	Ar1
Act2	Ae2	Ar2
Act3	Ae3	Ar3
Act4	Ae4	Ar4
Act5	Ae5	Ar5
Act6	Ae6	Ar6
Act7	Ae7	Ar7
Act8	Ae8	Ar8
Act9	Ae9	Ar9
Act10	Ae10	Ar10
Act11	Ae11	
Act12	Ae12	
Act13	Ae13	
Act14	Ae14	
Act15	Ae15	
Act16	Ae16	
Act17	Ae17	
Act18	Ae18	
Act19	Ae19	
Act20	Ae20	
Act21	Ae21	
⋮	⋮	
Act...	Ae...	



Calibration Factor

$$\frac{\sum_{now-1}^{now-n} Ar}{\sum_{now-1}^{now-n} Ae}$$

Value Still To Earn

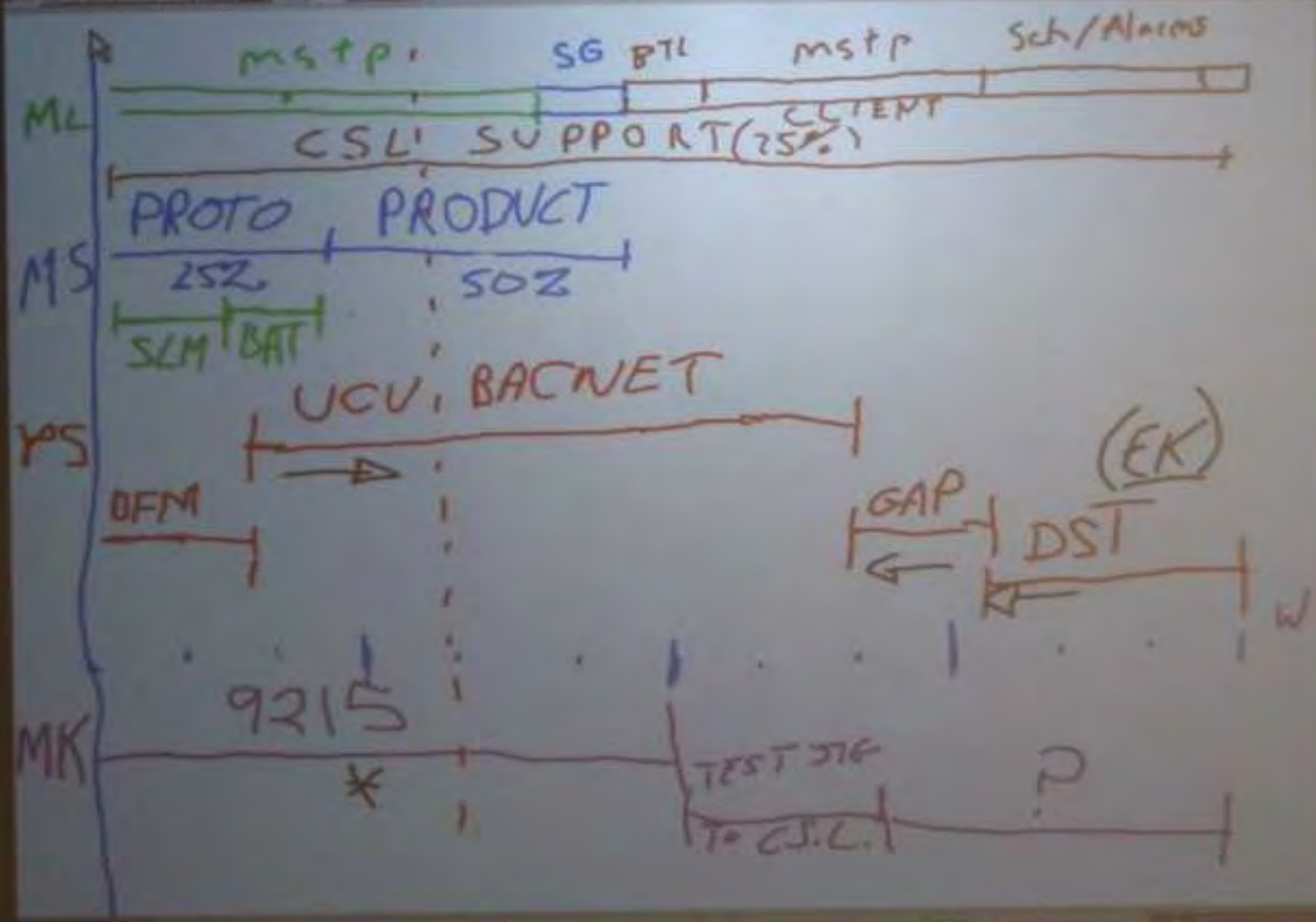
$$\text{Calibration Factor} * \sum_{now}^{then} Ae$$

Predicting *what* will be done *when*

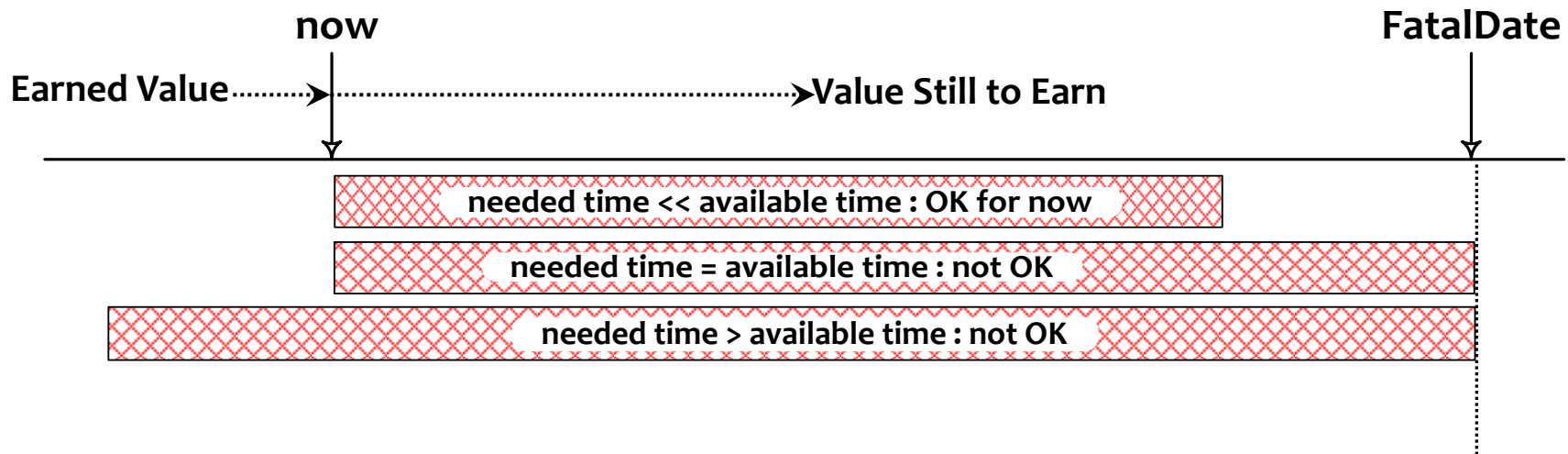
Line	Activity	Estim	Spent	Still to spend	Ratio real/es	Calibr factor	Calibr still to	Date done
1	Activity 1	2	2	0	1.0			
2	Activity 2	5	5	1	1.2	1.0	1	30 Mar 2009
3	Activity 3	1	3	0	3.0			
4	Activity 4	2	3	2	2.5	1.0	2	1 Apr 2009
5	Activity 5	5	4	1	1.0	1.0	1	2 Apr 2009
6	Activity 6	3				1.4	4.2	9 Apr 2009
7	Activity 7	1				1.4	1.4	10 Apr 2009
8	Activity 8	3				1.4	4.2	16 Apr 2009
↓	↓							
16	Activity 16	4				1.4	5.6	2 Jun 2009
17	Activity 17	5				1.4	7.0	11 Jun 2009
18	Activity 18	7				1.4	9.8	25 Jun 2009

Product/Portfolio/Resource Management

- **Current Program/Portfolio/Resource Management is based on hope**
- **More a game than management**
- **With TimeLine we can provide PPR Management with sufficiently reliable data**
- **To start managing**



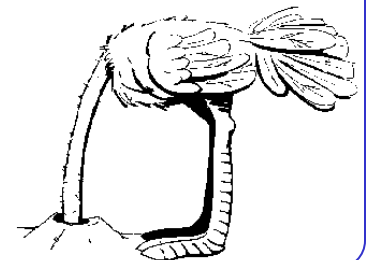
What do we do if we see we won't make it on time ?



- If it doesn't fit ... count backwards

FatalDay

- FatalDay is the last moment *it shall be there*
- After the FatalDay, we'll have real trouble if the Result isn't there
- Real Option Theory says that we should do things as late as possible, but not later
 - As late as possible, having the most up-to-date information to decide what to do
 - Not later: the option has expired; it has no value any more
- Count backwards from the FatalDay to know when we should have started
- If that's before now, what are we going to do about it, because *failure is not an option*



Deceptive options

- **Hoping for the best** (fatalistic)
- **Going for it** (macho)
- **Working Overtime** (fooling ourselves)
- **Moving the deadline**
 - **Parkinson's Law**
 - Work expands to fill the time for its completion
 - **Student Syndrome**
 - Starting as late as possible,
only when the pressure of the FatalDate is really felt

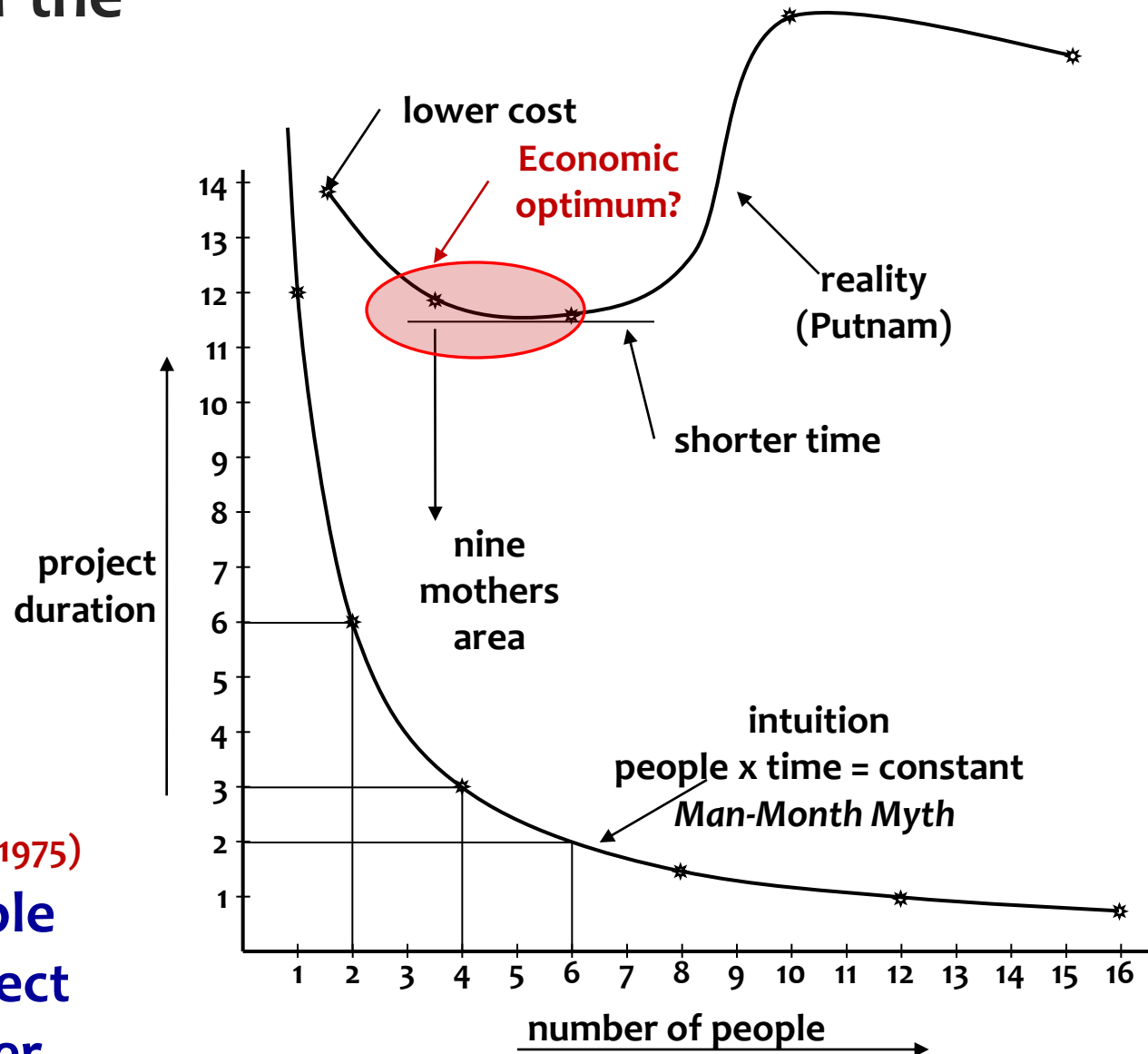
Adding people to a late project ...

makes it later

(Brooks' Law, 1975)

The Myth of the Man-Month

Brooks' Law (1975)
Adding people
to a late project
makes it later





Saving time

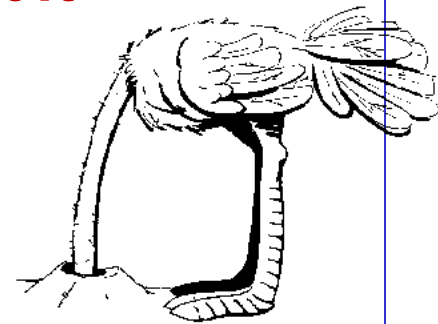
Continuous
elimination of waste

**We don't have enough time, but we can save time
without negatively affecting the Result !**

- **Efficiency in *what (why, for whom) we do*** - doing the right things
 - Not doing what later proves to be superfluous
- **Efficiency in *how we do it*** - doing things differently
 - The product
 - Using proper and most efficient solution,
instead of the solution we always used
 - The project
 - Doing the same in less time,
instead of immediately doing it the way we always did
 - Continuous improvement and prevention processes
 - Constantly learning doing things better
and overcoming bad tendencies
- **Efficiency in *when we do it*** - right time, in the right order
- **TimeBoxing** - much more efficient than FeatureBoxing

TimeLine

- The TimeLine technique doesn't solve our problems
- It helps to expose the real status **early and continuously**
- Instead of accepting the undesired outcome, *we do something about it*
- The earlier we know, the more we can do about it
- We start saving time from the very beginning
- We can save a lot of time in any project, while producing a better outcome



If, and only if, we are serious about time !

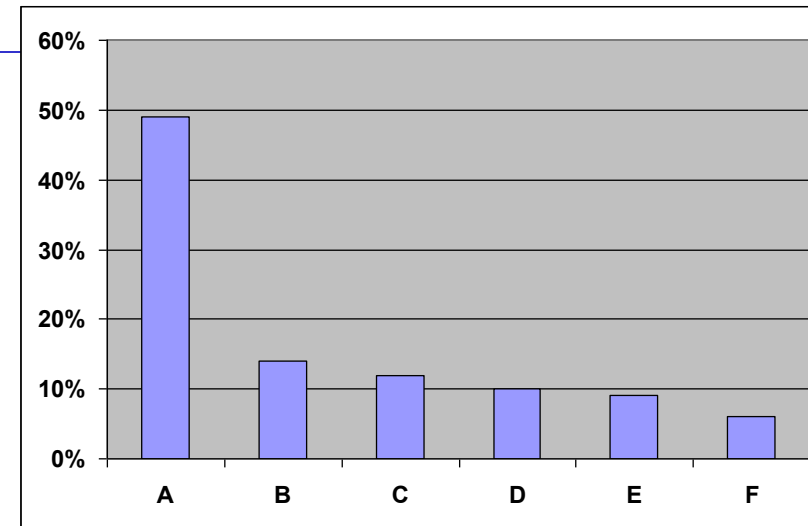
Estimation

Estimation techniques used

- **Just-enough estimation** (don't do unnecessary things)
 - Maximizing Return-on-Investment and Value Delivered
- **Changing from optimistic to realistic predictions**
 - Estimation of Tasks in the TaskCycle
 - Prediction what will be done when in TimeLine
- **0th order estimations** (ball-park figures)
 - For decision-making in Business Case and Design
- **Simple Delphi**
 - For estimating longer periods of time in TimeLine
 - For duration of several (15 or more) elements of work
- **Simpler Delphi**
 - Same, but for quicker insight
 - Recently added by practice
- **Calibration**
 - Coarse metrics provide accurate predictions
- **Doing something about it** (if we don't like what we see)
 - Taking the consequence
 - Saving time



The Pareto principle (20 - 80 rule)



A collection of problems always can be divided into a small number of large problems and a large number of smaller problems

- **The *vital few* are dealt with individually**
- **The *useful many* are dealt with as a group**

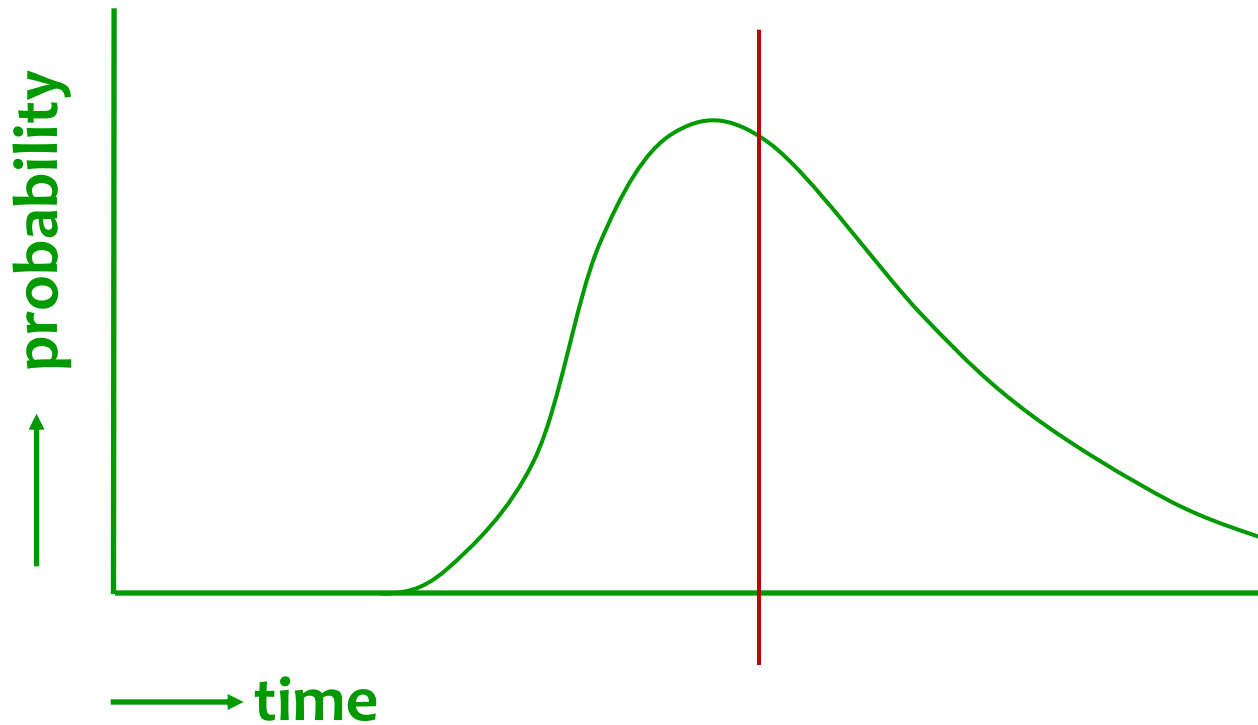
Juran, 1960

**Example: better have 80% of the Requirements 100% done than 100% 80% done
It may even take only 20% of the resources**

Realistic estimation in 3 weeks

- In 3 weeks people can change estimation from optimistic to realistic
- 1st week 40%, 2nd 80%, 3rd week 100%
- Commitment
- Use 'the mirror'
 - Commitment: they see themselves in the mirror
 - No commitment: they see you

Variation often is non-symmetric



Simple Delphi estimation



1. **Make a list of things we think we have to do in just enough detail**
2. **Distribute the list among people who will do the work, or who should be knowledgeable about the work**
3. **Ask them to add what we apparently forgot, and to estimate how much time the elements of work would cost, “as far as you can judge”**
4. **In a meeting the estimates are compared**
5. **If estimates differ significantly between estimators, *do not take the average*, but discuss about the contents of the work, *not about the estimate* (some may forget to include things that have to be done, some others may think that more has to be done than necessary)**
6. **After discussion, people estimate *individually* again and the estimates are compared again**
7. **Repeat until sufficient consensus (usually not more than once or twice)**
8. **Add up all the estimates to end up with an estimate for the whole project**

Simple and Simpler Delphi



1. List things to do
2. Distribute the list
3. Add and estimate
4. List estimates
5. Discuss if differences
6. Estimate again
7. Repeat until consensus
8. Add up all the estimates

1. List things to do
2. Distribute the list
3. Add and estimate
4. List estimates: min and max
5. Discuss if differences
6. Agree on value between min and max
7. Add up all the estimates

Even with coarse estimates per element of work,
the sum averages out the variations and can be quite predictive

0th- order approximations

- In the Business Case we often use 0th- order estimations
- Order of magnitude
- Better than $0 < \text{guess} < \infty$ (*any number is better than no number*)
- 0th order is better than no clue
- 1st order is often less accurate than 0th order
- Using two different ways of estimation for crosscheck
- Errors will average if we estimate several pieces

Optimizing Estimation

- **Immediately consuming the metrics for learning**
- **Change from optimistic to realistic estimation in 3 weeks**
- **Only if we are Serious about Time (Sense of Urgency)**
- **Using the metrics for calibration of predictions**
- **Estimation method: Intuition + optimizing intuition**
- **The person doing the task estimates**
- **Others should never challenge the estimation**
- **Estimates are non-negotiable !**
- **We can and should negotiate about the contents**

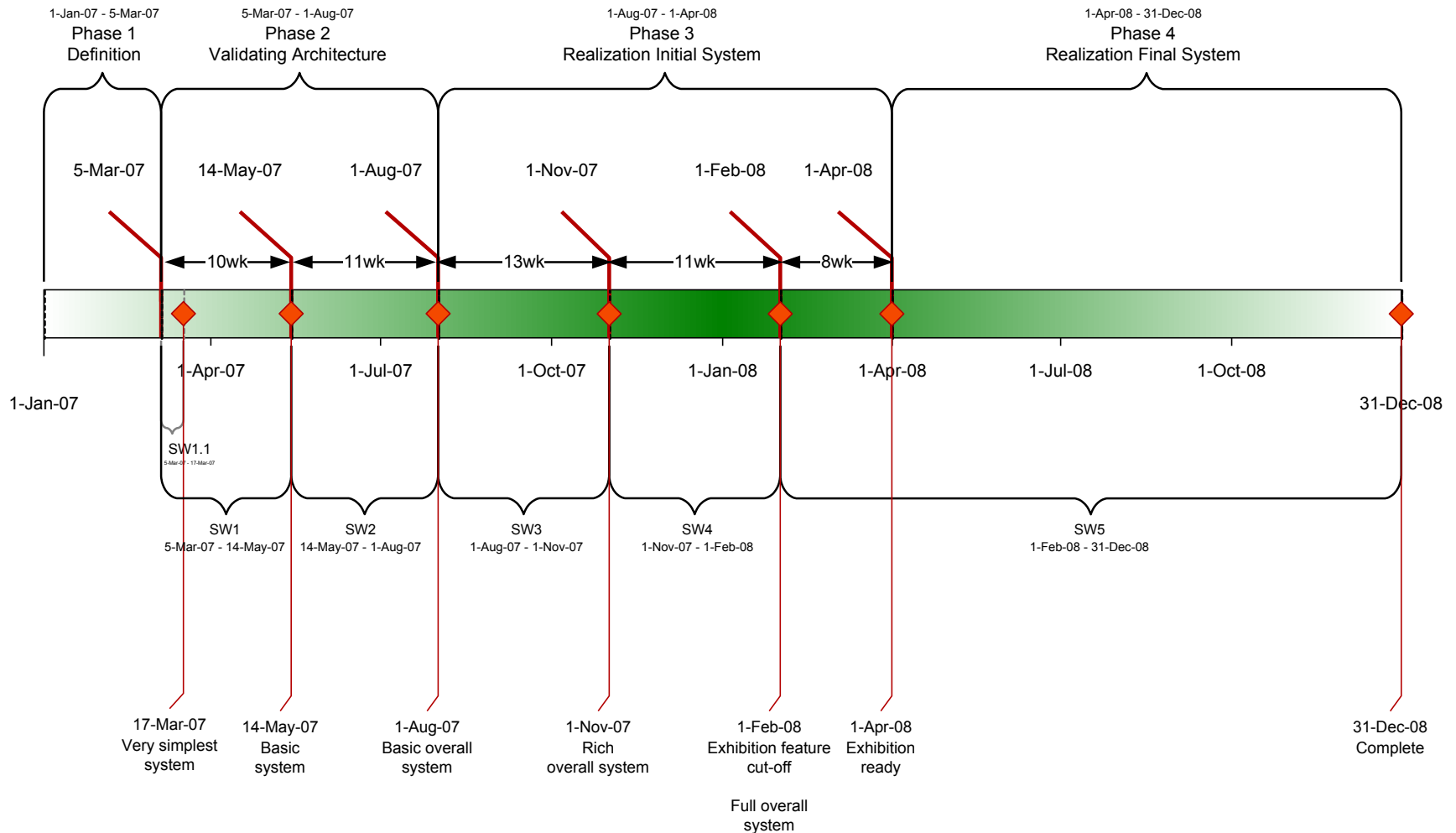
Evo Planning: Weekly TaskCycle



- Goal is not to be a good estimator
- Goal is to learn to promise what you can and will do and then to live up to your promises
- It's easier to estimate in hours than in pieces of cake
- We estimate net effort to do the work
- All work to be 100% done at the end of the week
- We plan $\frac{2}{3}$ of the available time
- The other $\frac{1}{3}$ is for all other things we'll do anyway
- We only work on planned things

TimeLine examples

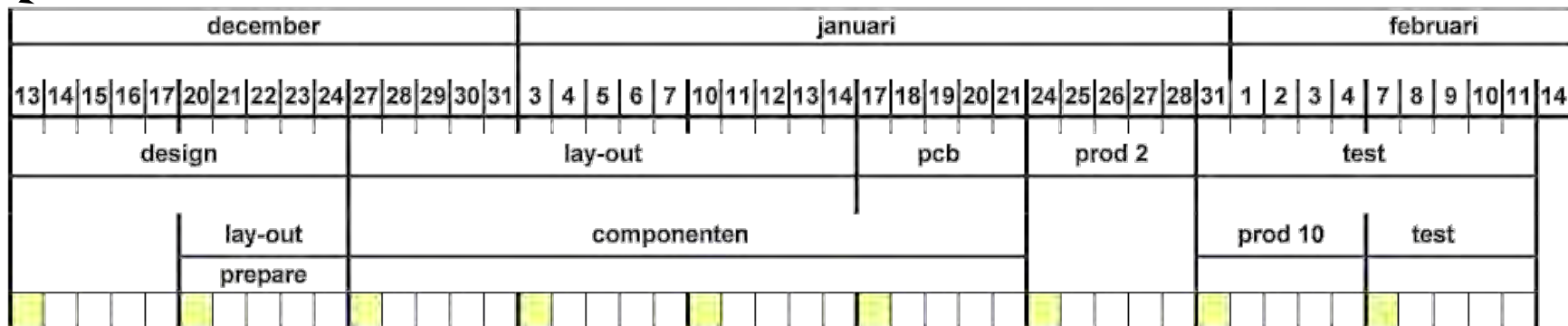
TimeLine example



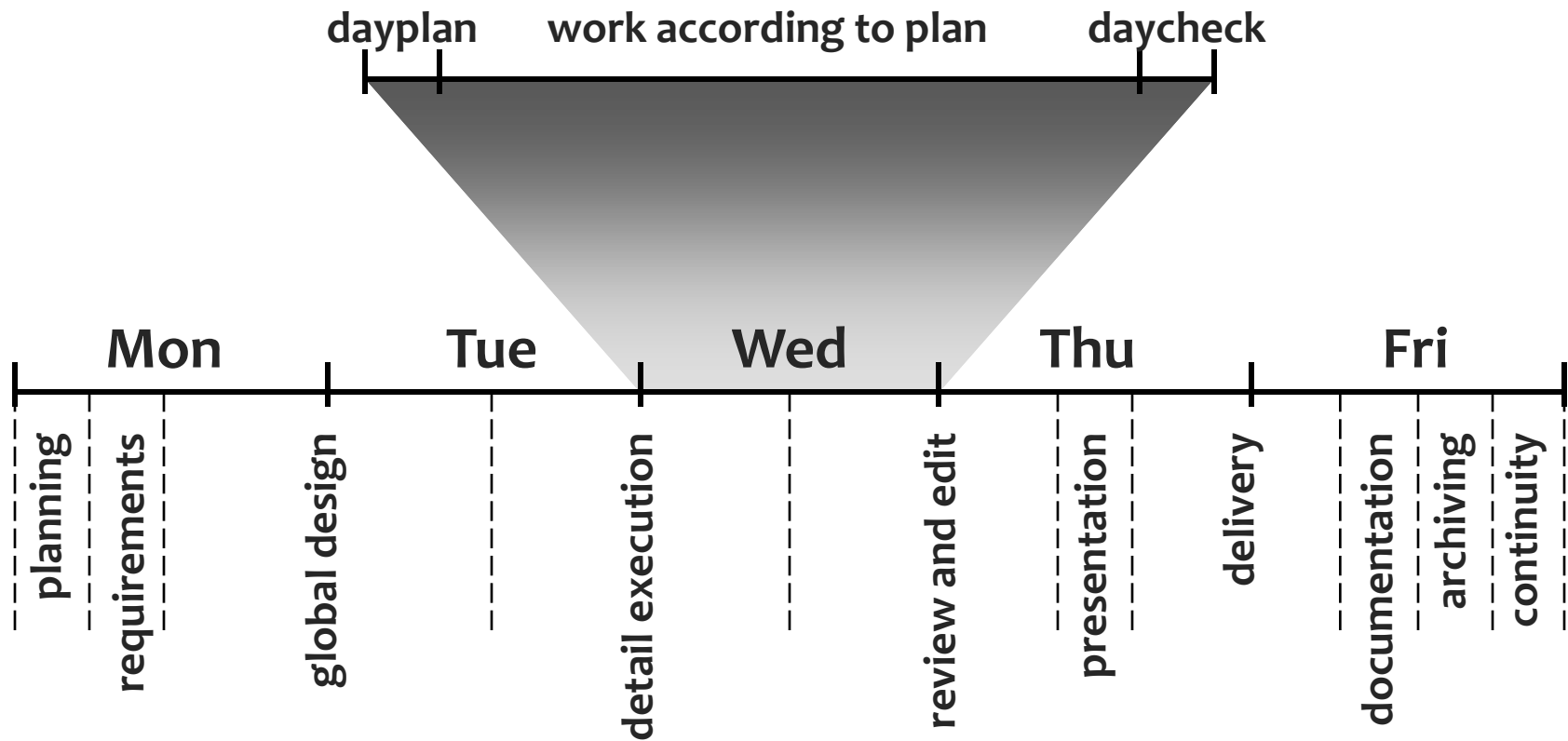
TimeLine planning



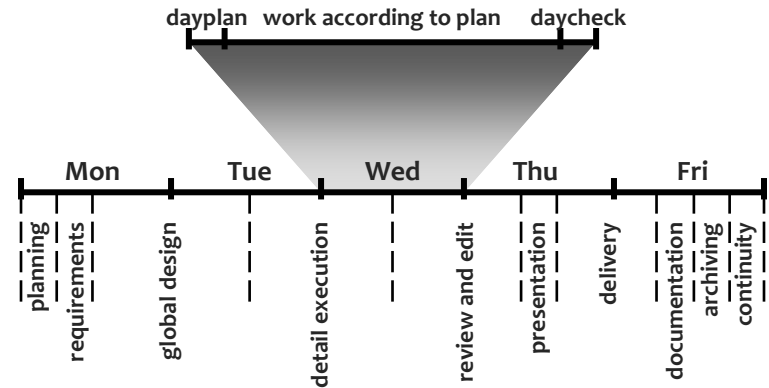
FatalDate: 11 feb



5 day project model



Available TimeBoxes



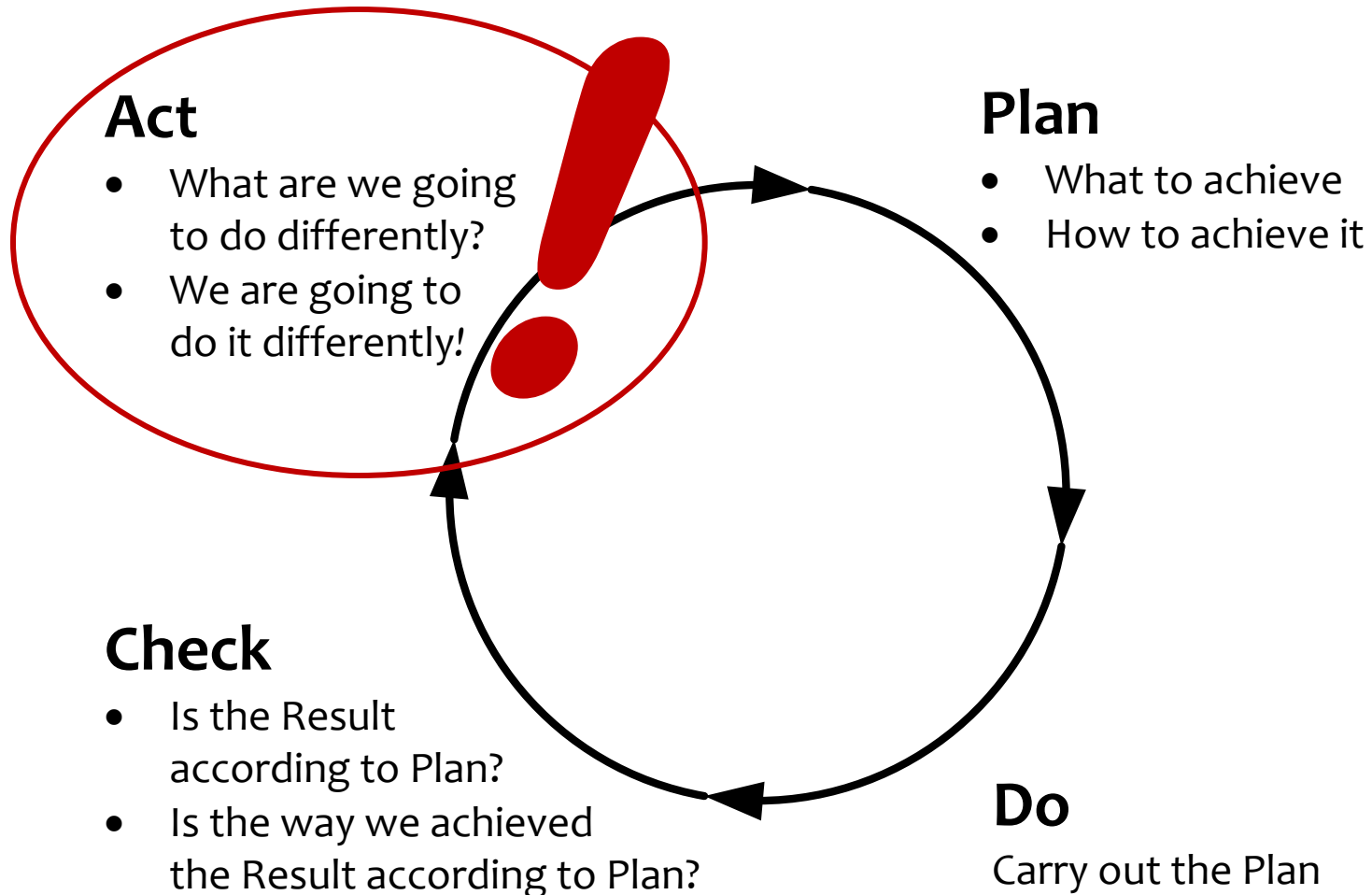
activity	~%	hrs
Planning	5	2
Requirements	5	2
Global design	20	8
Detail execution	20	8
Review and edit	20	8
Presentation	5	2
Delivery	10	4
Documentation	5	2
Archiving	5	2
Continuity	5	2
total	100	40

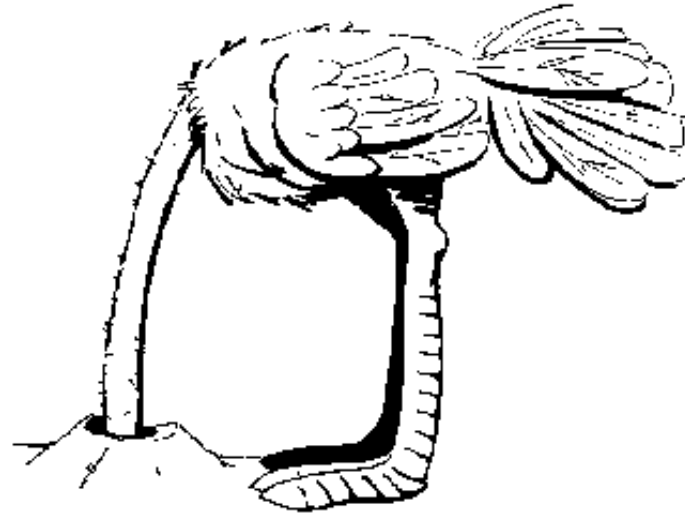
Help ! We have a QA problem !

- **Large stockpile of modules to test**
(hardware, firmware, software)
- **You shall do Full Regression Tests**
- **Full Regression Tests take about 15 days each**
- **Too few testers** (“Should we hire more testers ?”)
- **Senior Tester paralyzed**
- **Can we do something about this?**



Do you think you can help us ?





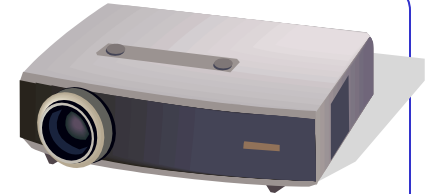
In stead of complaining about a problem ...

(Stuck in the Check-phase)

Let's do something about it !

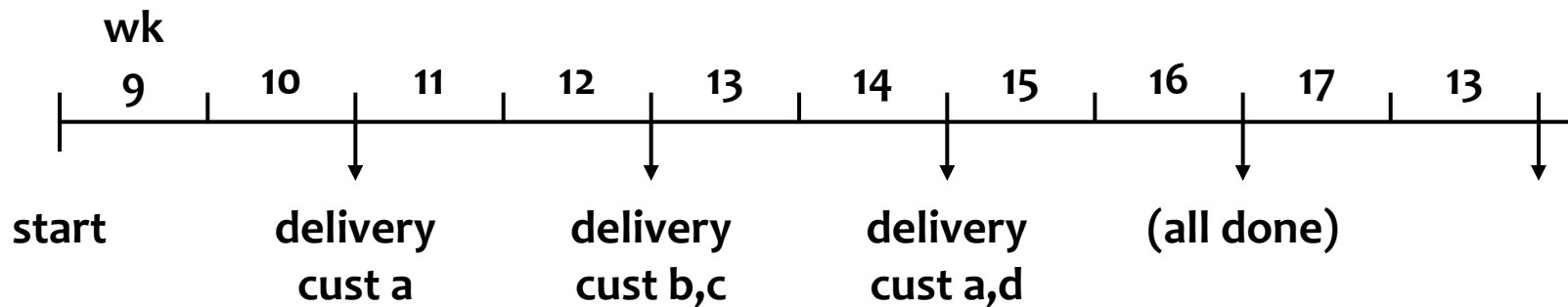
(Moving to the Act-phase)

Objectifying and quantifying the problem is a first step to the solution



Line	Activity	Estim	Alter native	Junior tester	Devel opers	Customer	Will be done (now=22Feb)
1	Package 1	17	2	17	4	HT	
2	Package 2	8	5		10	Chrt	
3	Package 3	14	7	5	4	BMC	
4	Package 4 (wait for feedback)	11				McC?	
5	Package 5	9	3		5	Ast	
6	Package 6	17	3	10	10	?	
7	Package 7	4	1		3	Cli	
8	Package 8.1	26	1			Sev	
9	Package 8.2	1	1			?	
10	Package 8.3	1	1			Chrt	24 Feb
11	Package 8.4	1	1			Chrt	
12	Package 8.5	1.1	1.1			Yet	28 Feb
13	Package 8.6	3	3			Yet	24 Mar
14	Package 8.7	0.1	0.1			Cli	After 8.5 OK
15	Package 8.8	18	18			Ast	
	totals	106	47	32	36		

TimeLine



Selecting the priority order of customers to be served

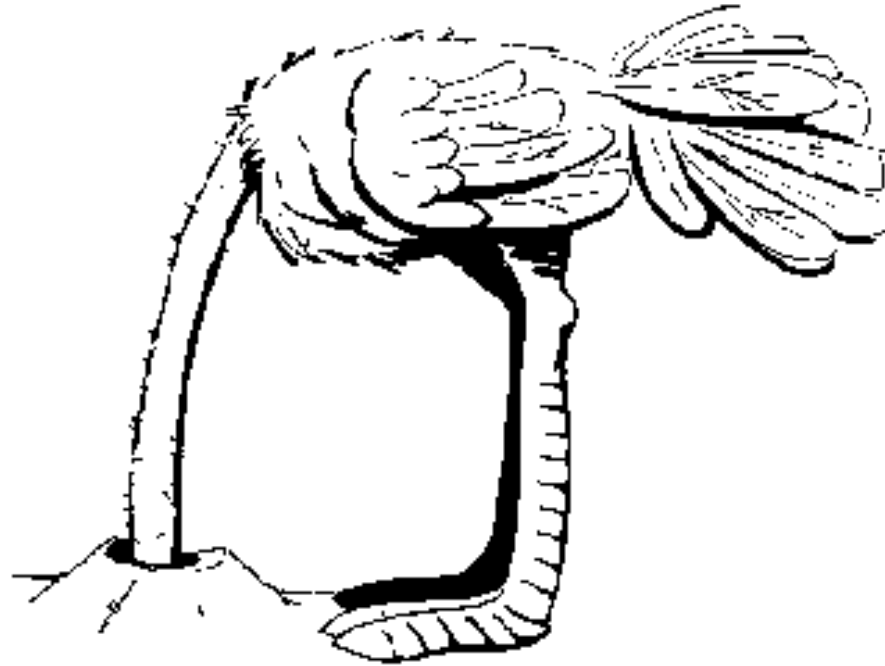
- “We’ll have a solution at that date ... Will you be ready for it ?”
An other customer could be more eagerly waiting
- Most promising customers

Result

- **Tester empowered**
- **Done in 9 weeks**
- **So called “Full Regression Testing” was redesigned**
- **Customers systematically happy and amazed**
- **Kept up with development ever since**
- **Increased revenue**

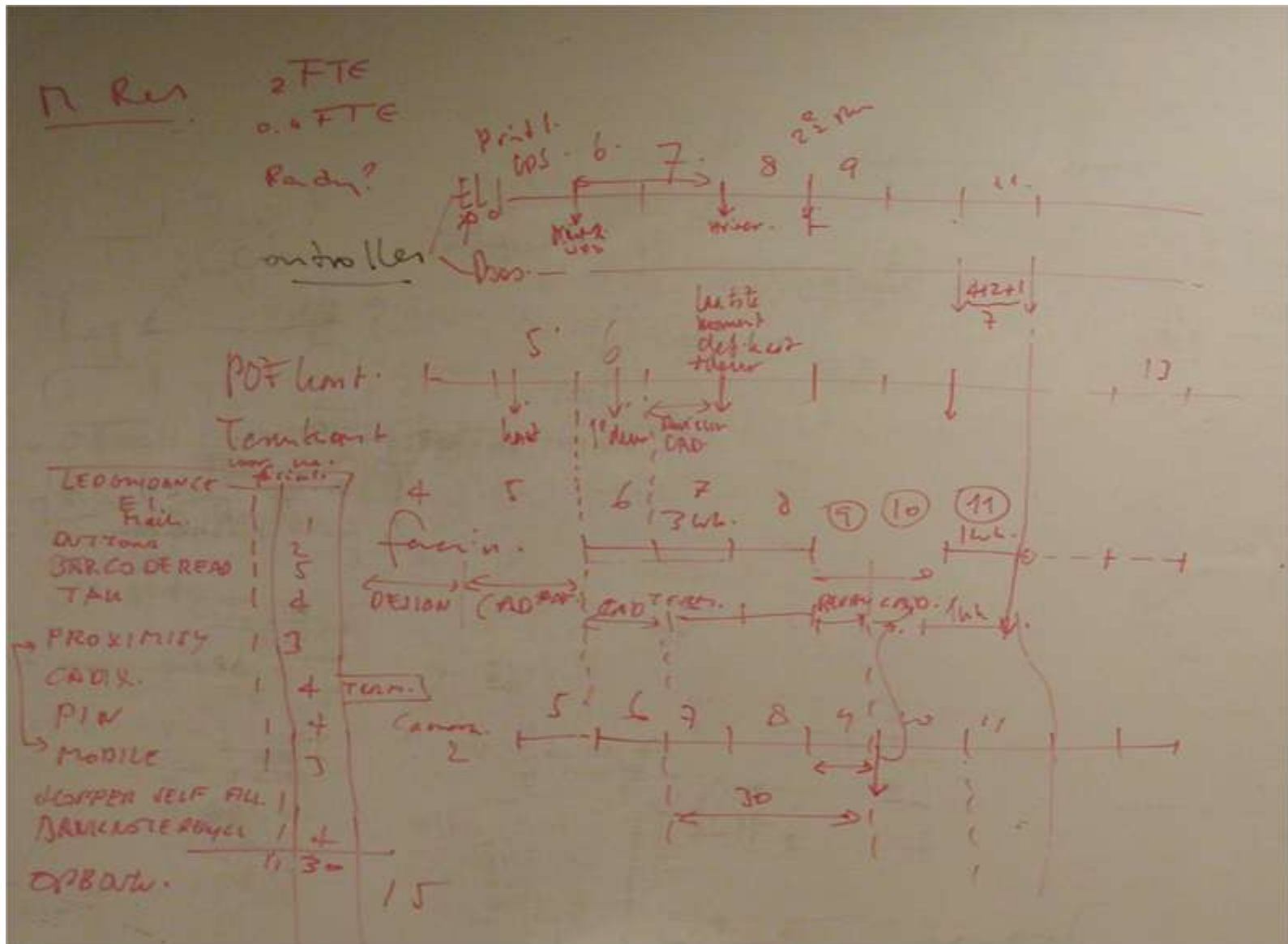
Recently:

- **Tester promoted to product manager**
- **Still coaching successors how to plan**

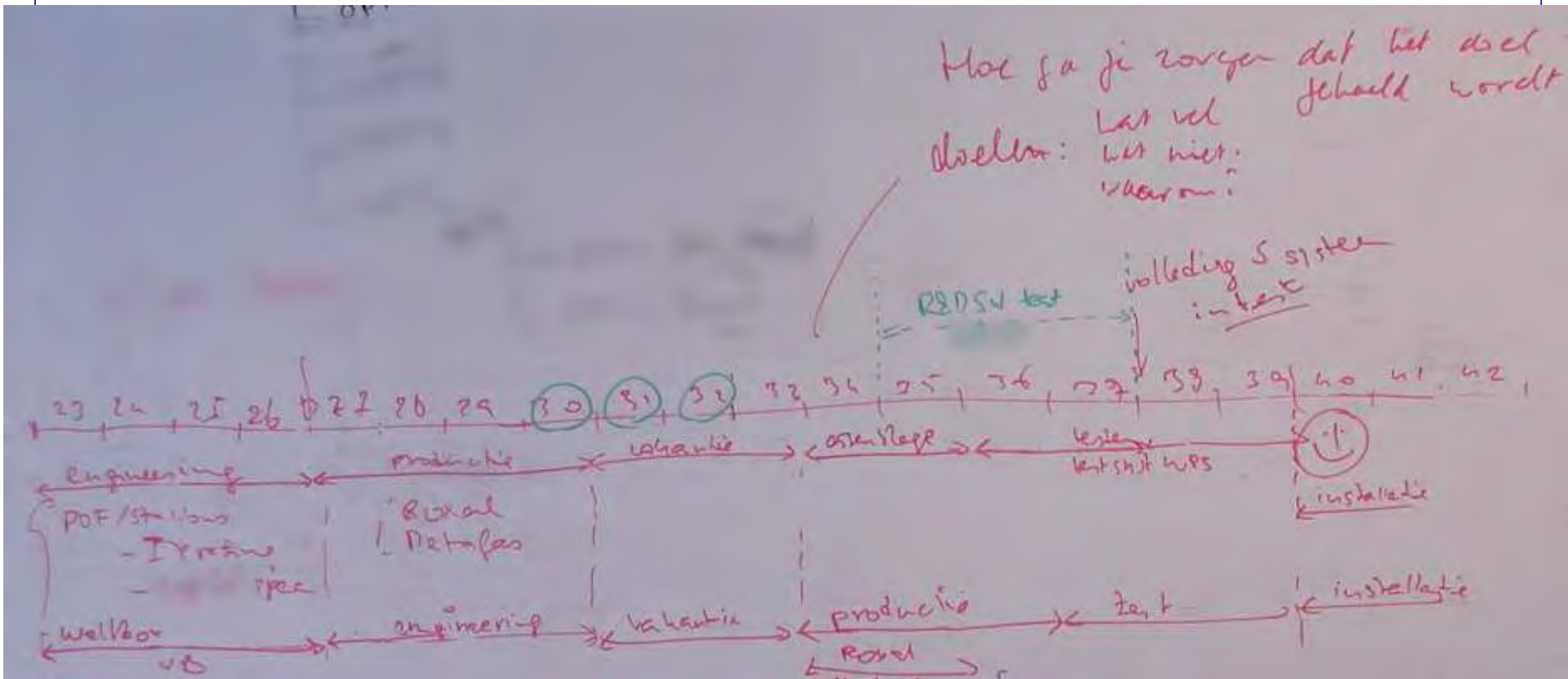


**The problems in projects are not the real problem,
the real problem is that we don't do something about it**

Whiteboard TimeLine Planning



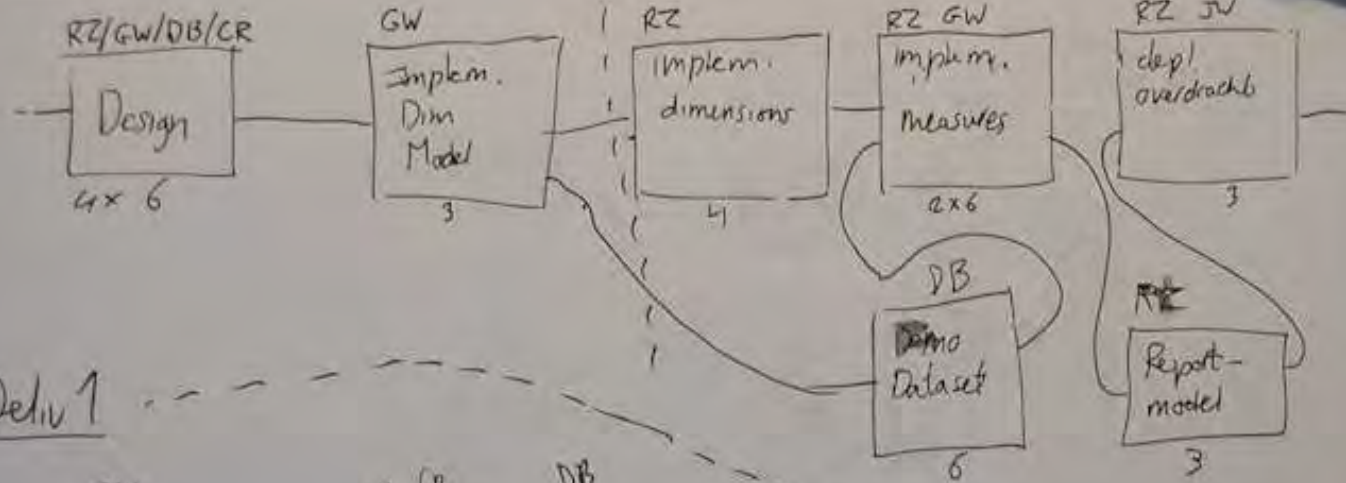
Whiteboard TimeLine Planning



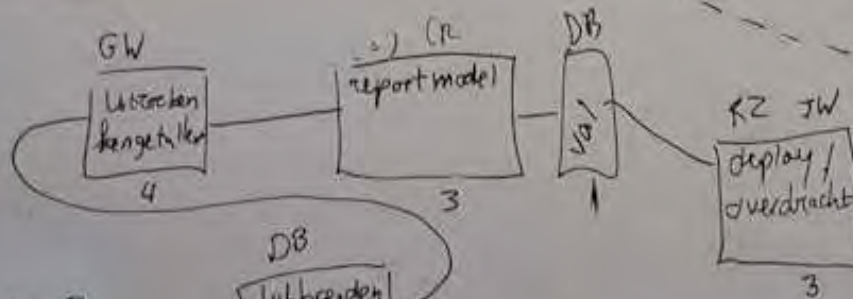
Making individual TimeLines



Deliv 2



Deliv 1

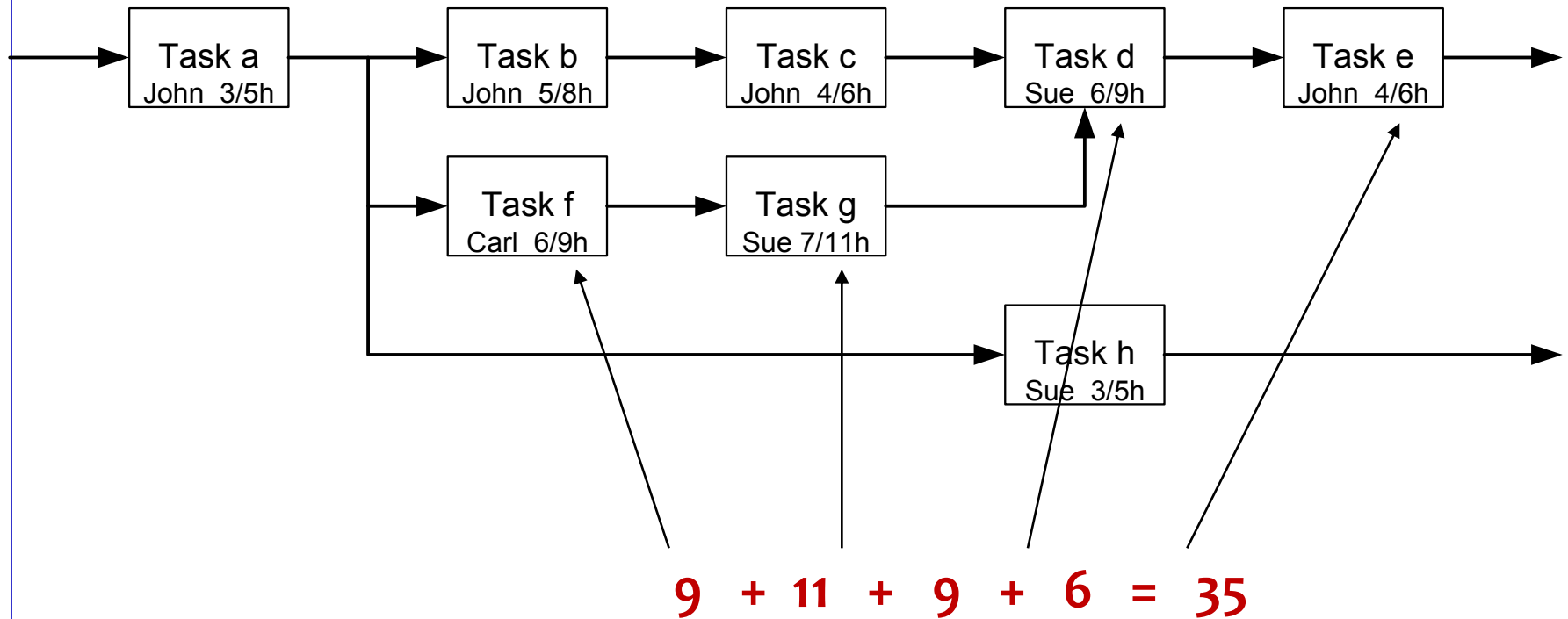


Deliv 3



PERT (Project Evaluation Review Technique)

Designing a Delivery

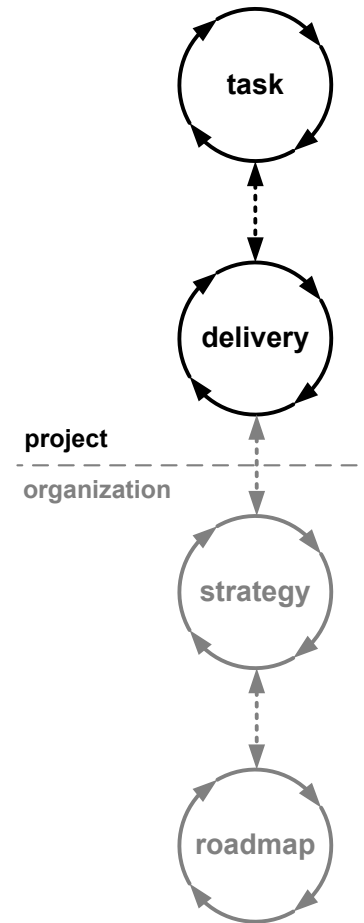


Some details

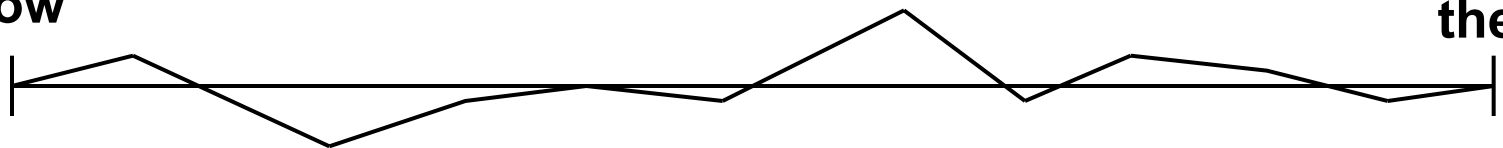
Tasks - Deliveries - Projects

Tasks - Deliveries - Projects
actually are similar, except for
the time and complexity scales

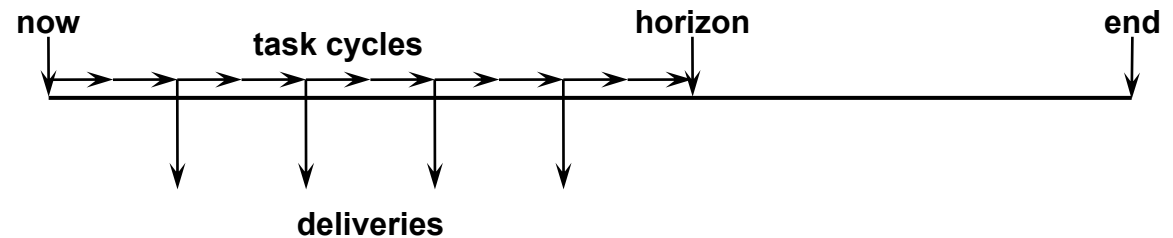
- At the end there is a defined Result, 100% done
- The journey to the Result should be *designed*



now



What if ...

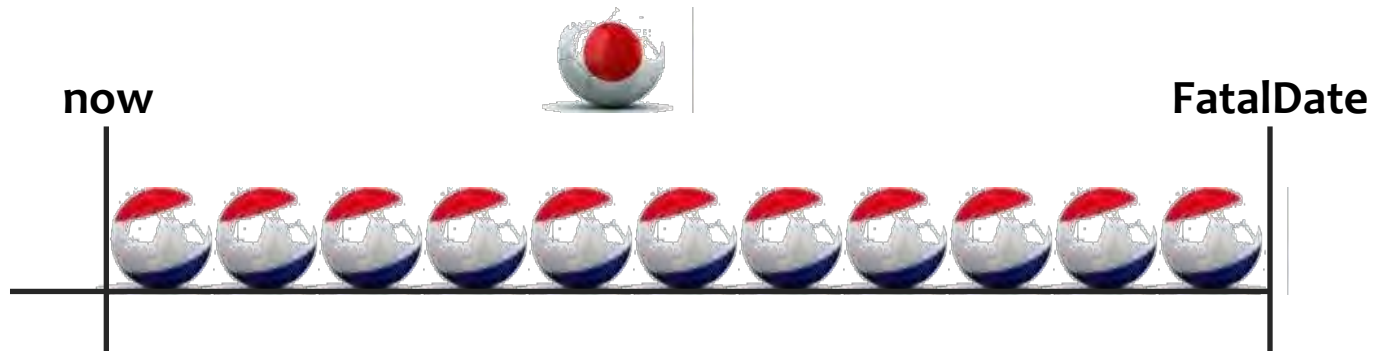


- **Somebody gets sick**
 - Swap deliveries
- **Requirement change (functional, performance, constraint)**
 - Never say “Is impossible, we have no time”, rather:
 - Of course it is possible, tell me what to leave out
 - Where shall we fit it in the TimeLine?

TimeLine provides control over the Project

If we add something ...

If we add something, something else will not be done



Making best use of limited available time

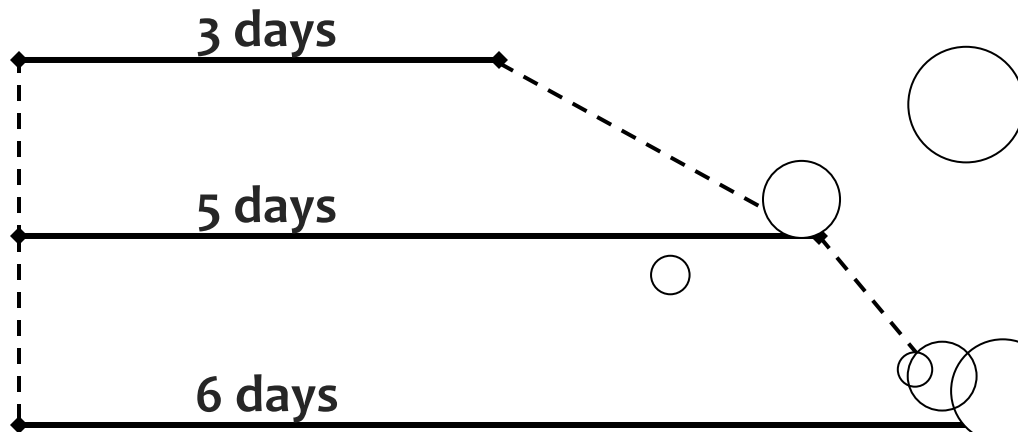
- **If the work is done, the time is already spent**
- **If we still have to do the work, we can decide**
 - What is really important
 - What is less important
 - What we must do
 - What we can do
 - What we are going to do
 - What we are not going to do
- **Therefore we plan first, in stead of finding out later**
- **We cannot work in history**

TimeBox

- taking Time seriously

- **A TimeBox is the maximum time available for a Task**
- **When the time is up, the Task should be completely done: there is no more time !**
- **Because people tend to do more than necessary**
(especially if the requirements of the Task are unclear)
 - Check halfway whether you're going to succeed on time
 - If not: what can you do less, without doing too little
 - Define the requirements of the Task well
 - If the TimeBox is unrealistic: take the consequences (pdAct) immediately
(if a Task suddenly proves to need much more time, is it still worth the investment?)
- **If you really cannot succeed within the TimeBox:**
 - Check what you did
 - Check what you didn't do
 - Check what still has to be done
 - Define new Tasks with estimations (TimeBoxes !)
 - Stop this Task to allow for finishing the other committed Tasks
(don't let other Tasks randomly be left undone)

Parkinson's Law



Evo

- Do 3 days in 5 days!
- Success
- Unstress
- Energy
- Motivation = Motor of productivity
- Higher productivity!!

Standard Management

- Do 6 days in 5 days!
- Never succeed
- Frustration
- De-motivation
- Stress
- Higher productivity??

“Work expands to fill the time available”

Active Synchronization

Somewhere around you, there is the bad world.

If you are waiting for a result outside your control, there are three possible cases:

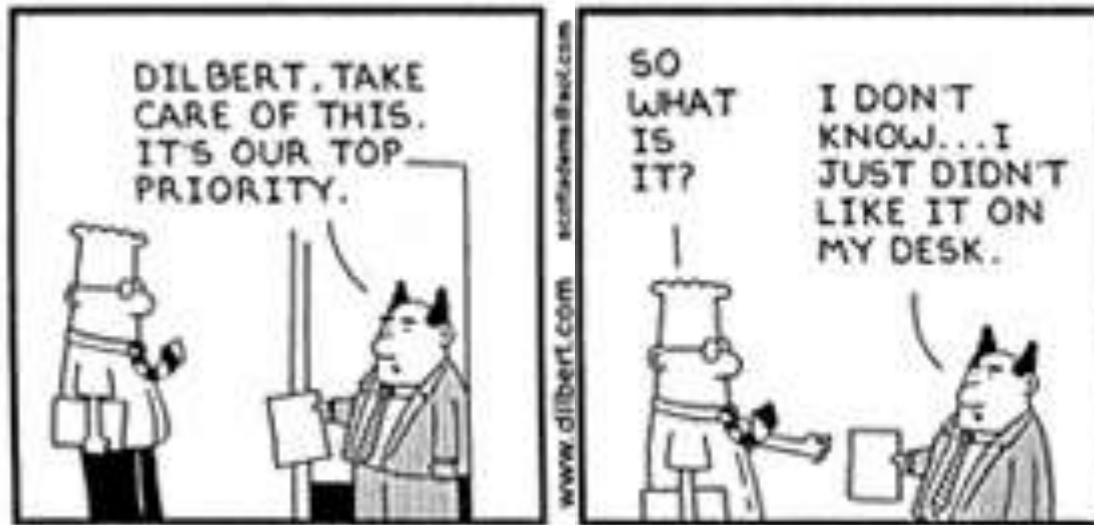
1. You are sure they'll deliver Quality On Time
2. You are not sure
3. You are sure they'll not deliver Quality On Time
 - If you are not sure (case 2), better assume case 3
 - From other Evo projects you should expect case 1
 - Evo suppliers behave like case 1

In cases 2 and 3: Actively Synchronize: Go there !

1. Showing up increases your priority
2. You can resolve issues which otherwise would delay delivery
3. If they are really late, you'll know much earlier

Interrupts

- Boss comes in: “Can you paint my fence?”
- What do you do?



- In case of interrupt, use interrupt procedure

Interrupt Procedure "We shall work only on planned Tasks"

In case a new task suddenly appears in the middle of a Task Cycle (we call this an Interrupt) we follow this procedure:

- 1. Define the expected Results of the new Task properly**
- 2. Estimate the time needed to perform the new Task, to the level of detail really needed**
- 3. Go to your task planning tool (many projects use the ETA tool)**
- 4. Decide which of the planned Tasks is/are going to be sacrificed (up to the number of hours needed for the new Task)**
- 5. Weigh the priorities of the new Task against the Task(s) to be sacrificed**
- 6. Decide which is more important**
- 7. If the new Task is more important: replan accordingly**
- 8. If the new Task is not more important, then do not replan and *do not work* on the new Task. Of course the new Task may be added to the Candidate Task List**
- 9. Now we are still working on planned Tasks.**

Task selection criteria

- **Most important requirements first**
- **Highest risks first**
- **Most educational or supporting for development first**
- **Actively Synchronize with other developments**
- **Every cycle delivers a useful, completed, result**

Delivery selection criteria

1. What will generate the optimum feedback
 2. We deliver only to eagerly waiting stakeholders
 3. Delivering the juiciest, most important stakeholder values that can be made in the least time
- What will make Stakeholders more productive now
 - **Every delivery must have a useful set of stakeholder values (features, qualities), otherwise the stakeholders get stuck**
 - Delete ↔ Add
 - Copy ↔ Paste
 - **Every new delivery must have clear extras, otherwise the stakeholders won't keep producing feedback**
 - **Every delivery delivers smallest clear increment, to get the most rapid and most frequent feedback**
 - **If a delivery takes more than two weeks, it can usually be shortened: try harder**

Types of Tasks

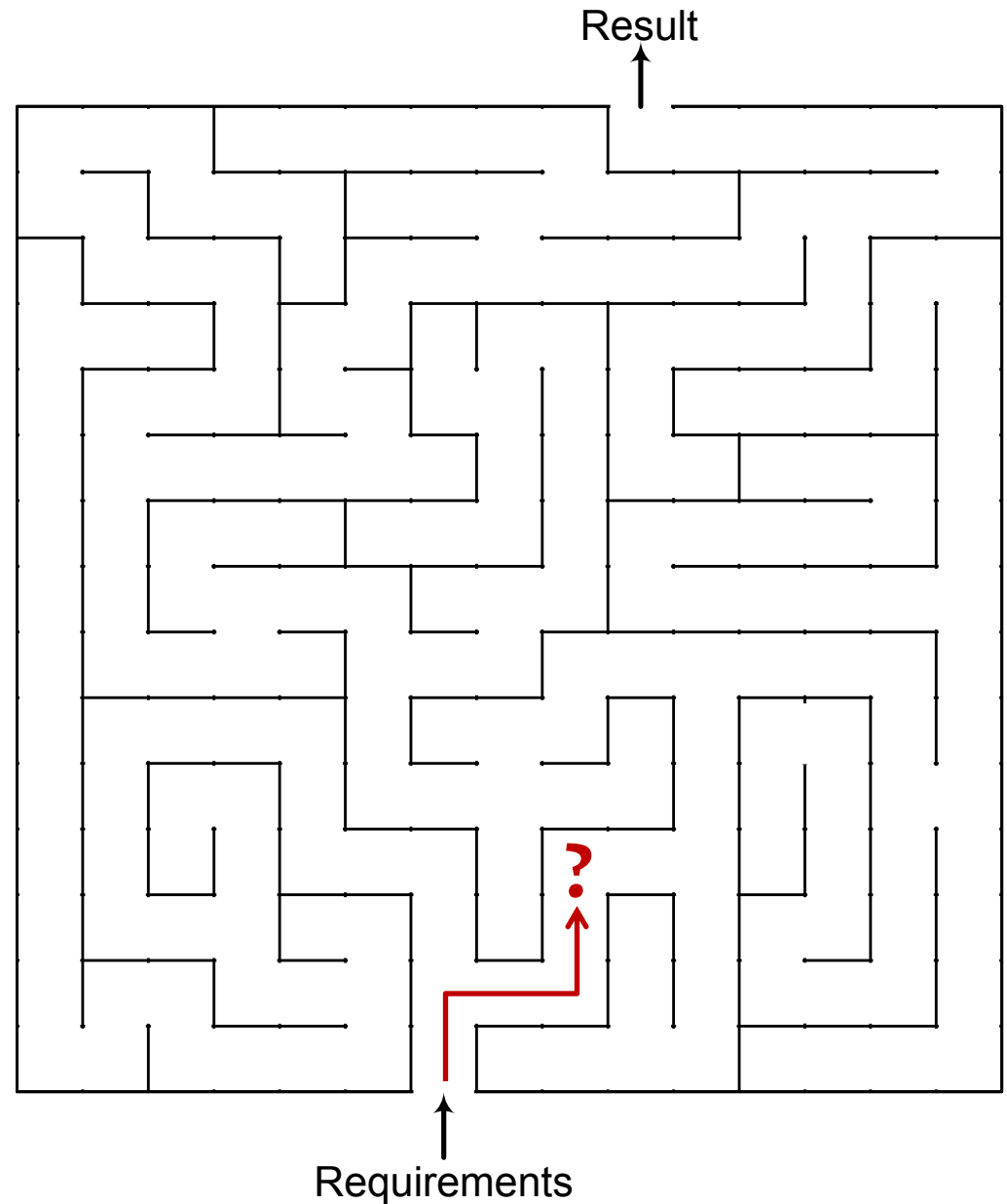
- 1. Tasks done within estimated time** (= timebox)
- 2. Analysis Tasks** (*too short timebox*)
 - What do you know now
 - What do you still not know
 - What do you still have to know
 - Which tasks can you define
- 3. Mis-estimated tasks** (we're only human)
 - Feed the disappointment about the failure to your experience/intuition mechanism
 - What did you do
 - What did you not do
 - What do you still have to do
 - Which tasks can you define

Smallest step with highest value

- **Evo tries the smallest possible step**
 - If the result proves to be incorrect, we have to redo as little as possible
 - The earlier we are done, the more time we have in our future
 - Just enough to see we're on the right track
- **Because our (and their !) assumptions may be wrong**

**What is the
shortest way
through the maze ?**

**How quickly
do we know
an alley
is the main road
or a dead-end?**



Accepting a Task

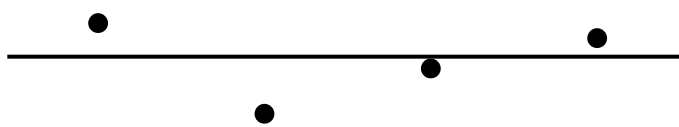
- **Accepting a Task means:**
 - Taking full responsibility for the successful conclusion of the Task within the time agreed
- **This also means:**
 - Once you know that you will not be able to conclude the task successfully, then notify Project Management immediately to decide what to do with this information
 - When the agreed time has come, no excuse (except act of God) is good enough for not having successfully concluded the Task: you simply failed your responsibility

What to plan and what not to plan

- We plan tasks that don't get done unless planned
- We do not plan tasks that don't have to be planned to get done. Such planning costs more than it saves
- Account for these tasks as “unplannable tasks”
- Default we allocate $\frac{2}{3}$ for plannable tasks and $\frac{1}{3}$ for unplannable tasks
- We may include tasks in the planning to show that the hours for these tasks are not available for other work
- Plan all plannable hours

Beware of longer Tasks

- Beware of Tasks longer than about 6 hrs
- Estimation is never exact
- If you have 4 or more Tasks in a week, the variation in the Tasks estimations should average



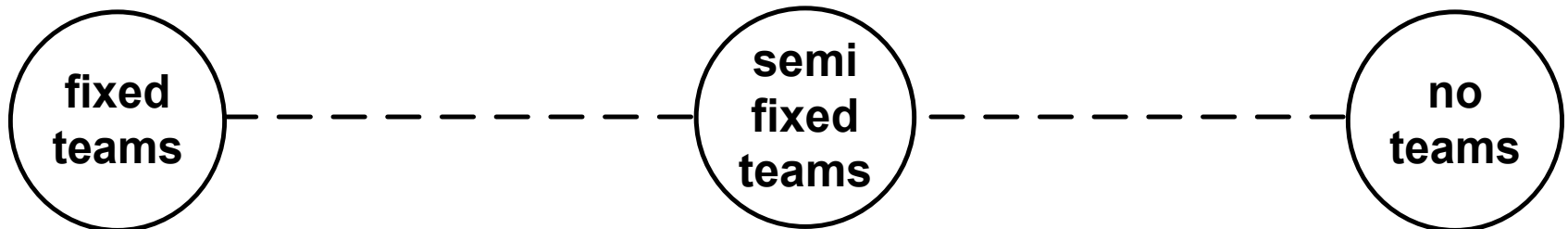
Only the *average* should be OK:
Result is all that counts

- You have only $2/3$ plannable time, so you can cheat a bit to get all the committed tasks done
- May seem contradictory to the TimeBox principle ...

We work on more projects

- Define how many hours available for this project
- Deliver these hours

- **Vision:**



Why TaskCycle?

- **Reflection and Preflection (PDCA - Quick feedback)**
- **Not working on anything less important**
- **Learning to know what to promise**
- **And then living up to our promises**
- **Taking responsibility**
- **Getting the info to be able to carry the responsibility**
- **Coping with interrupts**
- **Active Synchronization**
- **Calibration of estimations on the TimeLine**
- **Taming Parkinson's Law and Student Syndrome**

What to do with the time gained?

If our original requirements are done in 70% of the time, what do we do with the 30% gained?

1. **Choosing the next project**
2. **Continuing evolutionarily adding extras**
3. **Beware of Parkinson's Law!**
4. **Extending the horizon of the project to assure success**

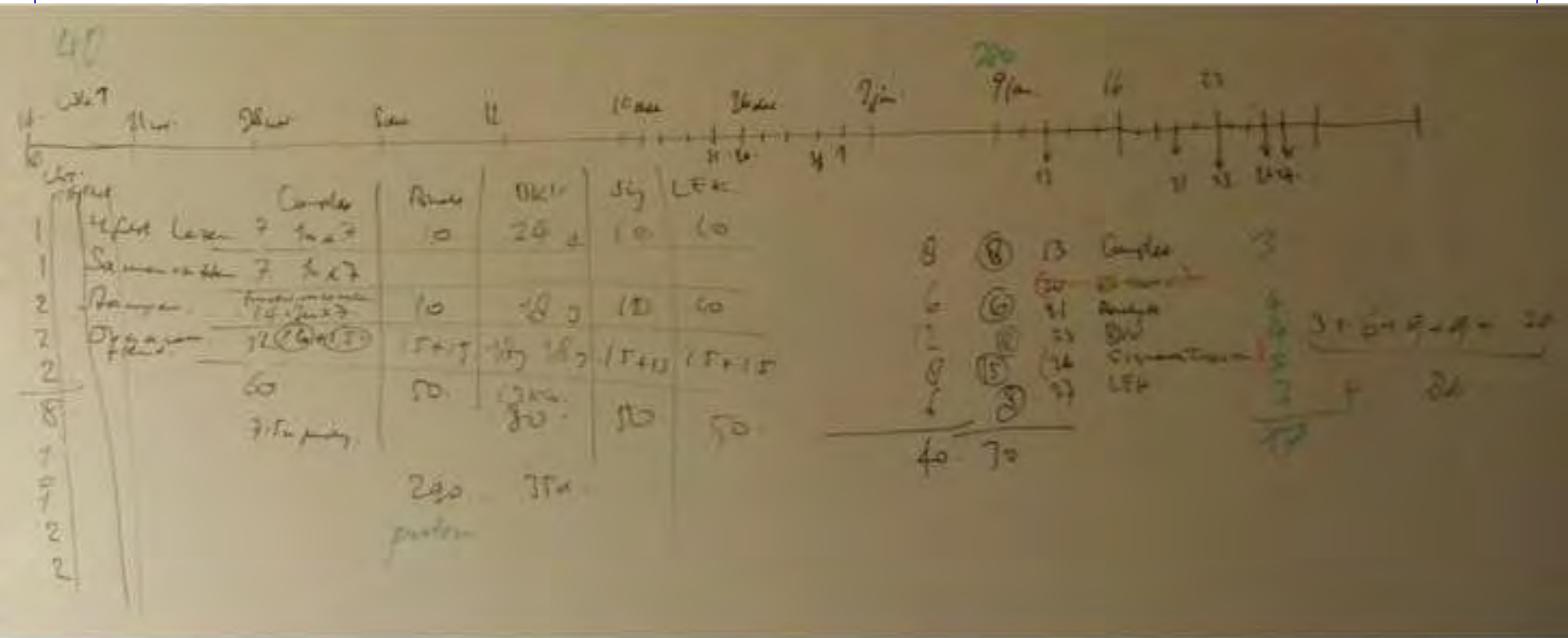
Extending the project horizon to success

- **Many projects end at: Hurray, it works!**
- **If customer success is paying our salaries, shouldn't we make sure success is going to happen**
- **Now a lot of quality requirements suddenly make sense:**
 - **User friendliness - Usability**
 - **Intuitiveness - Learnability**
 - **Installability**
 - **Serviceability - Maintainability**

TimeLine exercise example

- **Preparing for student exams**

What we did



TimeLine exercise for your Project

- **What is the FatalDate, how many weeks left**
- **What is the expected result (←Business Case / Reqs)**
- **What do you have to do to achieve that result**
- **Cut this into chunks and make a list of chunks of activities**
- **Estimate the chunks (in weeks or days)**
- **Calculate number of weeks**
- **Compensate for estimated incompleteness of the list**
- **How many people are available for the work**
 1. **More time needed than available**
 2. **Exactly fit**
 3. **Easily fit**
- **Case 1 and 2: work out the consequence at this level**
- **Case 3: go ahead (but don't waste time!)**

Business Case

Business Case

- **Why are we running a project ?**
- **The new project improves previous performance**
- **Types of improvement:**
 - **Less loss**
 - **More profit**
 - **Doing the same in shorter time**
 - **Doing more in the same time**
 - **Being happier than before**
- **In short: *Adding Value***

Higher Productivity

- **All functionality we produce** *does already exist*
- **The real reason for running our projects is** *creating better performance*
- **Improvement of value, productivity, success, happiness** *for our customers through users*

Improving on *existing* qualities

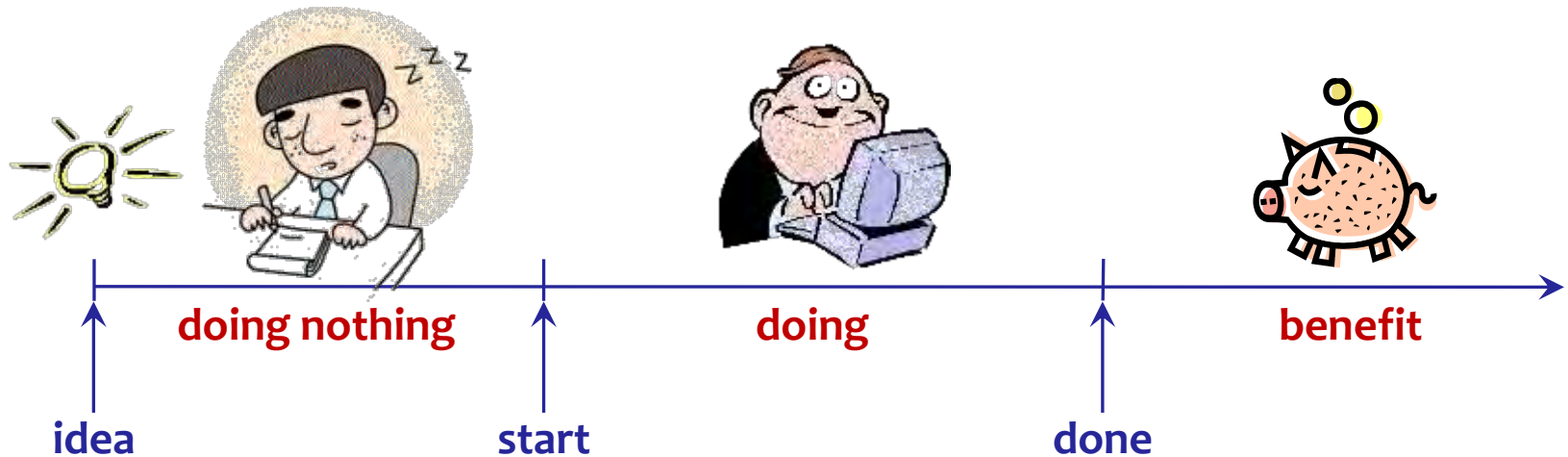
<ul style="list-style-type: none"> • Usability.Productivity: <ul style="list-style-type: none"> • Time to set up a typical specified report • Time to generate a survey • Time to grant access to report, distribute logins to end-users 	V8.5 65 120 80	V9.0 20 0.25 5	min min min
<ul style="list-style-type: none"> • Usability.Intuitiveness: <ul style="list-style-type: none"> • Time for medium experienced programmer to find out how to do ... 	265 15	25.25 5	min min
<ul style="list-style-type: none"> • Capacity.RuntimeConcurrency <ul style="list-style-type: none"> • Max number of concurrent users, click-rate 20 sec, response time < 0.5 sec 	250	6000	users

after FIRM / Gilb 2005

How many Business Cases ?

- **Do you have a Business Case documented for your project ?**
- **How many Business Cases ?**
- **There are usually at least two Business Cases:**
 - **Theirs**
 - **Yours**
- **So, how many Business Cases will there probably be in your project ?**

Nobody's working on the project yet



Return on Investment (RoI)

- + **Benefit of doing** - huge (otherwise other projects would be more rewarding)
- **Cost of doing** - project cost, usually minor compared with other costs
- **Cost of doing nothing** - every day we start later, we finish later
- **Cost of being late** - lost benefit

Business Case

First develop the problem
interdisciplinarily,
then develop the solution
and then the implementation

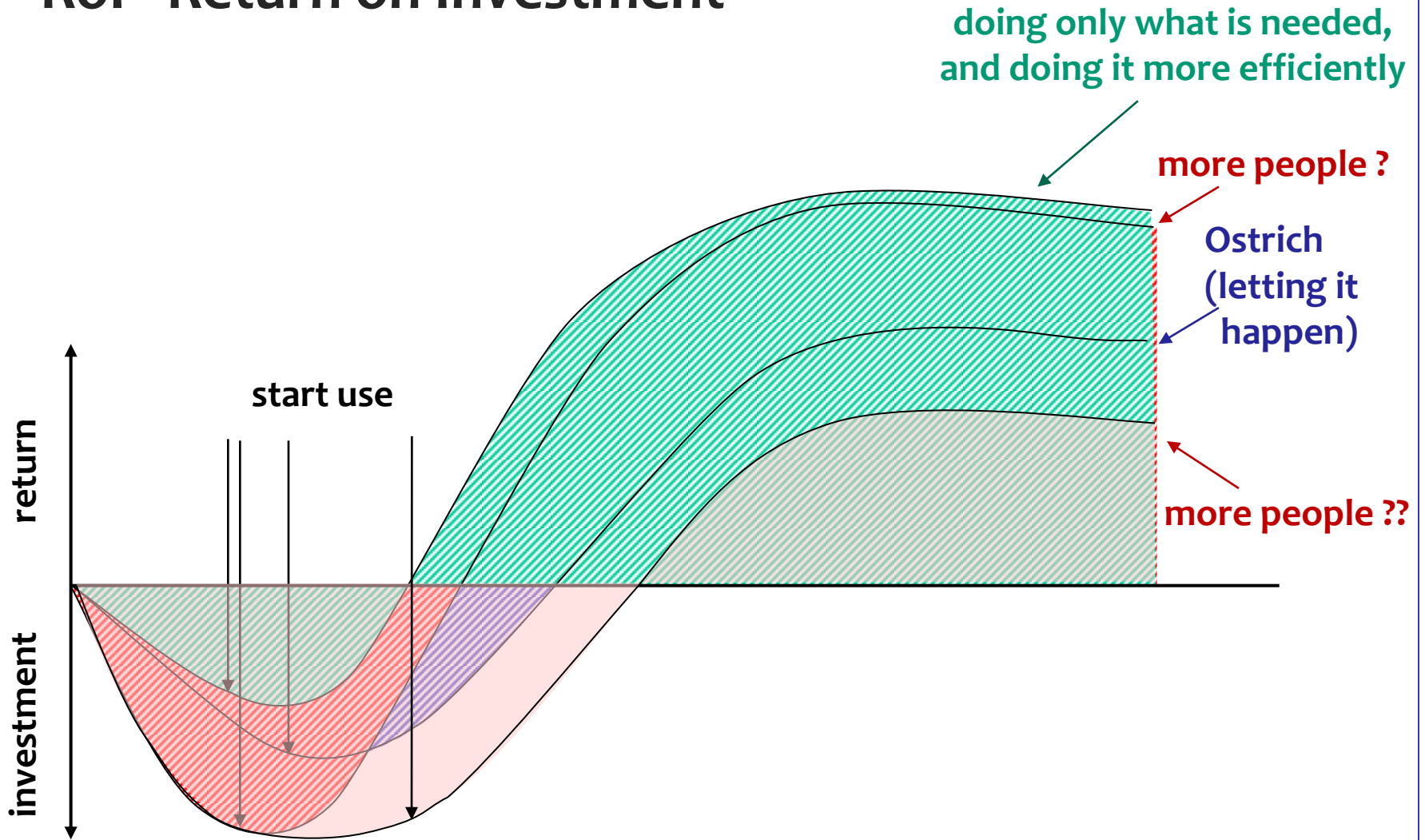
- What to improve and Why
- Used to continually align the Projects progress to the business objectives
- Drives the decision making processes
- Will probably change during the project
- Stakeholders
- Expected Return on Investment (RoI)
Benefit of doing – Cost of doing – Cost of delay – Cost of doing nothing
- Total LifeCycle

Do your projects know
and maintain
the Business Cases ??

0th order approximations

- In the Business Case we often use 0th order estimations
- Order of magnitude
- Better than $0 < \text{guess} < \infty$ (any number is better than no number)
- 0th order is better than no clue
- 1st order is often less accurate than 0th order
- Using two different ways of estimation for crosscheck
- Errors may average if we estimate several pieces

Rol - Return on Investment



Business Case exercise

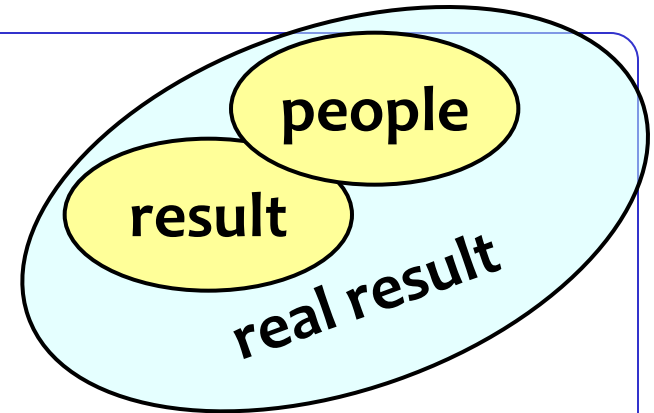
(groups of 2 or 3 people)

Write down a (simplified) Business Case for your current project

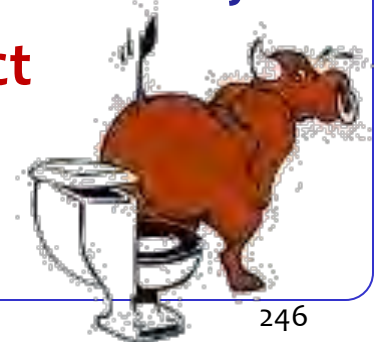
- **What is going to be improved - and what not**
- **Why are we doing this**
- **Who's waiting for it**
- **When do they need it**
- **Expected Return on Investment (RoI)**
Benefit of doing – Cost of doing – Cost of delay – Cost of doing nothing

Stakeholders

Stakeholders are people



- **Every project has some 30 ± 20 Stakeholders**
- **Stakeholders have a stake in the project**
- **The concerns of Stakeholders are often contradictory**
 - Apart from the Customer *they don't pay*
 - So they *have no reason to compromise !*
 - In many cases, finally, *we all pay*
- **Some Stakeholders are victims of the project**
 - They have no reason for the project to succeed, on the contrary
- **Project risks, happening in almost every project**
- **No excuse to fail !**



Victims can be a big Risk



Narita Airport ... ?

The two donkeys:

Competition
or
Cooperation ?



What are the Requirements for a Project ?

- **Requirements are what the Stakeholders require**
but for a project ...
- **Requirements are the set of stakeholder needs that**
the project is *planning to satisfy*
This is what you'll get, if you let us continue
- **The set of Stakeholders doesn't change much**
- **Do you have a checklist of possible Stakeholders ?**

No Stakeholder?

- **No Stakeholder: no requirements**
- **No requirements: nothing to do**
- **No requirements: nothing to test**
- **If you find a requirement without a Stakeholder:**
 - Either the requirement isn't a requirement
 - Or, you haven't determined the Stakeholder yet
- **If you don't know the Stakeholder:**
 - Who's going to pay you for your work?
 - How do you know that you are doing the right thing?
 - When are you ready?

Which stakeholders ?

- **Documentation**
- **Prototypes**
- **Who will be trying to make us fail**
- **What do we need to succeed**
- **Who defines success**
- **How much is enough**

Stakeholder exercise

- **Write down a list of Stakeholders for your project environment**
- **Discuss with neighbour**

Requirements

Top-level Requirement for the Organization

- We must earn a living, and perhaps some profit
- We shouldn't work at a loss

- So:

We should profit from our work

- But:

Customers provide our income

Top level Requirement for any Project

Quality on Time

- **Providing the customer with**
 - what he needs
 - at the time he needs it
 - to be satisfied
 - to be more successful than he was without it
- **Constrained by** (win - win)
 - what the customer can afford
 - what we mutually beneficially and satisfactorily can deliver
 - in a reasonable period of time

Wish Specification

Nice Input

Requirements Compliance ?

The question is not

- to comply with the original Requirements

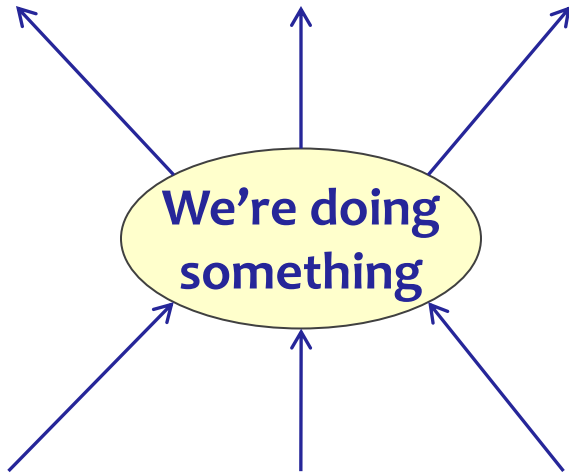
but

- to make it work according to the top level requirement

Watching over the first line of shoulders

Who's waiting for it ?

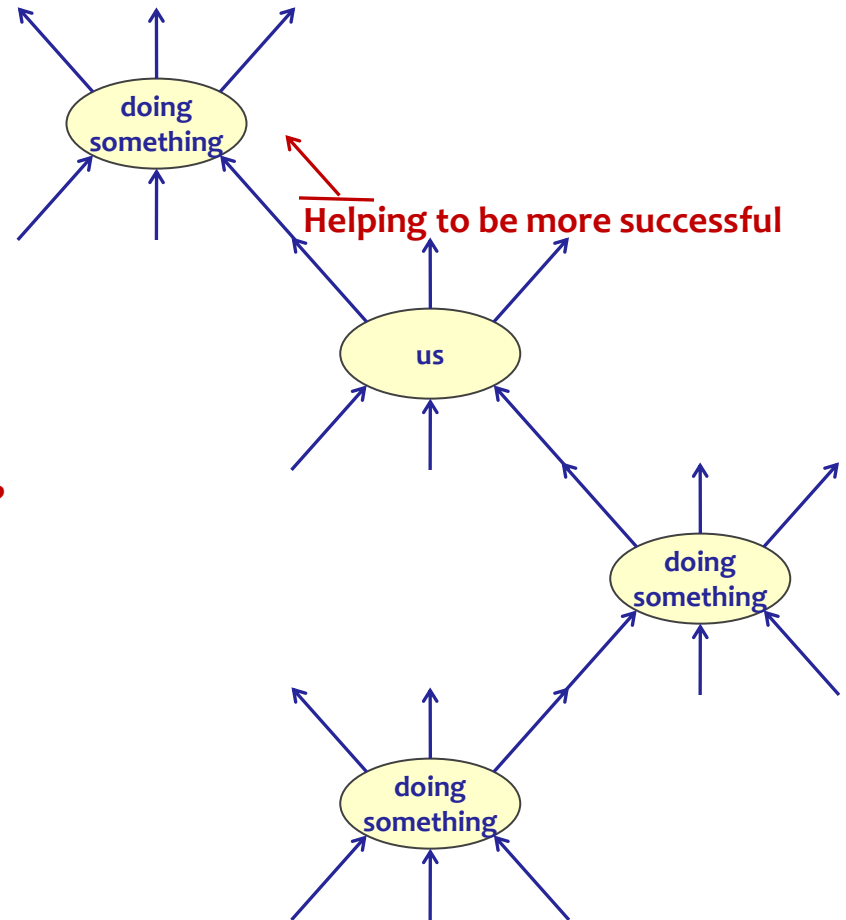
Who's waiting for it ?



↑
For whom ?

↓
Who's help ?

Who can help us to do it better ?



Wish Specification

- **What Wish Specification did you receive ?**
 - **How did you receive it ?**
 - **From whom ?**
 - **What did you do ?**
-
- **Was it complete ?**
 - **Was it clear ?**
 - **Did it reveal the problem to be solved ?**

The Requirements Problem

**The hardest part of building a system is deciding precisely what to build.
No other part of the work so cripples the resulting system if done wrong.
No other part is more difficult to rectify later.**

Fred Brooks, in No Silver Bullet: Essence and Accidents of Software Engineering (1987)

It was a problem in 1987

The Requirements Problem

From time to time I work as an expert witness in software lawsuits. Most of the cases are very similar - the contracts are ambiguous and what the client expects is not what the vendor thought was meant. You can trace the lawsuit back to problems with requirements and project management.

Capers Jones, in Conflict and Litigation Between Software Clients and Developers (1999)

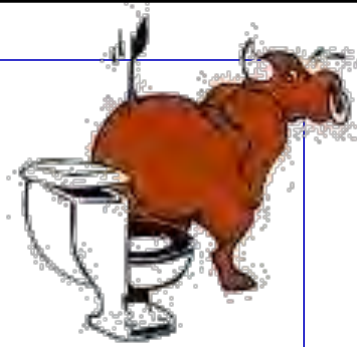
12 years later, not much has changed...

Requirements carved in stone ?

- **We don't know the real requirements**
- **They don't know the real requirements**
- **Together we'll have to find out (stop playing macho!)**
- **What the customer wants he cannot afford**
- **Is what the customer wants what he needs?**
- **People tend to do more than necessary**
(especially if they don't know exactly what to do)

**If time, money, resources are limited,
we should not overrun the budgets**

Fallacy of 'all' requirements



- “We’re done when *all* requirements are implemented”
- Isn’t delivery time a requirement ?
- Requirements are always *contradictory*
- Do we really know the *real* requirements ?
- Who’s requirements are we talking about ?
- Are customers able to define requirements ?
 - Customers specify things they do not need
 - And forget things they do need
 - They’re even less trained in defining requirements than we are
- What we think we have to do should fit the available time
- Use the Business Case

Requirements Compliance ?

The question is not

- to comply with the original Requirements

but

- to make it work according to the top level requirement

Watching over the first line of shoulders

Customer Success

- **Customer**
 - Orders the system
 - Pays for the system

- **Success and failure**
 - *Through* users of the system
 - *More general: through* Stakeholders

Use Cases / Scenarios

- **Used to capture product usage and high level features**
- **Usage data is essential to requirements generation and validation activities**
- **Use cases are easy to read and comprehend**
- **Use cases are *not* the same as requirements**
(Rational/IBM wants us to believe they are)
- **Mis-Use Cases are as important**
 - **20% of the software is there to make the computer do what it should do**
 - **80% of the software is there to make the computer not do what it should not do**
 - **Surely other fields are similar - can you think of examples ?**

Basic Types of Requirements

- **Functional** binary
 - *What the system must do*
 - **Functional Requirements Scope the Project**
 - **Functional requirements are binary** (they're there, or not there)
- **Quality / Performance*** scalar
 - *How much to enhance the performance of the selected functions*
 - **Negotiable: there is always contradiction between requirements**
- **Constraints** binary / scalar
 - *What should we not do, be aware of, be limited by*
 - **There requirements are basically non-negotiable**

* **Better not use *non-functional* requirements !**

Performance Requirements

- **How fast**
- **How big**
- **How nice to see**
- **How nice to use**
- **How accurate**
- **How reliable**
- **How secure**
- **How dependable**
- **How well usable**
- **How well maintainable**
- **How well portable**
- **How well**

Extended ISO Model

Reliability
maturity
fault tolerance
recoverability
availability
degradability

Efficiency
time behavior
resource behavior

Portability
adaptability
installability
conformance
replaceability

Functionality
suitability
accuracy
interoperability
compliance
security
traceability

Usability
understandability
learnability
operability
explicitness
customisability
attractivity
clarity
helpfulness
user-friendliness

Maintainability
analyzability
changeability
stability
testability
manageability
reusability

safety?

real time behavior?

dependability

this-ability?

that-ability?

Constraints

*Constraints are harder
than the other requirements*

- **What it should not do**
- **Budget**
 - Money
 - Time
- **People**
 - You'd want to have the best in your team
 - You'll have to do with what you have. That's the challenge !
- **Standards**
- **Legal**
- **Political**
- **Ethical**

5 times “Why?”

*First develop the problem
interdisciplinarily,
then develop the solution
and then the implementation*

- **Freud and Jung:**
 - Problems are in our sub-consciousness
 - Solutions pop up
 - Solutions are how people tell their problems
- **What’s your problem ?**
 - If there’s no problem, we don’t have to do something
- **Within 5 times “Why?”**
we usually come down to the real problem to solve
 - Otherwise we will be perfectly solving the wrong problem

Requirements exercise

- **What are the Requirements of your current project?**

Exercise:

- **Write down 1 or 2 most important requirements**
- **With Stakeholders** (Who's waiting for it?)
- **Try using 5 times “Why ?”**

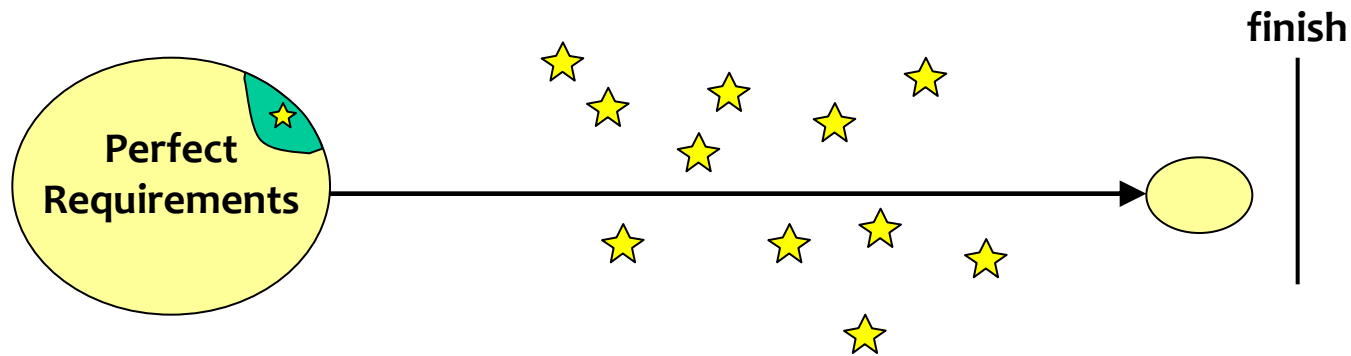
The Requirements Paradox

- Requirements must be stable
- Requirements always change

→ Use a process that can cope with the requirements paradox

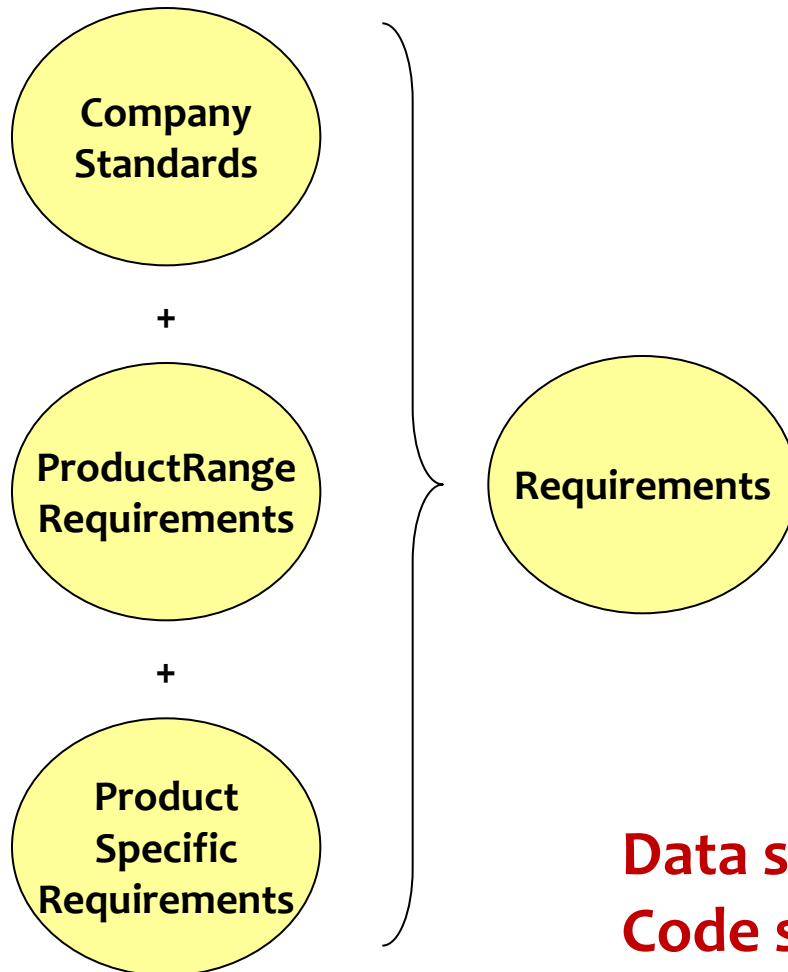
**You cannot foresee every change,
but you can foresee change itself**

The 2nd requirements paradox



- We don't want requirements to change, however,
- Because requirements change now is a *known risk*:
We must *provoke* requirements change *as early as possible*

Requirements should be at one place only



Data should be at one place only
Code should be at one place only

Attributes of a Good Requirement

A Good Requirement is:

Relevant

Complete

Consistent

Unambiguous

Feasible

Clear

Elementary

Concise

Correct

Has no *weak words*

Unique

Verifiable

Traceable

No solution

Does your project have Good Requirements?

Rule

Rule: All quality requirements must be expressed *quantitatively*

Typical requirements found:

- The system should be extremely user-friendly
- The system must work exactly as the predecessor
- The system must be better than before

- It shall be possible to easily extend the system's functionality on a modular basis, to implement specific (e.g. local) functionality

- It shall be reasonably easy to recover the system from failures, e.g. without taking down the power

Lord Kelvin (1824 - 1907)

**When you cannot measure it,
when you cannot express it in numbers,
your knowledge is of a meagre and unsatisfactory kind**

...

**It may be the beginning of knowledge,
but you have scarcely in your thoughts advanced
to the stage of science**

Can we measure everything ?

- How beautiful is music, a painting, a man/woman ?
- Beauty of a car, spacecraft, piece of electronics ?
- What's your weight ?
- How clever are you ?
- Can you quantify 'Love' ?



**Not everything that counts can be measured,
and not everything that can be measured counts**

Einstein

Why quantifying ?

- **The most important things in life cannot be measured, the more important they are, the less you can measure them**

Ron Baker

- **Still**

- The measurement isn't a goal in itself
- Trying to define the measurement provides us with *better understanding* what the problem really is about

Tom Gilb:

The fact that we can set numeric objectives and track them is powerful, but in fact is not the main point. The main purpose of quantification is to force us to think deeply, and debate exactly, what we mean, so that others, later, cannot fail to understand us

How to Measure Anything

Douglas W. Hubbard

Definition of Measurement:

A quantitatively expressed *reduction of uncertainty* based on one or more observations

Expected Value of Perfect Information: **100% reduction**

Expected Value of Information: **actual reduction**

**How to add, subtract, multiply, divide ranges of information:
use Monte Carlo simulation**

Somebody said the requirements should be *SMART*

- Do we have documented requirements ?
- Are they SMART ?

- **S** **Specific**
- **M** **Measurable**
- **A** **Attainable**
- **R** **Realisable**
- **T** **At the right Time (some say: Traceable)**

Requirements with Planguage

ref Tom Gilb

Definition:

RQ27: Speed of Luggage Handling at Airport

Scale: Time between <arrival of airplane> and first luggage on belt

Meter: <measure arrival of airplane>, <measure arrival of first luggage on belt>, calculate difference

Benchmarks (Playing Field):

Past: 2 min [minimum, 2009], 8 min [average, 2009], 83 min [max, 2009]

Current: < 4 min [competitor y, Jan 2010] ← <who said this?>, <Survey Feb2010>

Record: 57 sec [competitor x, Jan 2010]

Wish: < 2 min [2011Q3] ← CEO, 19 Feb 2010, <document ...>

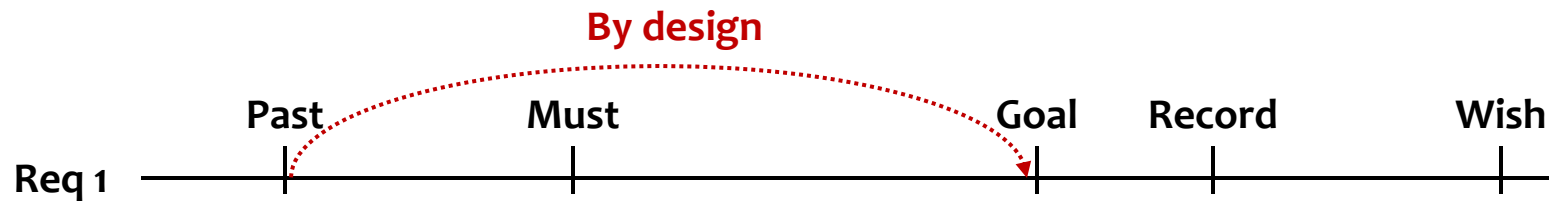
Requirements:

Must: < 10 min [99%, Q4] ← SLA

Must: < 15 min [100%, Q4, Schiphol] ← SLA

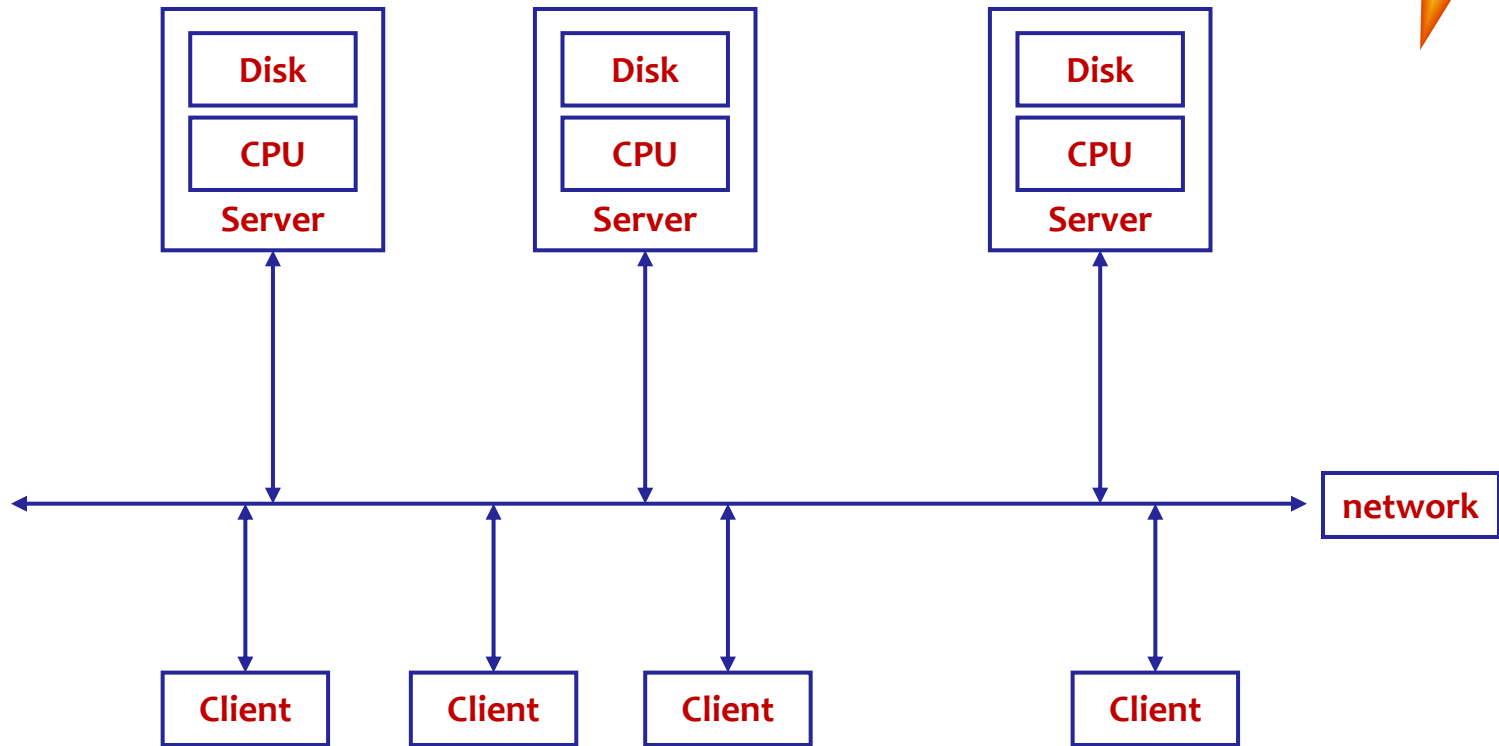
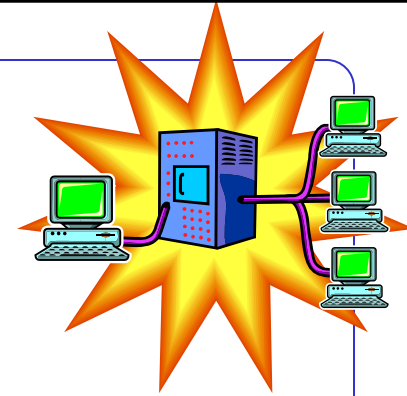
Goal: < 15 min [99%, Q2], < 10 min [99%, Q3], < 5 min [99%, Q4] ← marketing

Design to a Quality Requirement



- **Electronic Hardware Engineering**

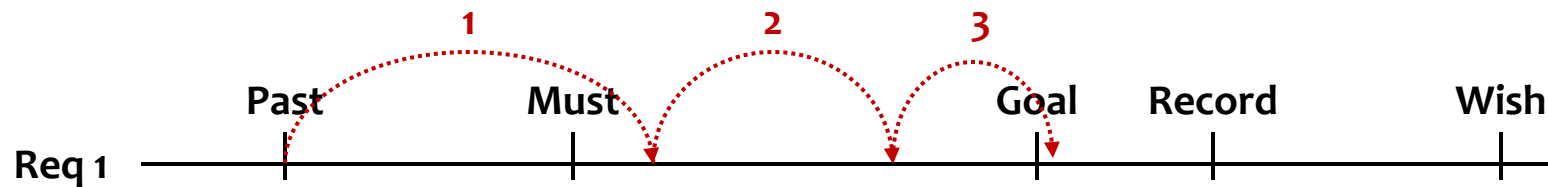
Step-by-step example



Gradually reaching required response time

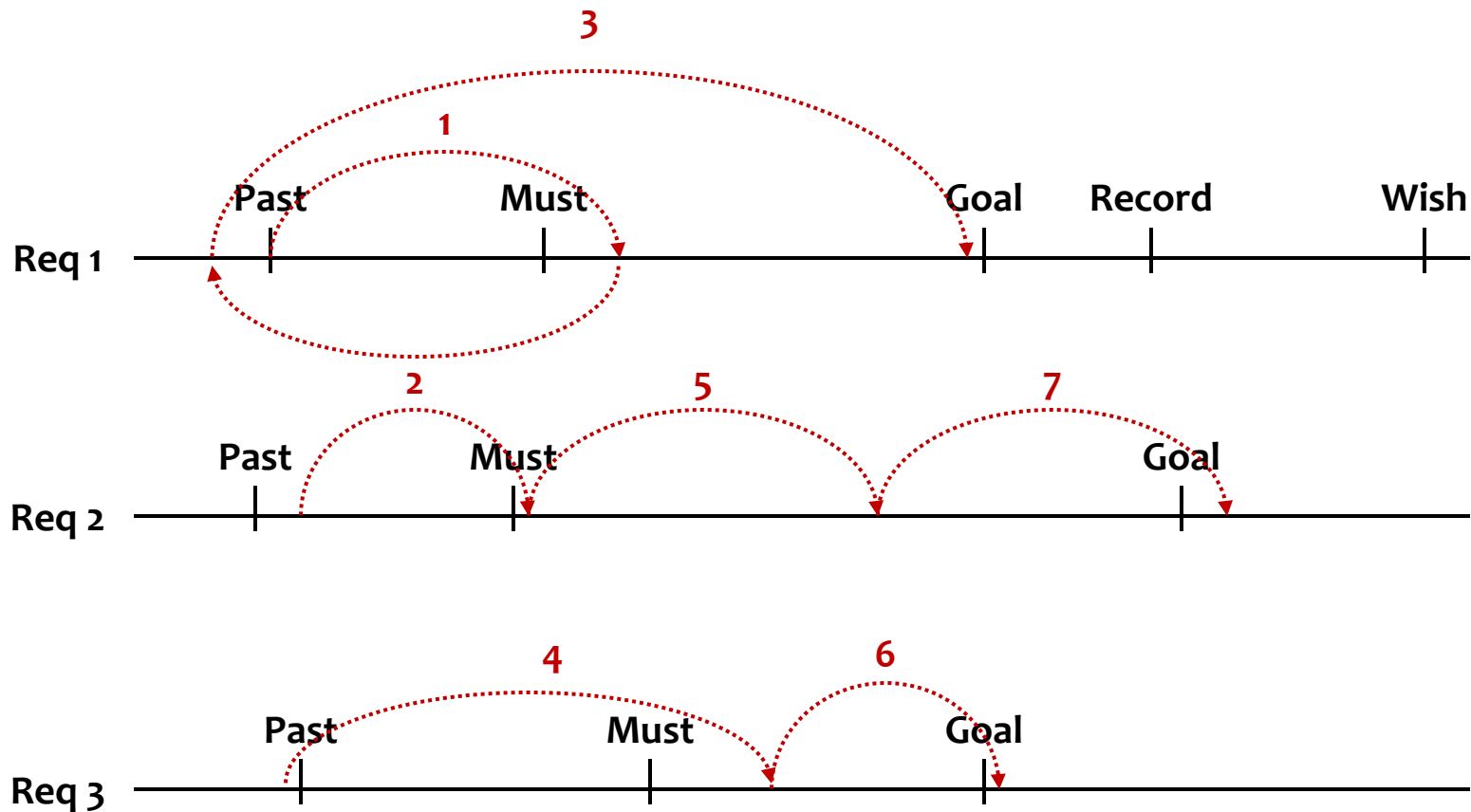
Design to a Quality Requirement

one step at the time



**If the Quality Requirement is composed of several elements,
start with the best ROI**

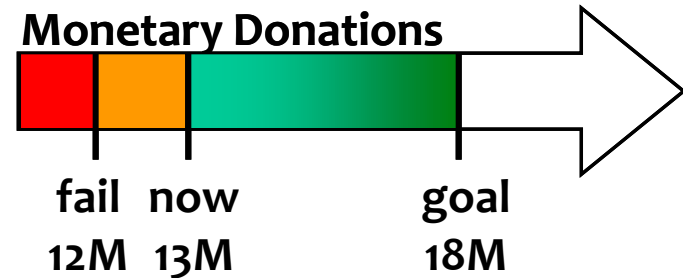
Design to Multidimensional Quality Requirements



Requirements Case

- **Organization collecting online giving for charities**
- **CEO: “Improve website to increase online giving for our ‘customers’ (charities)”**
- **Increasing market share for online giving**
- **Budget: 1M€ - 10 months**
- **Show results fast**

Objective: Monetary Donations



Name Monetary Donations

Scale Euro's donated to non-profits through our website

Meter Monthly Donations Report

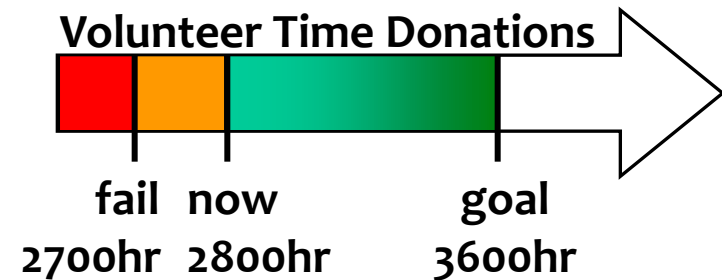
Fail 12M

Now 13M [2008] ← Annual Report 2008

Goal 18M [2009]

Ref Ryan Shriver
ACCU Overload Feb 2009

Objective: Volunteer Time (Natura) Donations



Name Volunteer Time Donations

Scale Hours donated to non-profits through our website

Meter Monthly Donations Report

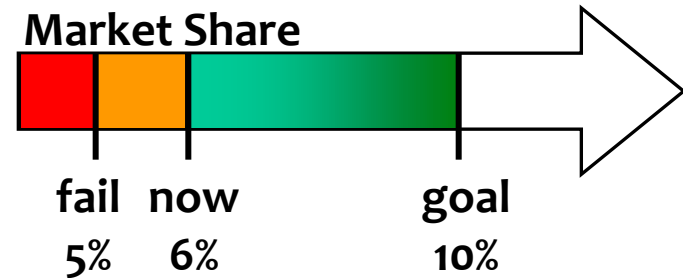
Fail 2700 hr

Now 2800 hr [2008] ← Annual Report 2008

Goal 3600 hr [2009]

Ref Ryan Shriver
ACCU Overload Feb 2009

Goal: Market Share



Name Market Share

Scale Market Share %% online giving

Meter Quarterly Industry Report

Fail 5%

Now 6% [Q1-2009] ← Quarterly Industry Report

Goal 10% [Q1-2010]

Ref Ryan Shriver
ACCU Overload Feb 2009

Priorities are essential

- We don't have the time we'd like to have
- We cannot do the impossible in impossible time, even if we do our best
- To make the best of the available time, we have to do less, without doing too little (not doing what later proves to be unnecessary)
- Possible because people tend to do more than necessary (especially if they don't know exactly what to do)
- Better 80% 100% done, than 100% 80% done
Let it be the most important 80%
- Importance may change all the time:
prioritizing is a constant dynamic process

Impact Estimation example

Impact Estimation	Monthly Donations	Facebook integration	Image & video uploads	Total effect for requirement
€ donations 13M€ → 18M€	80% ±30%	30% ±30%	50% ±20%	160% ±80%
Time donations 2800hr → 3600hr	10% ±10%	50% ±20%	80% ±20%	140% ±50%
Market share 6% → 10%	30% ±20%	30% ±20%	20% ±10%	80% ±50%
Total effect per solution	120% ±60%	110% ±70%	150% ±50%	380% ±180%
Cost - money % of 1M€	30% ±10%	20% ±10%	50% ±20%	100% ±40%
Cost - time % of 10 months	40% ±20%	20% ±10%	50% ±20%	110% ±50%
Total effect / money budget	120/30 = 4 1.5 ... 9	110/20 = 5.5 1.3 ... 18	150/50 = 3 1.4 ... 6.7	
Total effect / time budget	120/40 = 3 1 ... 9	120/20 = 6 1.3 ... 18	120/50 = 2.4 1.4 ... 6.7	

Ref Ryan Shriver - ACCU Overload Feb 2009

Quantified Requirements

Name	Description	Constraint Type	Measure	Current Level	Target Level	Page
Max. Flow Rate	The maximum fuel flow rate	Performance	litres/min.		150	9
Completion Notification	Time from transaction completing to kiosk being informed.	Timing	seconds		5	10
Display Volume Resolution	The amount of fuel dispensed at which the dispenser display should update its volume and price readings.	Performance	ml.		10	11
Flow Sample resolution	The minimum volume of fuel at which the flowmeter must be capable of measuring the flow.	Performance	ml.		5	12
MTBF	Mean time between failure of control system	Reliability	months		12	12
MTRR	Mean time to repair	Reliability	hour		1	13
Service Request Notification	Time taken to notify operator that nozzle has been removed	Timing	seconds		2	14
Start Dispensing	The time between the operator authorising dispensing and fuel being pumped	Timing	seconds		2	15

Examples of Scales

Environmental Noise

dBA at 1.0 meter

System Security

Time required to <break into the system>

Software Maintainability

Time needed from <acceptance of change> to <availability of change>

System Reliability

The Mean Time Before Failure (MTBF) of the system

System Learnability

Average time for <Novices> to become <Proficient> at a defined set of tasks

Productivity

Number of FTE's (Full Time Equivalent)

Examples of Scale Templates

(re-use of Requirements!)

Availability

% of <Time Period> a <System> is <Available> for its <Tasks>

Adaptability

Time needed to <Adapt> a <System> from <Initial State> to <Final State> using <Means>

Usability

Speed for <Users> to <correctly> accomplish <Tasks> when <given Instruction> under <Circumstances>

Reliability

Mean time for a <System> to experience <Failure Type> under <Conditions>

Integrity

Probability for a <System> to <Cope-with> <Attacks> under <Conditions>
Define “Cope-with” = {detect, prevent, capture}

Decomposition of Complex Concepts

- **If you cannot quantify a quality, we call it a *Complex Concept***
- **Decompose complex qualities into elementary ones**
- **Complex ideas may require several scales of measure**

Dependability is a Complex Concept

Dependability.Availability

Readiness for correct service

Scale: % of <TimePeriod> a <System> is <Available> for its <Tasks>

Dependability.Reliability

Continuity of correct service

Scale: Mean time for a <System> to experience <Failure Type> under <Conditions>

Dependability.Safety

No danger, harm, risk

Example: star-system for cars (adult / child, in-car / pedestrian)

Dependability.Security

Free from intrusions (theft, alteration)

Scale: Time required to <break into the system>

Availability



- **Dependability.Availability**
 - Readiness for correct service
 - Scale: % of <TimePeriod> a <System> is <Available> for its <Tasks>
- **Probability that the system will be functioning correctly when it is needed**
- **Examples**
 - (preventive) maintenance may decrease the availability
 - Telephone exchange (no dial tone) < 5 min per year (99.999%)
 - Snow on the road

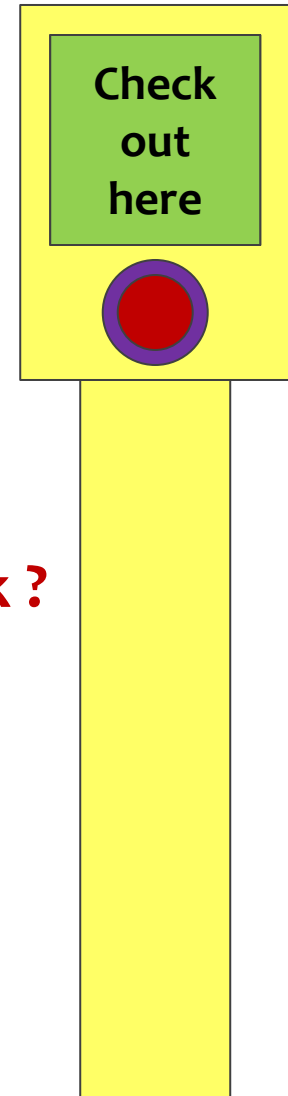
Availability

Availability %	Downtime per year	Downtime per month	Downtime per week	Typical usage
90%	36.5 day	72 hr	16.8 hr	
95%	18.25 day	36 hr	8.4 hr	
98%	7.30 day	14.4 hr	3.36 hr	
99%	3.65 day	7.20 hr	1.68 hr	
99.5%	1.83 day	3.60 hr	50.4 min	
99.8%	17.52 hr	86.23 min	20.16 min	
99.9% (three nines)	8.76 hr	43.2 min	10.1 min	Web server
99.95%	4.38 hr	21.56 min	5.04 min	
99.99% (four nines)	52.6 min	4.32 min	1.01 min	Web shop
99.999% (five nines)	5.26 min	25.9 sec	6.05 sec	Phone network
99.9999% (six nines)	31.5 sec	2.59 sec	0.605 sec	Future network

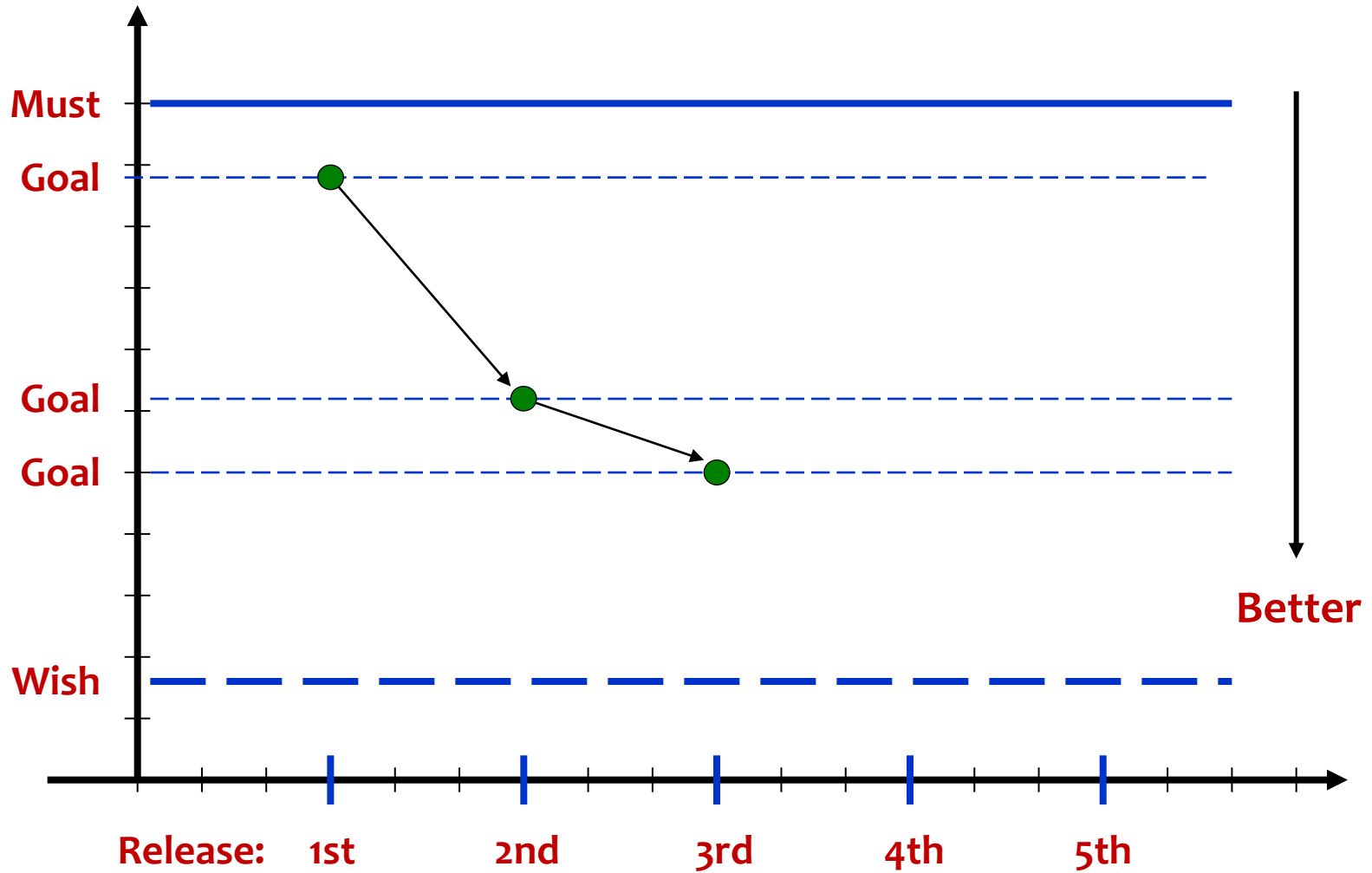
Incredible Public Transport Chip-Card

What would you design the system do if

- **The system is unavailable for it's intended task ?**
- **The server-connection is unavailable ?**
- **... ?**



Portability Goals



Nice things

- **OUT !**

- Isn't paid for
- May not be needed by the customer
- Isn't checked for consistency
- Doesn't get tested
- If the customer finds out, you'll have to support it
- May cause trouble later

- **If it's so important:**

- Make it a change request
- Make the customer pay for the extra (nobody else will)
- Better: decide what less important requirement to discard instead
- We can add any requirement, as long as we also delay a less important one

Example: Road-Pricing in the Netherlands

Realize a road-pricing system in four years

1. Fitting an electronic system in 8 million cars
2. Camera's for number plate recognition
3. Central system for data processing and invoicing
4. Law changes by politicians (tax law, traffic law)
5. Price differentiation for time, place, emissions

Will this succeed?

Requirements exercise:

(groups of 2 or 3 people)

Specify a quality / performance requirement for your current Project, using Planguage

Try to use:

Definition:

- Ambition
- Scale
- Meter
- Stakeholders

Benchmarks:

- Past
- Current
- Record
- (Wish)

Requirements:

- Must/Fail
- Goal

Note: you may end up with a different requirement than you started with ... (what is it really about ?)



Ambition	
Scale	
Meter	
Stakeholders	
Past	
Current	
Record	
Wish	
Must/Fail	
Goal	

Architecture and Design

Design is always a compromise

- Design is the process of collecting and selecting options how to implement the requirements
- The Requirements are *always* conflicting

example:

- Performance 
- Budget (time, money) 

Design and requirements

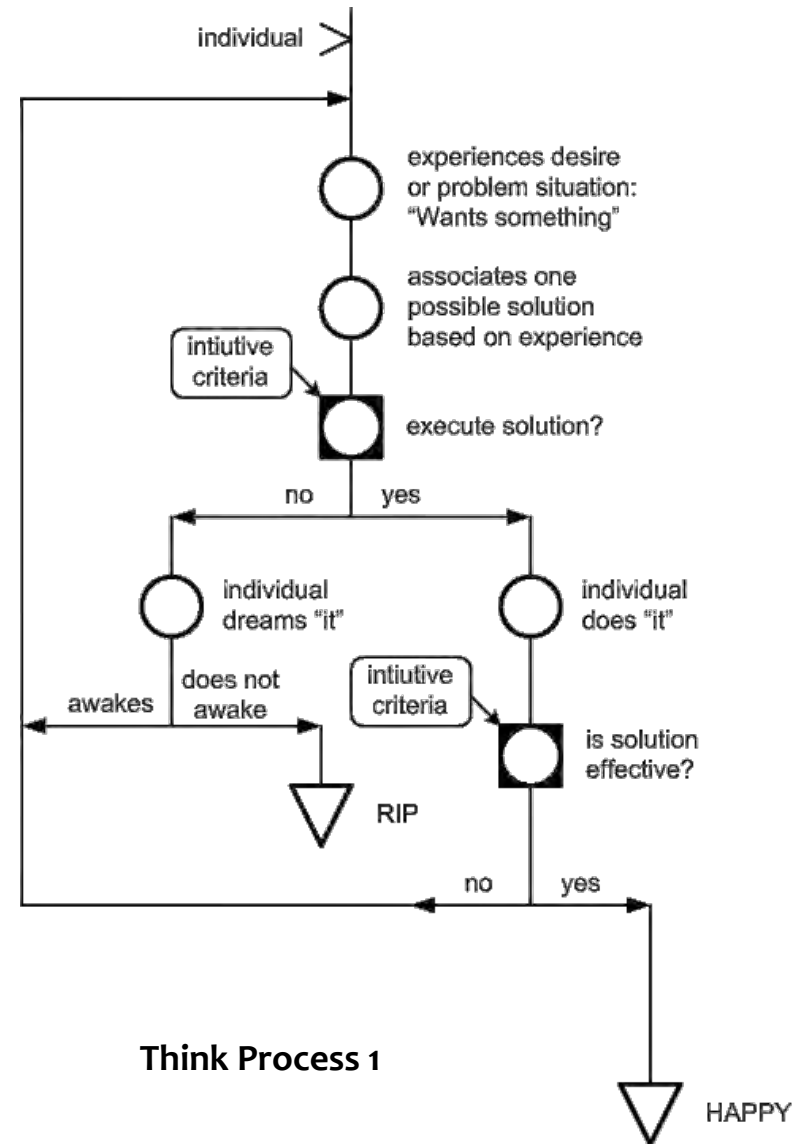
- **Design:**
Finding the best compromise between the conflicting requirements
- **All requirements are equal, but some are more equal than the others**
- **Some aren't really requirements**
- **Some elements will never be used**
- **Some requirements are incorrect**
- **A lot of real requirements are unexplored**

Design Process

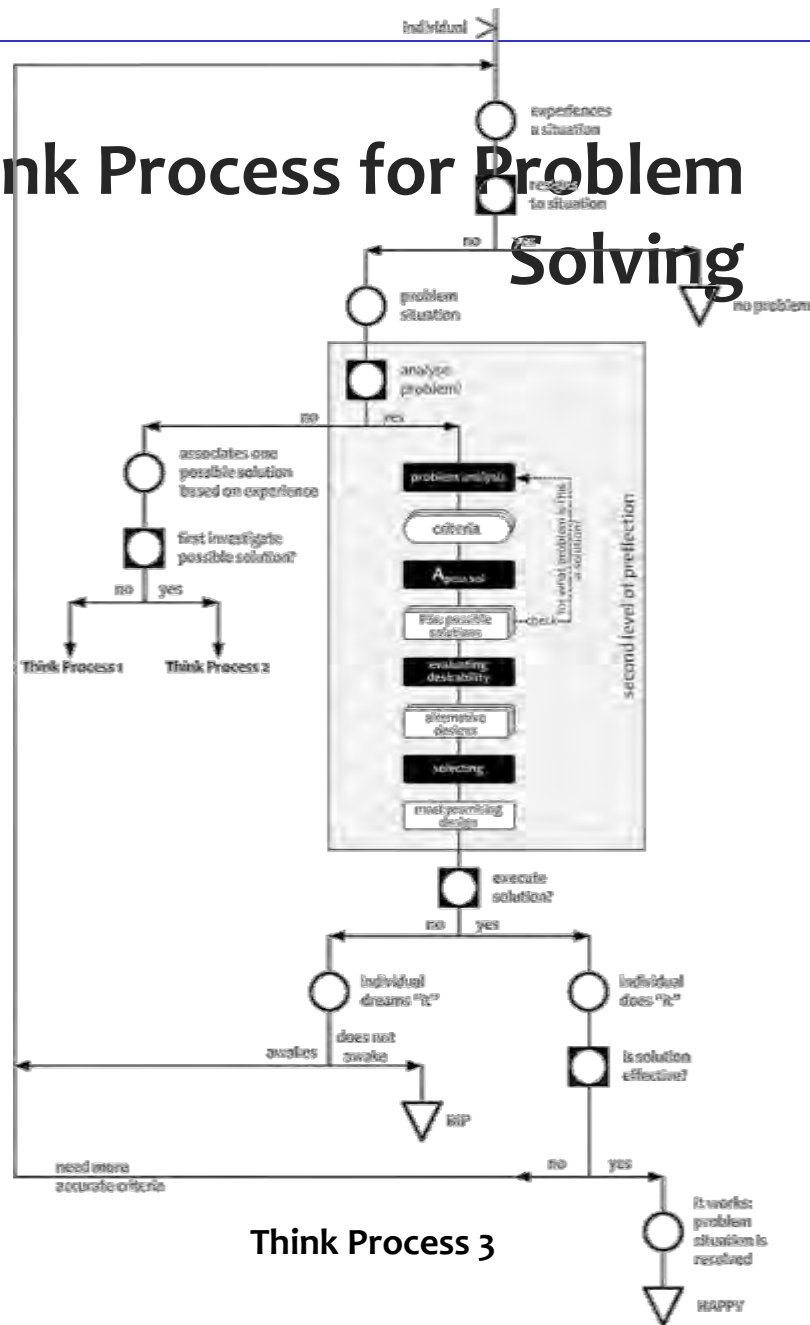
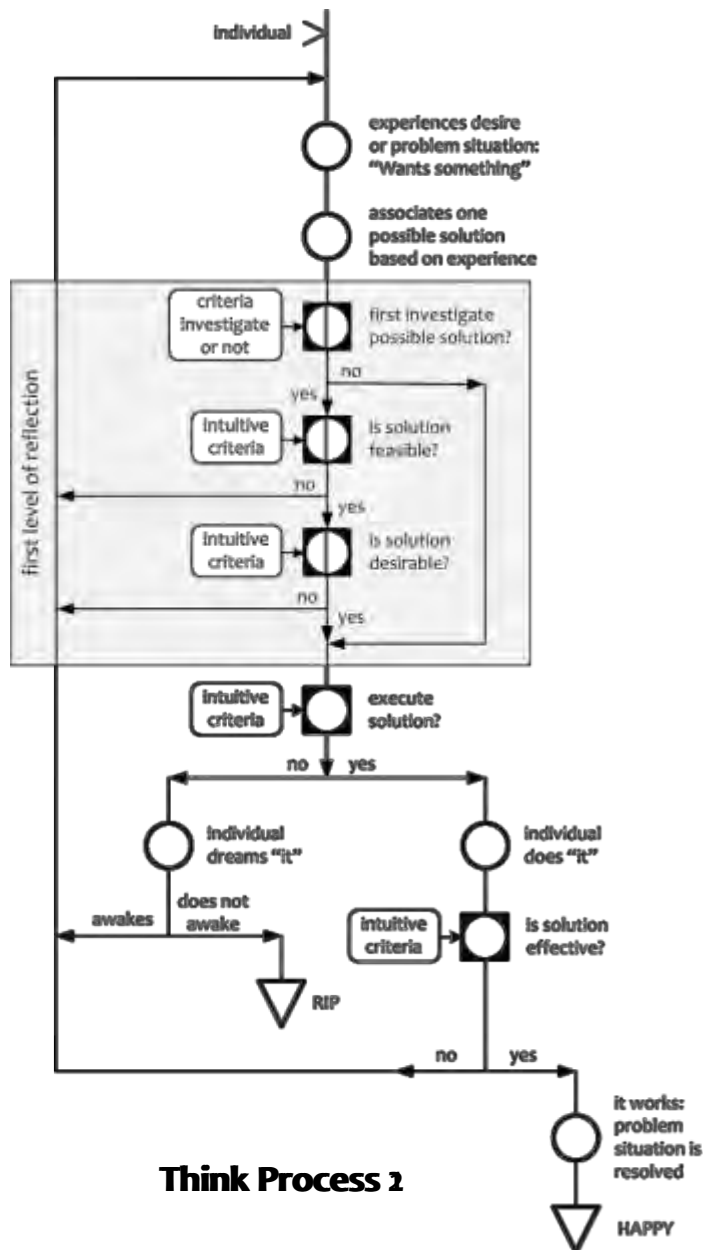
- **Collect obvious design(s)**
- **Search for one non-obvious design**
- **Compare the relative ROI of the designs**
- **Select the best compromise**
- **Describe the selected design**

- **Books:**
 - **Ralph L. Keeyney: Value Focused Thinking**
 - **Gerd Gigerenzer: Simple Heuristics That Make Us Smart**

Think Process for Problem Solving



Think Process for Problem Solving



Ref. Malotaux – Van der Goot

Impact Estimation example

Impact Estimation	Monthly Donations	Facebook integration	Image & video uploads	Total effect for requirement
€€ donations 13M€ → 18M€	80% ±30%	30% ±30%	50% ±20%	160% ±80%
Time donations 2800hr → 3600hr	10% ±10%	50% ±20%	80% ±20%	140% ±50%
Market share 6% → 10%	30% ±20%	30% ±20%	20% ±10%	80% ±50%
Total effect per solution	120% ±60%	110% ±70%	150% ±50%	380% ±180%
Cost - money % of 1M€	30% ±10%	20% ±10%	50% ±20%	100% ±40%
Cost - time % of 10 months	40% ±20%	20% ±10%	50% ±20%	110% ±50%
Total effect / money budget	120/30 = 4 1.5 ... 9	110/20 = 5.5 1.3 ... 18	150/50 = 3 1.4 ... 6.7	
Total effect / time budget	120/40 = 3 1 ... 9	120/20 = 6 1.3 ... 18	120/50 = 2.4 1.4 ... 6.7	

Impact Estimation Example

	<i>On-line Support</i>	<i>On-line Help</i>	<i>Picture Handbook</i>	<i>On-line Help + Access Index</i>
Learning 60 minutes <-> 10 minutes				
Scale Impact	5 min.	10 min.	30 min.	8 min.
Scale Uncertainty	±3 min.	±5 min.	±10 min.	±5 min.
Percentage Impact	110%	100%	60%	104%
Percentage Uncertainty	±6% (3 of 50 minutes)	±10%	±20%?	±10%
Evidence	Project Ajax: 7 minutes	Other Systems	Guess	Other Systems + Guess
Source	Ajax Report, p.6	World Report, p.17	John B	World Report, p.17 + John B
Credibility	0.7	0.8	0.2	0.6
Development Cost	120K	25K	10K	26K
Performance to Cost Ratio	$110/120 = 0.92$	$100/25 = 4.0$	$60/10 = 6.0$	$104/26 = 4.0$
Credibility-adjusted Performance to Cost Ratio (to 1 decimal place)	$0.92 * 0.7 = 0.6$	$4.0 * 0.8 = 3.2$	$6.0 * 0.2 = 1.2$	$4.0 * 0.6 = 2.4$

ref
Tom Gilb
Competitive Engineering

Impact Estimation principle

How much % of what we want to achieve do we achieve by this solution

Possible solutions to achieve it

Could we get all, within the budgets of time and cost ?

At what cost ?

		Design Idea #1	Design Idea #2	Design Idea #3	Total Impact
What to achieve	Objectives	Impact on Objective	Impact on Objective	Impact on Objective	Sum of Impacts on Objectives
Cost to achieve it	Resources Time Money	Impact on Resources	Impact on Resources	Impact on Resources	Sum of Impact on Resources
Return on Investment	Benefits to Cost Ratio	$\frac{\text{Benefits}}{\text{Cost}}$	$\frac{\text{Benefits}}{\text{Cost}}$	$\frac{\text{Benefits}}{\text{Cost}}$	

No Design in the Requirements

MIL-STD-498

- Requirements are what the acquirer *cares enough about* to make *conditions for acceptance* (may be “what” or “how”)
- Design is the set of decisions made by the developer in response to requirements (may be “what” or “how”) (solutions plus decisions)
- Requirement: A characteristic that a system must possess in order to be acceptable for the acquirer

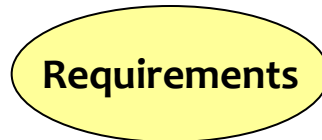
- Design: Solutions plus decisions by the designers
- Specification: This is *what* we are going to make and *how*

My definition for requirements:

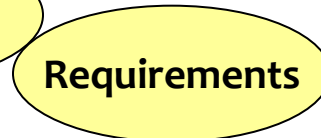
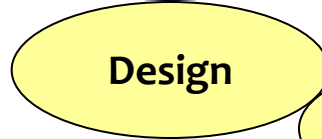
- Requirements are the set of stakeholder needs that a project is planning to satisfy

No Design in the requirements, but ...

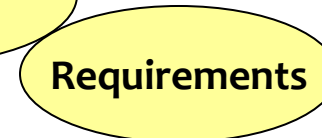
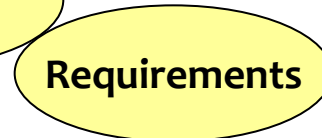
Needs:
what do we need



Options:
how can we do it



Selected solution:
this is how we are going to do it



**Design provides the
Requirements for the next level**

Priorities are essential

- **We don't have the time we'd like to have**
- **We cannot do the impossible in impossible time, even if we do our best**
- **To make the best of the available time, we have to do less, without doing too little (not doing what later proves to be unnecessary)**
- **Possible because people tend to do more than necessary (especially if they don't know exactly what to do)**
- **Better 80% 100% done, than 100% 80% done
Let it be the most important 80%**
- **Importance may change all the time:
prioritizing is a constant dynamic process**

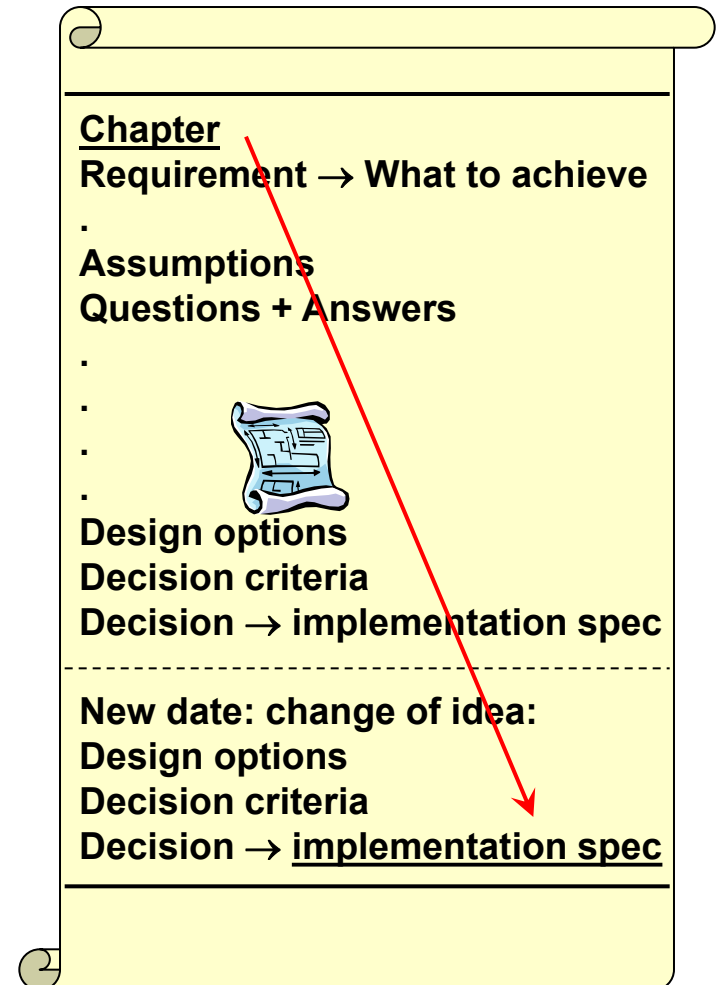
Experiments

- **An Experiment is for finding out how to do something**
- **Results generated in an Experiment *shall be thrown away***
- **We don't want scars in our delivered product/system**
- **Once we know how to do it,
we use that knowledge in the design**
- **The product of development is the design ('pile of paper')**
- **Implementation is a one-to-one translation of the design
into implementation**

DesignLog

(project level)

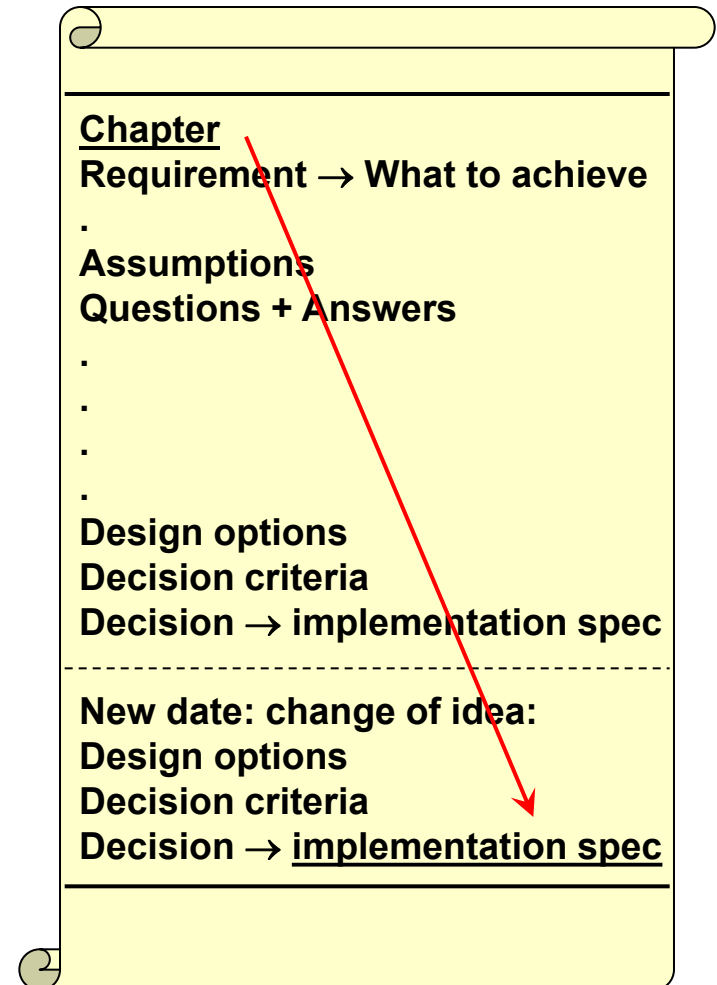
- **In computer, not loose notes, not in e-mails, not handwritten**
 - Text
 - Drawings!
 - On subject order
 - Initially free-format
 - For all to see
- **All concepts contemplated**
 - Requirement
 - Assumptions
 - Questions
 - Available techniques
 - Calculations
 - Choices + reasoning:
 - If rejected: why?
 - If chosen: why?
- **Rejected choices**
- **Final (current) choices**
- **Implementation**



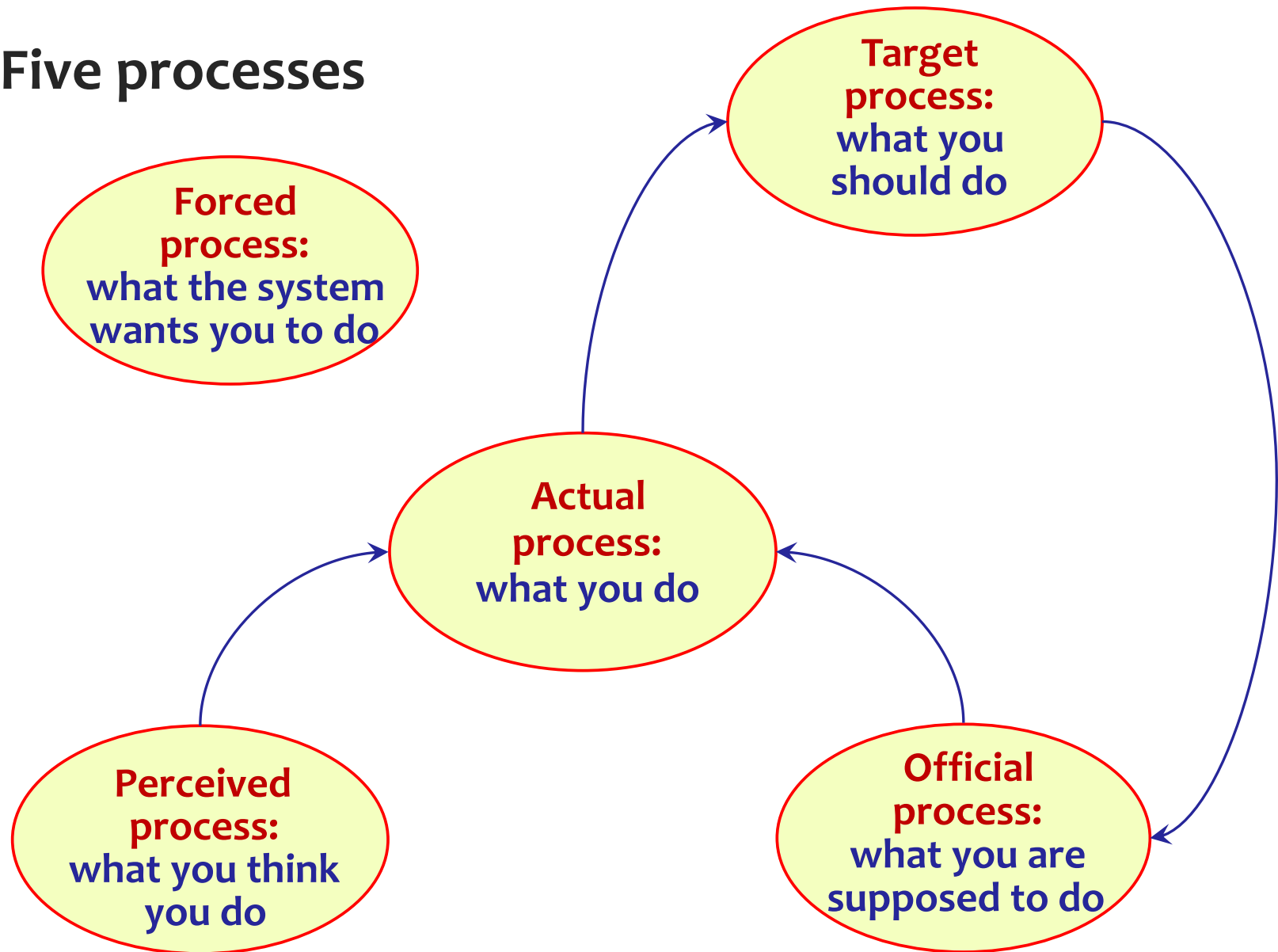
ProcessLog

(department / organization level)

- **In computer, not loose notes, not in e-mails, not handwritten**
 - Text
 - Graphics (drawings)
 - On subject order
 - Initially free-format
 - For all to see
- **All concepts contemplated**
 - Requirement
 - Assumptions
 - Questions
 - Known techniques
 - Choices + reasoning :
 - If rejected: why?
 - If chosen: why?
- **Rejected choices**
- **Final (current) choices**
- **Implementation**



Five processes



Risk

Risk Definition

**An uncertain event or condition that,
if it occurs,
has a negative effect
on a project's objectives**

(PMBOK)

- **0% probability is not a risk**
- **100% probability is an issue or a problem**

Defect and Risk

If a Defect is

a cause of a problem experienced by a stakeholder of the system, ultimately by the customer

then

- **Not satisfying the Goal is a defect**
- **Being late *may* be a defect**
- **Being over budget *may* be a defect**

and Risk is

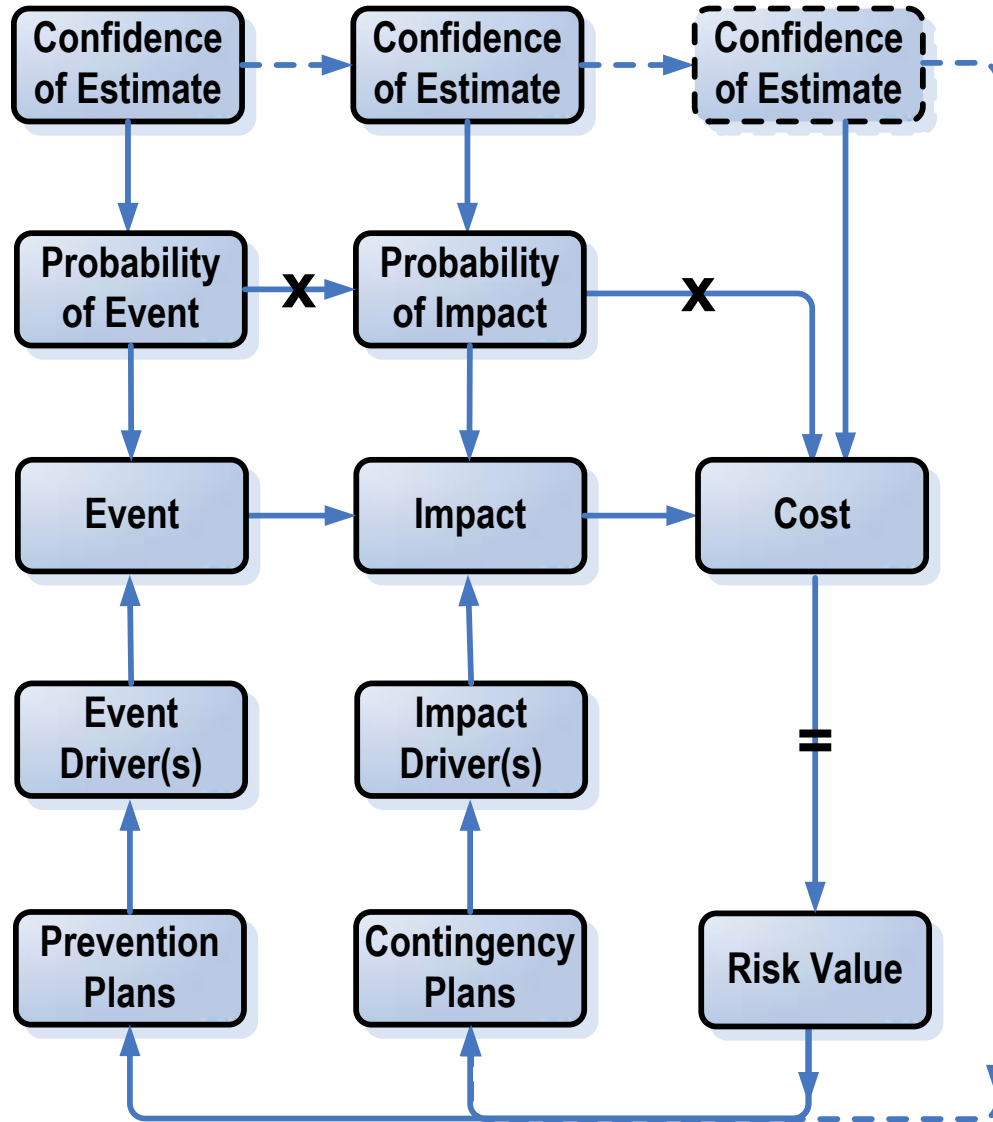
an event that may cause a defect

Our own Risk

Getting less profit than expected

- Takes more time to develop
- Costs more to develop
- Operating cost more than expected
- Performance less than expected
- Guarantee
- Contract liability
- Legal liability
- Claims

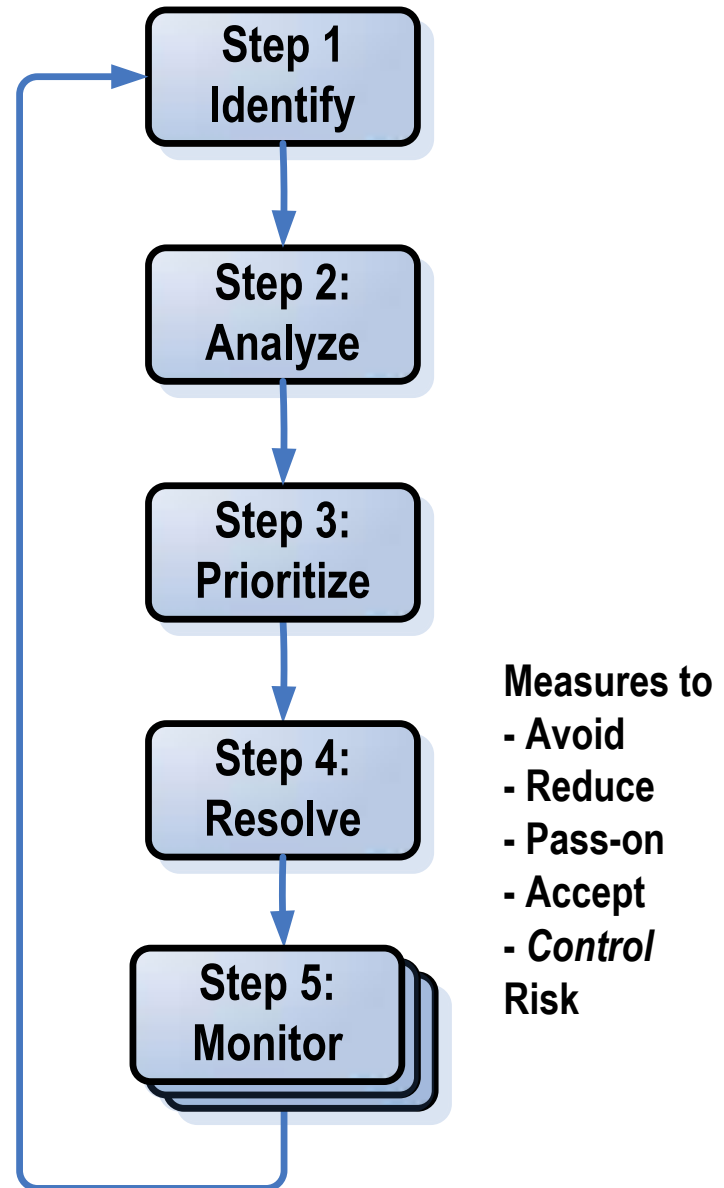
Risk Model



worst case ?

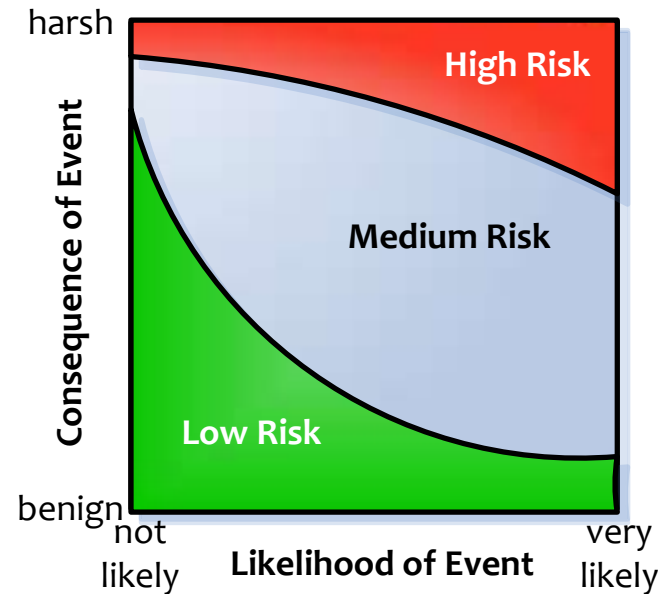
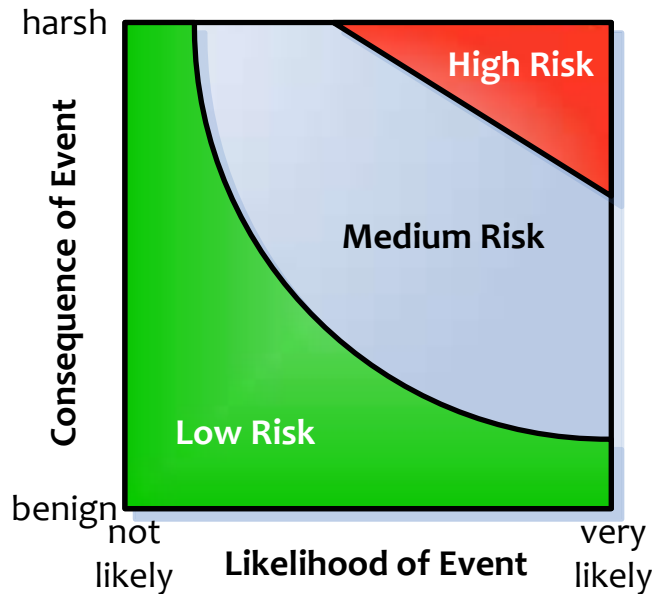
$$V_R = P_e * P_i * C$$

Risk Management



Prioritize Risk?

ref. INCOSE SE Handbook



Risk Priority = Likelihood x Consequence ??

Mathematical Risk Management can be risky

		From SBS (Program constraints)							WFA evaluation	
		Staff	Budget	Facilities	Type of contract	Restrictions / Dependencies	Customer	Subcontractor	ΣR	WFA order
From WBS	Develop Project Charter			I=3, p=2; R=5	I=6, p=3; R=10	I=7, p=7; R=14	I=7, p=5; R=15		110	1
	Define scope	I=3, p=6; R=18	I=7, p=4; R=28					I=4, p=4; R=16	82	3
	Develop Resource Plan	I=7, p=5; R=15				I=4, p=3; R=12			47	5
	Develop Communication Plan	I=5, p=6; R=15				I=3, p=2; R=6			21	8
	Develop Risk Plan	I=7, p=5; R=15							35	7
	Develop Change Control Plan								0	8
	Develop Quality Plan					I=4, p=3; R=12			12	10
	Develop Purchase Plan								0	12
	Develop Cost Plan		I=3, p=4; R=12						32	9
	Develop Organization Plan								0	12
	Develop Project Schedule								0	12
	Conduct Kickoff meeting	I=7, p=5; R=15			I=3, p=2; R=6				41	8
	Weekly Status Meeting								0	12
	Monthly Tactical Meeting								0	12
	Project Closing meeting						I=3, p=2; R=6		6	11
	Standards	I=3, p=6; R=18					I=7, p=5; R=15	I=4, p=4; R=16	80	2
Program Office					I=3, p=3; R=15	I=3, p=6; R=18		55	4	
Risky events evaluation	ΣR	100	60	6	35	94	124	51		
	Risky events order	100	4	7	5	3	2	8		

Exhibit 3 - Matrix KBM for a software development with a cardinal scale approach

ref
Carlo Rafele,
David Hillson,
Sabrina Grimaldi

Checklists for brainstorm

- **Human risk**
 - In the project
 - After the project
- **Technical risk**
 - Can we make it
 - Will it survive
- **Environmental risk**
 - Example: CE
- **Regulatory risk**
 - Example: CE
- **Consequential risk**
- ...

**Each of these can have
it's own checklist
to trigger the recognition
of real risks**

Project Management *is* Risk Management

- Don't set Risk Management apart
- Call it by the proper names:
 - Requirements
 - Planning
 - Design
 - etc
- Risk principles are quite simple
- Implementation as found in literature is vague
- Remember Murphy's Law

Murphy's Law

Whatever can go wrong, will go wrong

Should we accept fate?

Murphy's Law for Engineers:

Whatever can go wrong, will go wrong ...

Therefore:

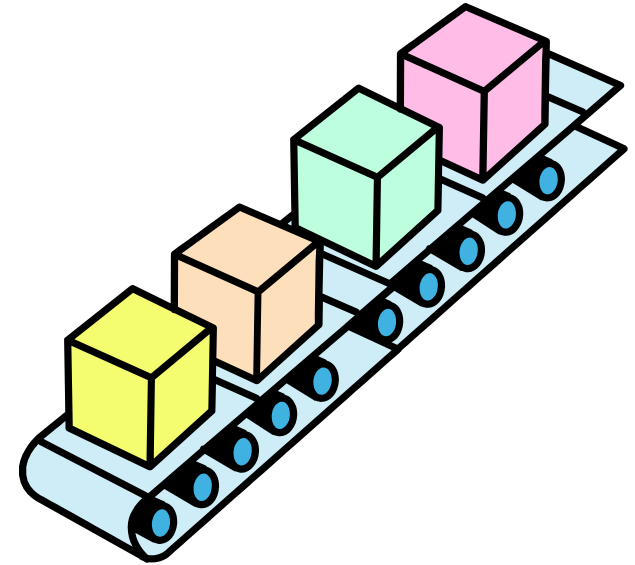
We should actively check all possibilities that can go wrong and make sure that they cannot happen

What are Risks in your Projects?

- ...
- ...
- ...

- **Are these really Risks?**
- **0% probability is not a Risk**
- **100% probability is not a Risk**

Controlling Risk by design



- **Every project is unique**
(otherwise it's production)

however

- **A lot is always the same:**
 - Every project is done by people
 - No project is very much unique
 - There are many similarities (known risks)
 - So, a lot is predictable
 - We know the Requirements will change (but don't know which)
 - Engineers control risks by design (= engineering)

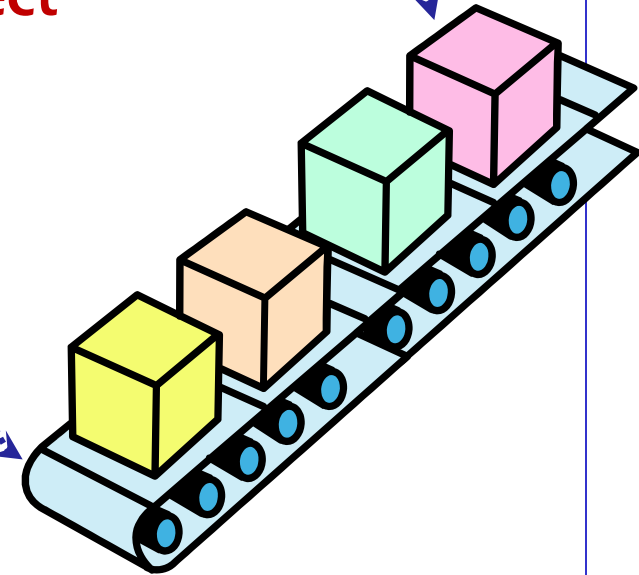
Many known risks are hardly risks

- Most of the *real* risks are in the product
- Most of the *known* risks are in the project

$$V_{\text{Risk}} = P_{\text{event}} * P_{\text{impact}} * C$$

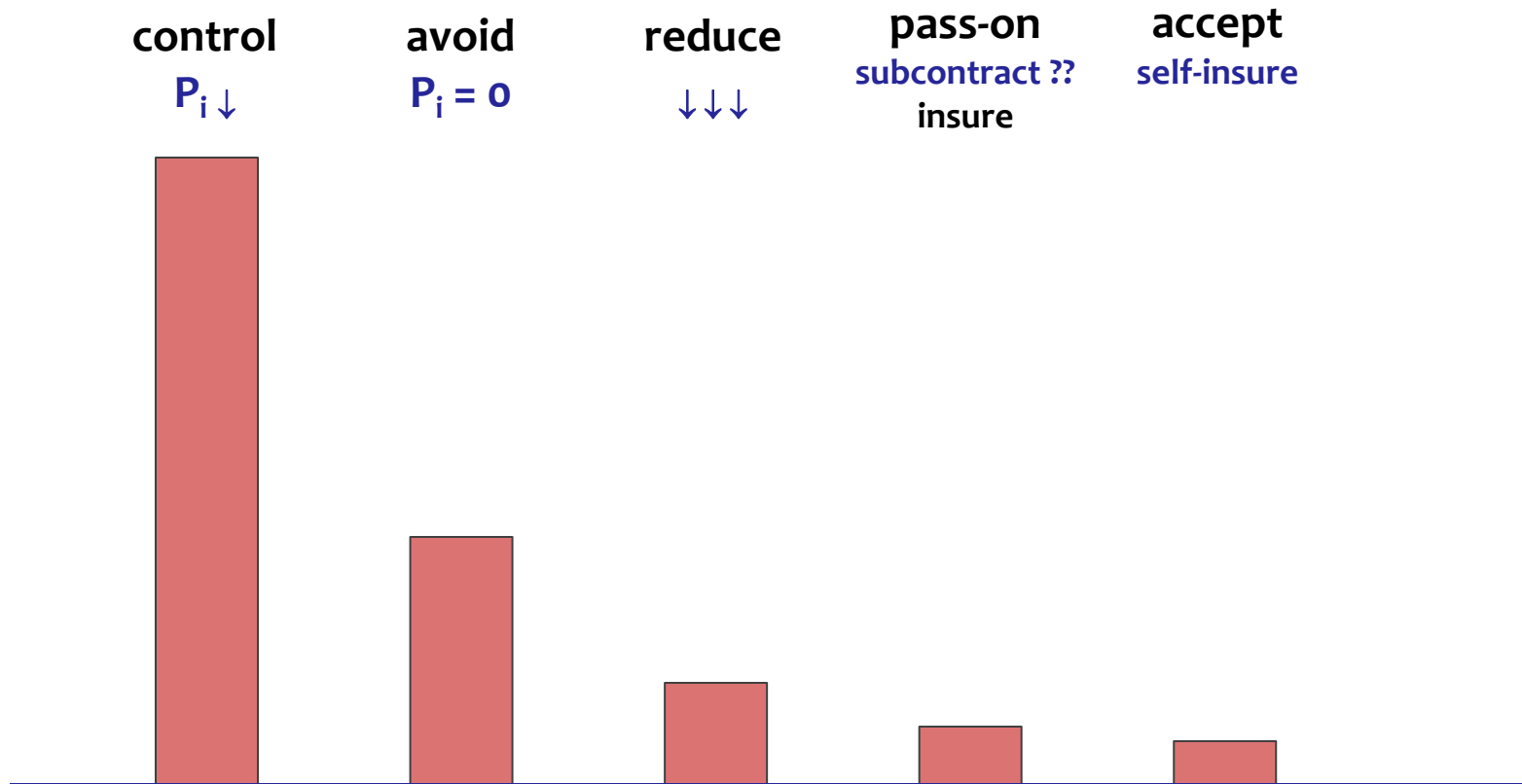
$P_{\text{event}} = 1$
 $P_{\text{impact}} \rightarrow 0$

- We don't only *design* the product,
- We also *design* the project
- If we control 80% of the risks *by design*
- We have more time to handle the 20% real risks



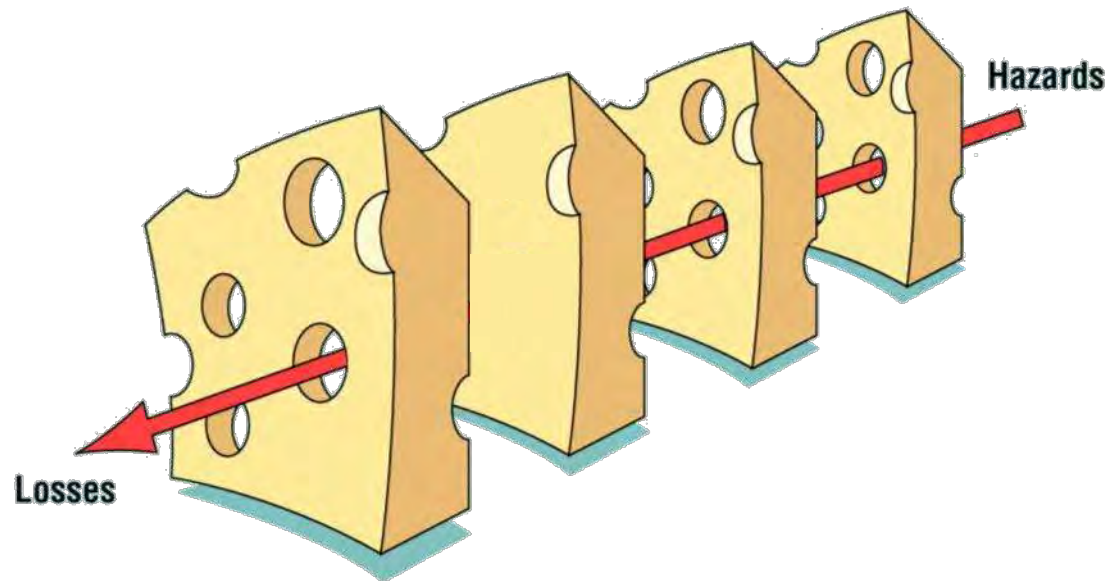
Risk mitigation

$$V_{Risk} = P_{event} * P_{impact} * Cost$$



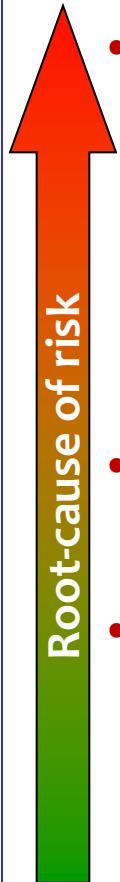
Swiss Cheese model

ref James Reason



Can we add some cheese from Holland?

Product Risks



- **Development**
 - Requirements errors
 - Incorrect Assumptions
 - Design errors
 - Calculation errors
 - Implementation errors
- **Maintenance**
 - Incorrect or insufficient maintenance
- **Use**
 - Operator errors
 - User errors
 - Victims
 - Technical errors

**All these risks are
introduced by humans
People like you and me**

How do Evo processes deal with Risk ?

- **Delivering the wrong result**
- **Delivering at the wrong time**
- **Not making the customer happy and more successful**
- **Promising more than we can do**
- **Doing the wrong things for too long**
- **Trying to do more than we can**
- **Making more mistakes than necessary (fatigue)**
- **Coping with suppliers beyond our control**
- **Gold Plating (doing more than needed)**
- **Interrupts: losing time on *seemingly* important things**

Evo Processes

- **Evo Planning**
 - Risk of delivering at the wrong time
 - Risk of delivering late
 - Risk of delivering unnecessary things
- **Evo Requirements Management**
 - Risk of delivering the wrong things
 - Risk of delivering unnecessary things
- **Evo Design process**
 - Selecting the best compromise for the contradicting requirements
- **Pro-active Synchronization**
 - Risk of others causing us to fail
- **Evo Interrupt process**
 - Risk of losing time on seemingly important things

Development time is limited

- **Having not enough development time is a safety risk**
- **Working overtime is a safety risk**
- **We must use the *limited available time well***
- **Therefore we should plan well**
 - What do we have to do?
 - What can we do?
- **Taking the consequence**
 - (People tend to do more than necessary, especially if they don't know exactly what to do)
 - What are we going to do?
 - What are we *not* going to do? (saving time!)
- **Carefully select the most important work**
- **Finding out as quickly as possible whether we are doing the right things**

What do we do to stay on time ?

- **Short term:**

- Many short cycles: continual result and opportunity to adjust
- Changing from optimistic to realistic estimation
- Estimating hours, not days
- Estimation is a TimeBox
- Focus on Value Delivery

- **Longer term**

- TimeLine
- Continuous stakeholder consequences / agreement

- TaskSheet (first think, then do)

What do we do to stay on budget ?

- **Budget is constraint in time, money**
- **Clearly define how much time and money are available:
Hard limit !**
- **If we don't get a limit, we'll create our own (BudgetBox)**
- **As quickly as possible finding out what is really needed**
- **If it doesn't fit the budgets, it won't be done (unless ...)**

How to deliver Quality

- **Early and frequent deliveries to check requirements and assumptions: only doing the right things**
- **Stakeholder identification en regular contact to find out together what is really needed**

Mastering unclearness

- **Analysis tasks:**

- What do I know now
- What do I still not know
- What do I still have to find out
- Defining and estimating Tasks
- Tight TimeBox

- **Result of analysis task:**

- What's the benefit ?
- What's the cost ?
- How important is it (where does it fit in) ?

Mastering supplier risks

- **Active synchronisation**
- **Showing the importance of FatalDates**
- **Proactively solving problems**
- **Foreseeing delays and doing something about it, before it's too late**

Mastering client risks

- **Are they capable to receive the result ?**
- **Are they ready to do something with the result ?**
- **Is the client system ready ?**

Frequent deliveries solve these problems

When is the project done ?

- **Using exit criteria**
- **Goal levels met**
- **Making sure the users will become more successful**
 - If the users won't be more successful, they're not going to generate our salaries

- **Risk management:**
 - Making sure our salaries can and will be paid

Remaining Risks

- **Risks beyond our control → Complex project**
 - Within or beyond control of client
 - Within or beyond control of suppliers
 - **Example: Satellite launching date moving**
 - Extra costs for maintaining project support
 - Extra costs of users waiting, having prepared using the satellite
 - Meanwhile new requirements may emerge
 - **Example: Delays caused by compulsory purchase procedures**
 - **Weather**
 - **Traffic Jams ?**

Personnel Shortfalls

Boehm 1991

- **There are a certain number of people in the organization**
- **If we don't get the people we think we need, they are working on more profitable activities**
- **Using TimeLine, we inform management about the consequences**

- **This is not risk - it's choice**

Unrealistic schedules and budgets

Boehm 1991

- How can we speak about realistic schedules if the requirements will change anyway?
- If the requirements aren't clear (which they almost never are), any schedule will do
- If the time/cost budgets are insufficient to get a profit, we shouldn't even start or continue
- If management/customers insist on unrealistic schedules (Check), they may need education (Act), or they want us to fail
- People can quickly learn to change from optimistic to realistic estimators and thus live up to their promises
- We continuously update the TimeLine to predict what we will get, what not and what we may get
 - Using "Earned Value" for calibration (reflection)
 - And "Value Still to Earn" for prediction (preflection)

Developing the wrong product

Boehm 1991

- **Why do we have Requirements?**
- **We don't know the real Requirements**
- **They don't know the real Requirements**
- **The circumstances change**

- **First develop the problem, then the solution**
- **Without feedback we probably are developing the wrong product**
- **Rapid feedback is used to quickly learn about the real Requirements and which assumptions are wrong**

Developing the wrong user interface

Boehm 1991

- **The goal is making the customer satisfied and more successful than he already was**
- **If the users don't become more productive we fail**
- **We don't want to fail**
- **So we quickly find out what the right user interface should be**

Gold plating

Boehm 1991

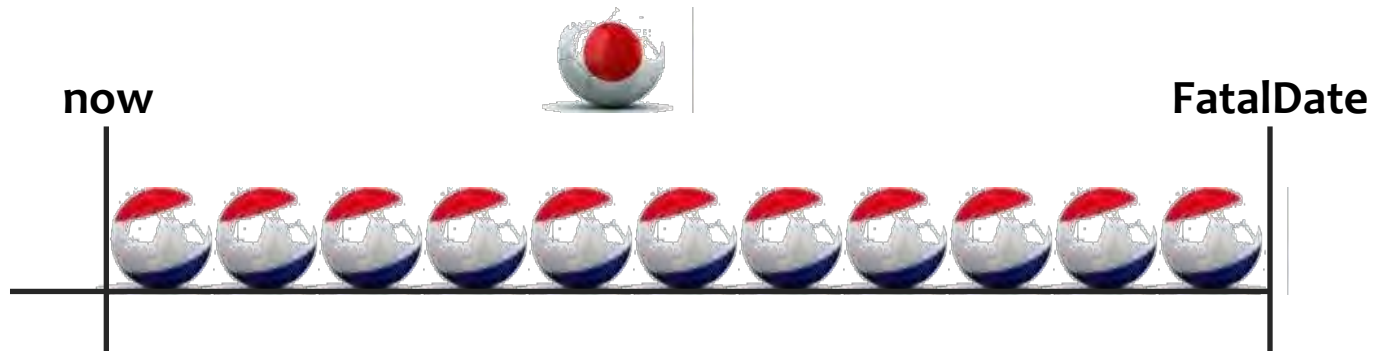
- We do as little as possible at every step
- We specify Must and Goal values
- When we reach the Goal value, we are done
- People tend to do more than necessary, especially if it is not clear what should be done
- So we define what should be done and *what not*
- Not so easy for technical people
- Developing the problem first provides *focus*
- We call doing more than needed: *a hobby*

Continuing stream of Requirements changes Boehm 1991

- **Requirements do change because**
 - We learn
 - They learn
 - The circumstances change
- **If we deliver according to obsoleted requirements, we don't create customer success**
- **We know that requirements will change, so we have to find out quickly which will change, therefore**
- **We even *provoke* requirements change as quickly as possible**

If we add something ...

If we add something, something else will not be done



Problems with externally furnished components

Boehm 1991

- **If our FatalDate has come, we have no excuse**
- **We use Active Synchronization to stay on top**

Real time performance shortfalls

Boehm 1991

- This is why we have Performance Requirements
- Then we use *engineering* techniques to make sure the system is according to the requirements
- Real time *should* and *can* be predictable
(using the right architecture !)

Managers ignorance

- The product has to generate income
- If management impede the workers to produce the product in the most optimal way ...
- Management usually is not stupid
- But if you don't supply the right facts ...

- The boss *may* mess up the Result,
if he's the owner of the company
- All the others have the option to leave

Worst risk

- **The worst risk is the one we forgot**
 - It's within our control, but we didn't see it before it happened
 - It's beyond our control, but we saw it too late and/or we didn't react appropriately
 - The trick is to be ahead of any problem, before it occurs
 - Don't ostrich: actively take your head out of the sand !
 - Don't keep it for yourself !
 - If anybody complains, we're too late
- **Be paranoid, be proactive !**
- **If we control 80% of the risks *by design*, we have a lot more time to address the remaining 20%**

Risk exercise

- **Select a project**
- **What do you want to achieve ?**
- **What can impede achieving it ?**
- **What can you do about it ?**
- **What will you do about what ?**

Verification & Validation

Testing

QA

Do you ever make a mistake?

- People make mistakes
- We are people

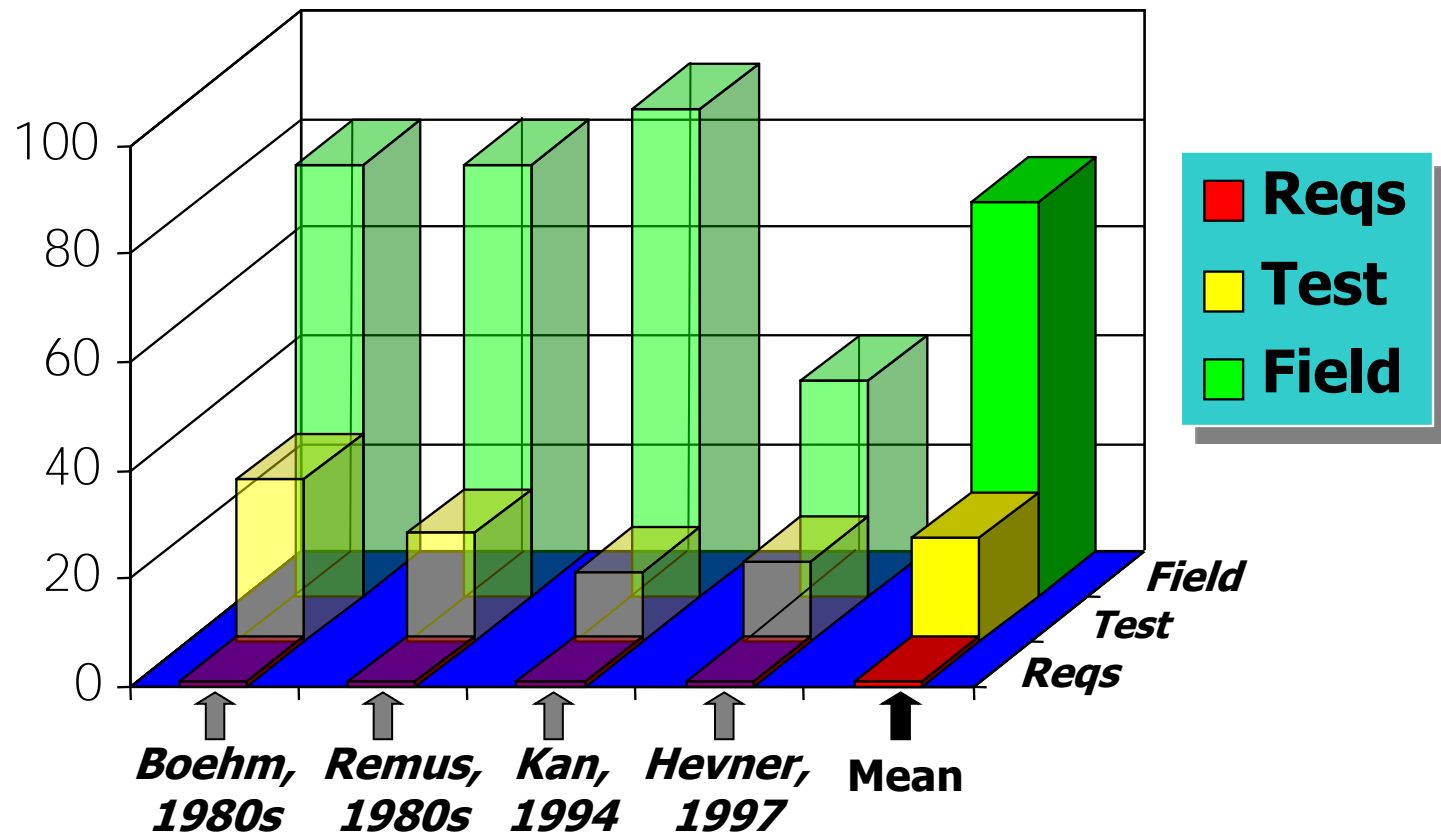
*If we think we are done
there are still defects*

Costs of defects

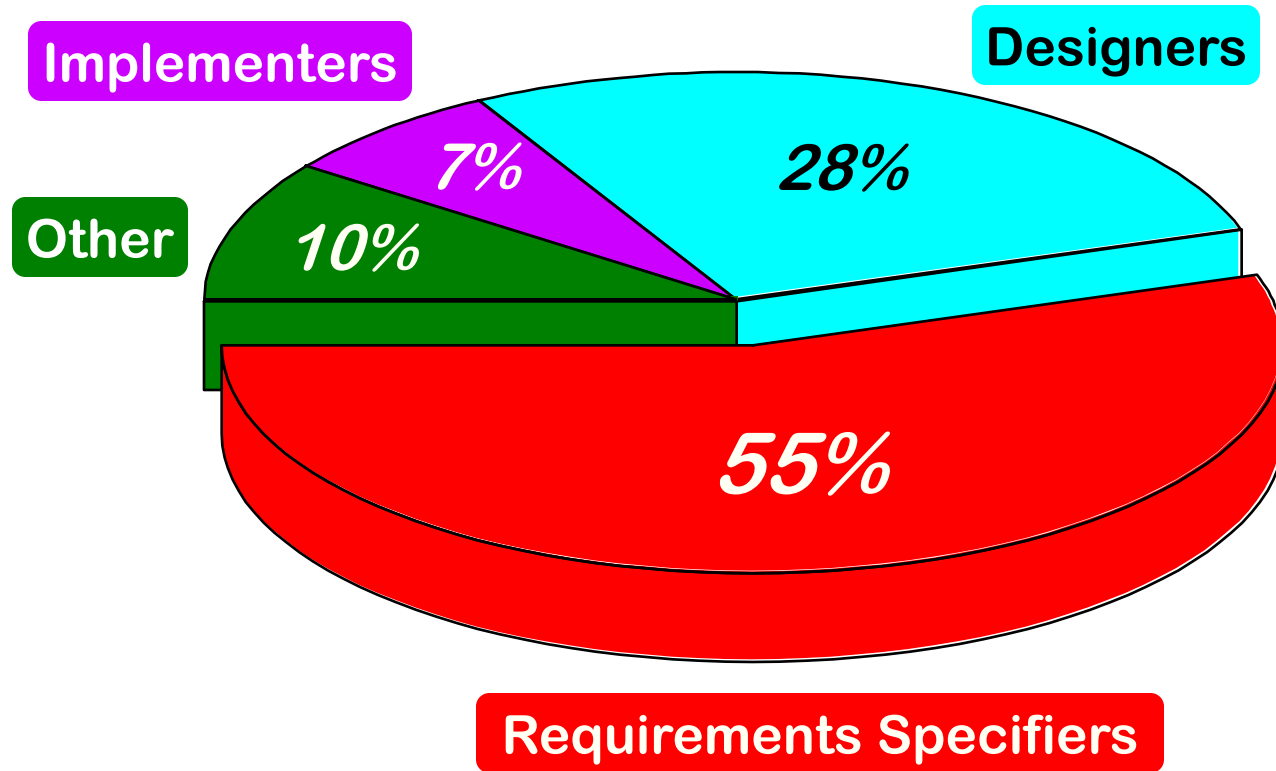
**The longer a defect stays in the system,
the more it costs to find and repair**

Cost of Requirements Defects

The longer a defect stays in the system, the more it costs to repair

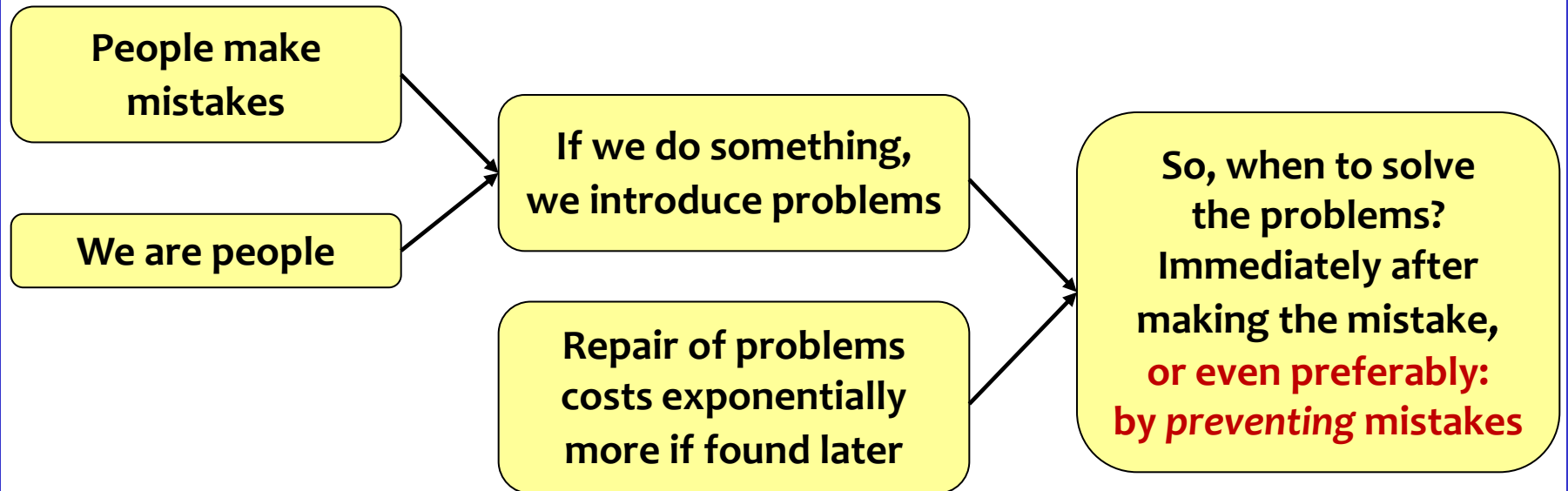


Typical Defect Injectors (cost breakdown)



After Bender Associates, 1996

Inevitable consequence

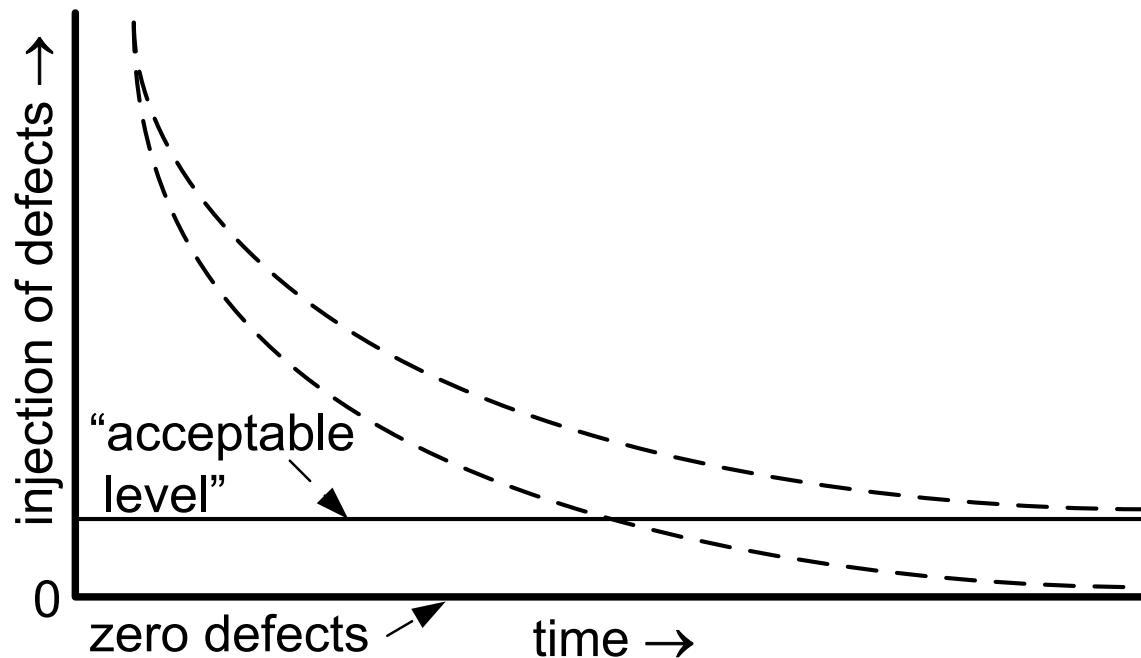


Do we deliver Zero Defect products ?

- **What do you think is acceptable ?**

Is Zero Defects possible ?

- **Zero Defects is an asymptote**

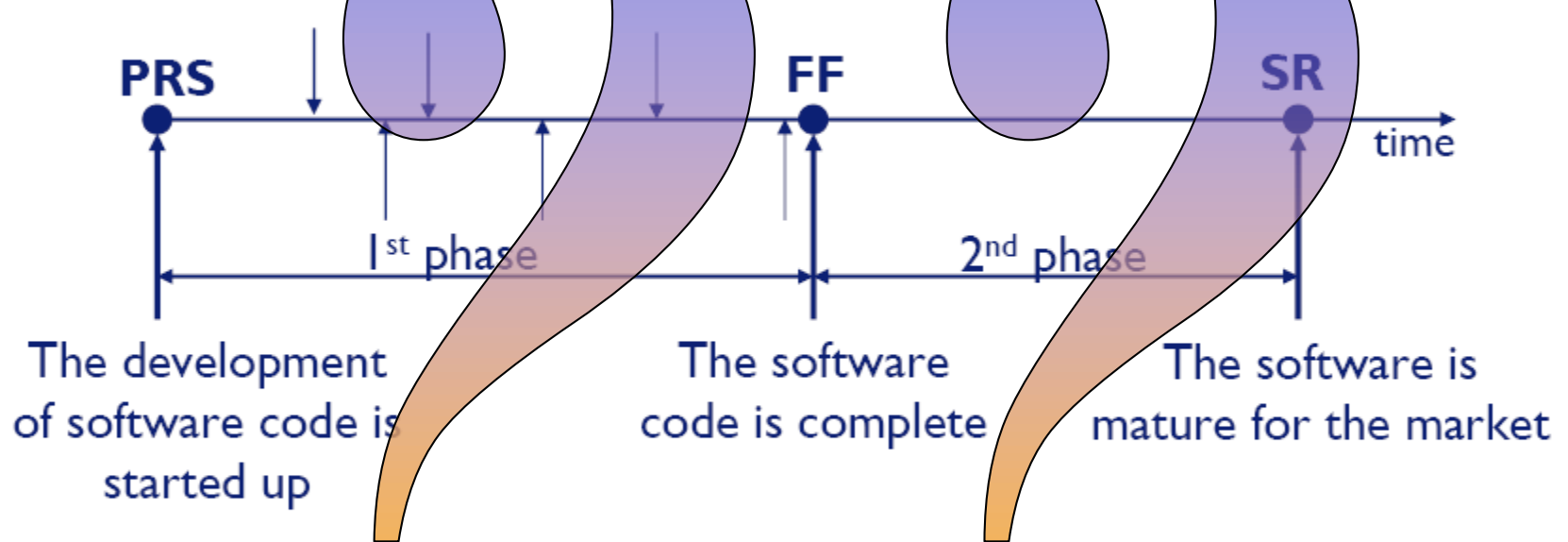


- **When Philip Crosby started with Zero Defects in 1961, errors dropped by 40% almost immediately**

The process of defect injection and detection

- **Conventional (software) development:**
 - Development phase: inject bugs
 - Debugging or Testing phase: find bugs and fix bugs
- **Can't we do better, or are we already doing things better ?**
- **Real Engineering is doing (most) things First Time Right**
(that's why engineers have a full curriculum)

Software development process



- 1st phase is developing phase
- 2nd phase is error-bugging phase

The Problem

- **A defect is a problem encountered by the customer (through users)**
- **Users experience problems**

apparently

- **Projects produce defects**
- **Too few defects are found before delivery to the customer/users**

however,

- **There is a lot of knowledge how to reduce the generation and proliferation of defects**

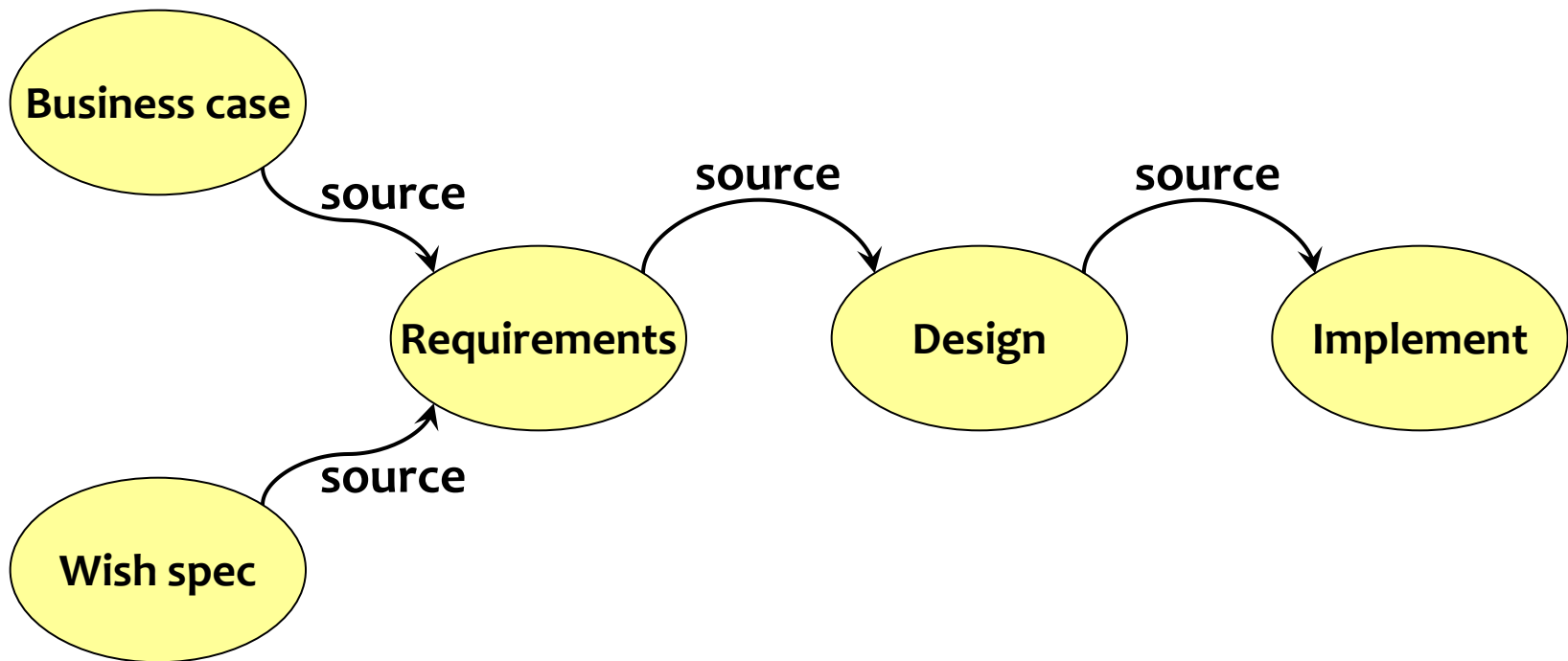
And there is a large budget to do something about it:

- **Some 50% of project time is consumed by all kinds of checking**
- **In software:**
 - **About 50% of developed software is never used**
 - **More than 50% of delivered software is never used**

Where do we make mistakes ?

- **Wish specification** Thank you, nice input
- **Business Case** Why are we doing it
- **Requirements** What the project agrees to satisfy
- **DesignLog** Selecting the 'optimum' compromise and how we arrived at this decision
- **Specification** This is how we are going to implement it
- **Implementation** Code, schematics, plans, procedures, hardware, documentation, training
- **Process Log** Describing how and why we arrived at which current practices

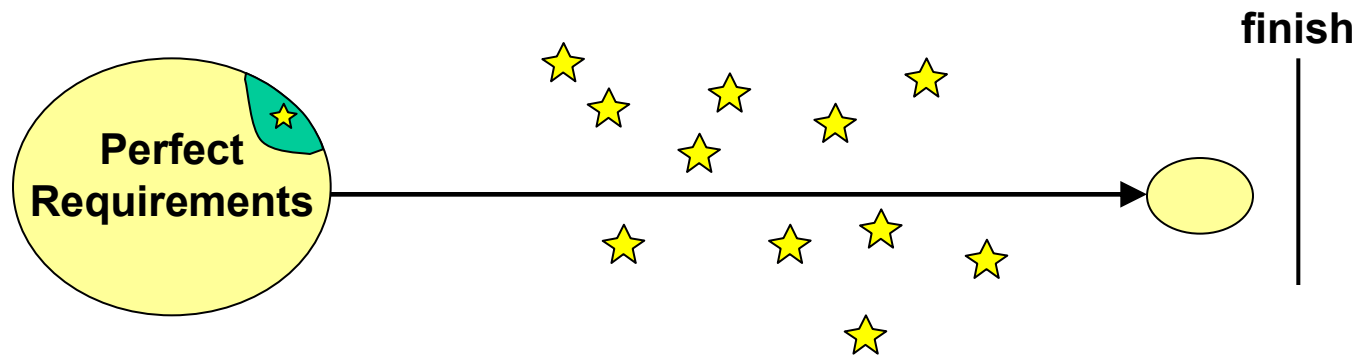
Documents and Sources



Defects



- A design does not have bugs, it has *defects*
- Defects do not *emerge*
- People make errors and thus cause defects
- Changing a requirement causes a lot of defects



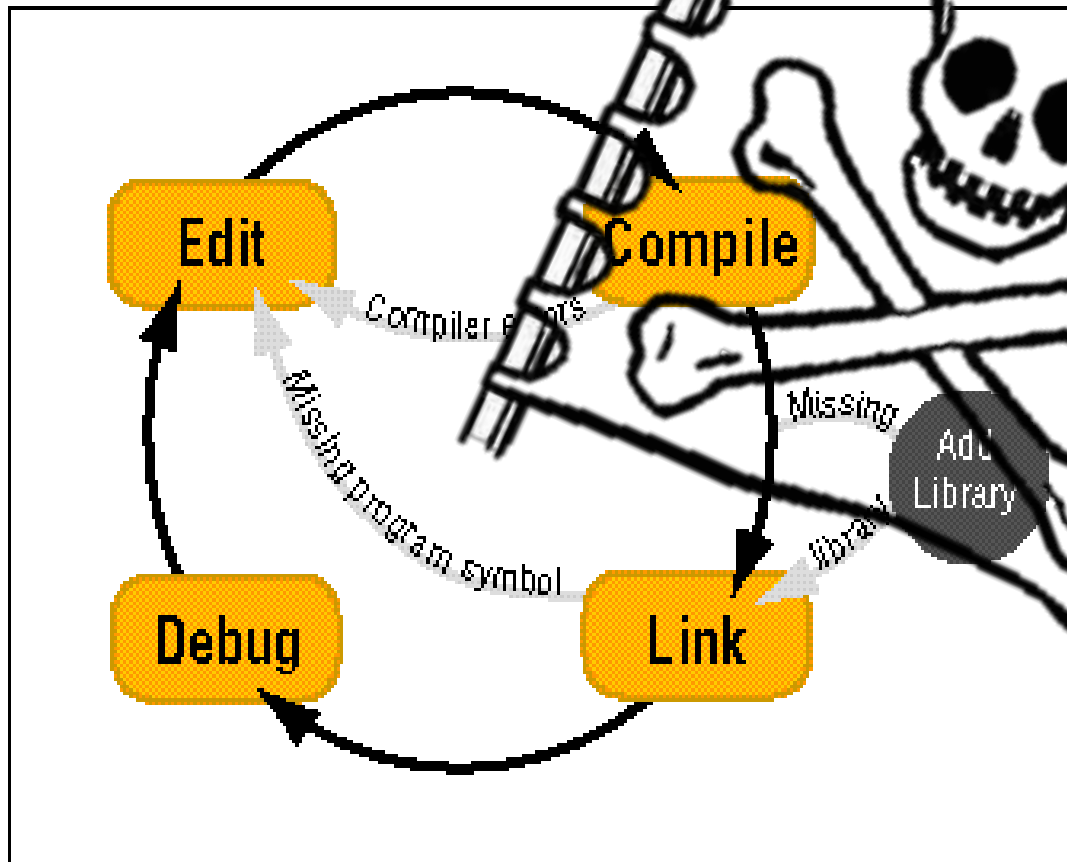
Are defects a problem for you ?

- Which types of defects ?
- How do you know ?
- Perhaps there are problems you don't even know ?
- What can we do about it ?

Debugging ???



Design during coding: trial-and-error method



Bugs are so important, are they really?

- People make mistakes; we are people
 - “Software without bugs is impossible”
 - Testers try to find as many bugs as possible
 - Bugs are counted
 - We try to predict the number of bugs we will find
 - It is suspect if we don't find the expected number
 - Bugs are normal
 - What would we do if there were no bugs any more?
- As long as we keep putting bugs in the center of the testing focus, there will be bugs

Defects found are symptoms of deeper lying problems

Repairing apparent defects creates several risks:

- Repair is done under pressure
- We think the problem is solved
- We introduce scars
- After finding the real cause, the redesign may make the repair redundant: time lost
- We keep repeating the same problems



→ **Do Root Cause Analysis and make sure it never happens again**

Dijkstra (1972)

It is a usual technique to make a program and then to test it

However:

Program testing can be a very effective way to show the presence of bugs

but it is hopelessly inadequate for showing their absence

- **Conventional testing:**
 - Pursuing the very effective way to show the presence of bugs
- **The challenge is, however:**
 - Making sure that there are no bugs
 - And how to show their absence if they're not there

Software testing

- **50% of defects is not found in test**
- **Repair of defects causes defects**
- **A compiler finds only 90% of syntax errors**
- **Of 4 defects:
2 found by compiler, 1 at test and 1 by the customer**
- **How much % of your projects is used for
test, finding, repair, re-test?**
- **How much % of the defects did you find and really fix?**
- **Now many of the defects will be repeated ?**

What is the main function of Testing and QA ?

- **Deming:**

- Quality comes not from testing, but from improvement of the development process. Testing does not improve quality, nor guarantee quality. It's too late. The quality, good or bad, is already in the product. You cannot test quality into a product.

→ **Development is the customer**

- **Testing helps developers to become perfect**
- **Testing is a project to run alongside and synchronized to the development project**
- **Therefore, it must be organised like any other project**

Testing is very expensive

- **You can prove the existence of a defect** (if you found one)
- **You cannot prove the absence of defects** (if you didn't find any)
- **Proving the absence of defects is difficult**
- **Proving the existence of defects is also difficult**
- **Why do we put so much emphasis on finding defects?**
- **While what we want is no defects**
- **Testers should learn better how to prove the absence of defects**

while

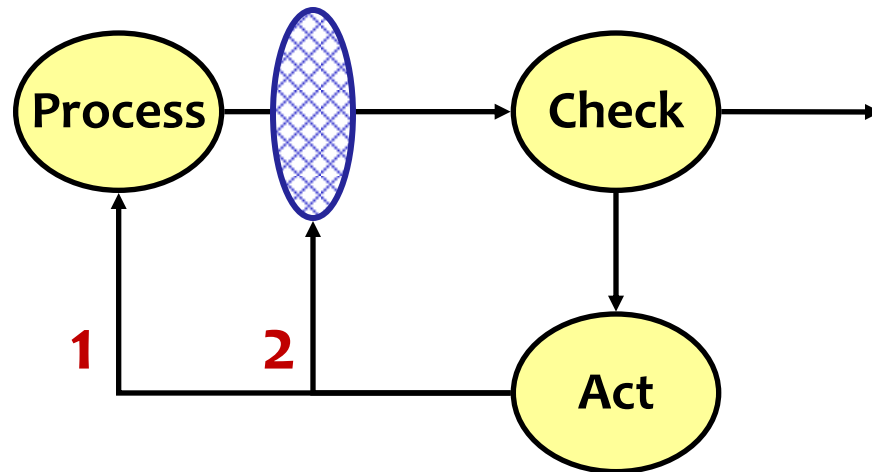
- **Developers should learn better how to avoid defects**
- **Testers can help them**

So, no testing?

- **Testing is important**
however
- **Goal should not be defect finding**
- **But rather measuring the quality of the production process**
- **And feedback to development**

- **Final testing is to check that it works correctly**

Testing is checking correctness



1. How can we prevent this ever happening again?
2. Why did our earliest sieve not catch this defect?

Let's move

Let's move from

Fixation to Fix

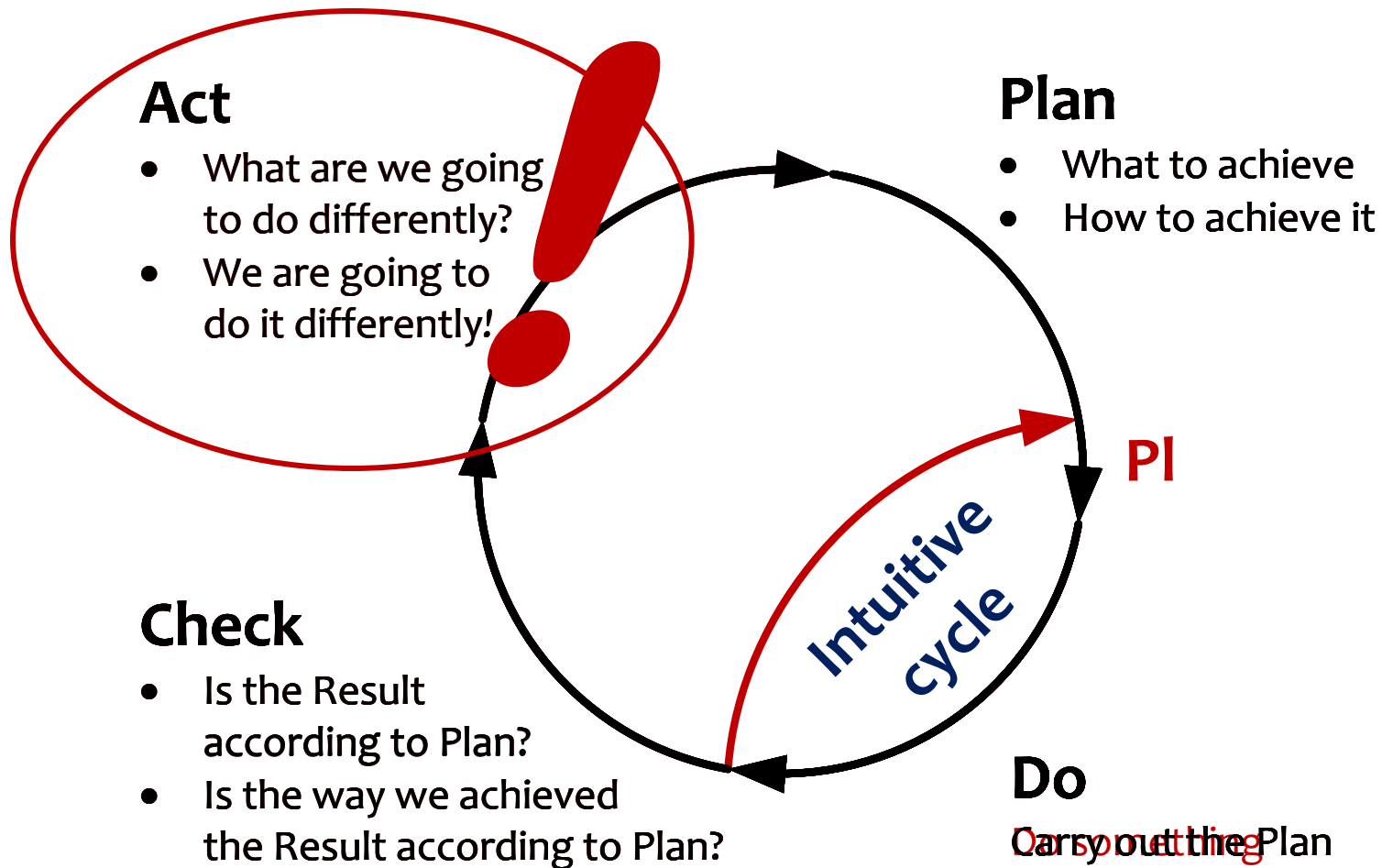
to

Attention to Prevention

- **If we don't deal with the root, we will keep making the same mistakes over and over**
- **Toyota Production system: "Stop the Line"**
- **Without feedback, we won't even know**
- **With quick feedback, we can put the repetition to a halt**

The essential ingredient: the PDCA Cycle

(Shewhart Cycle - Deming Cycle - Plan-Do-Study-Act Cycle - Kaizen)



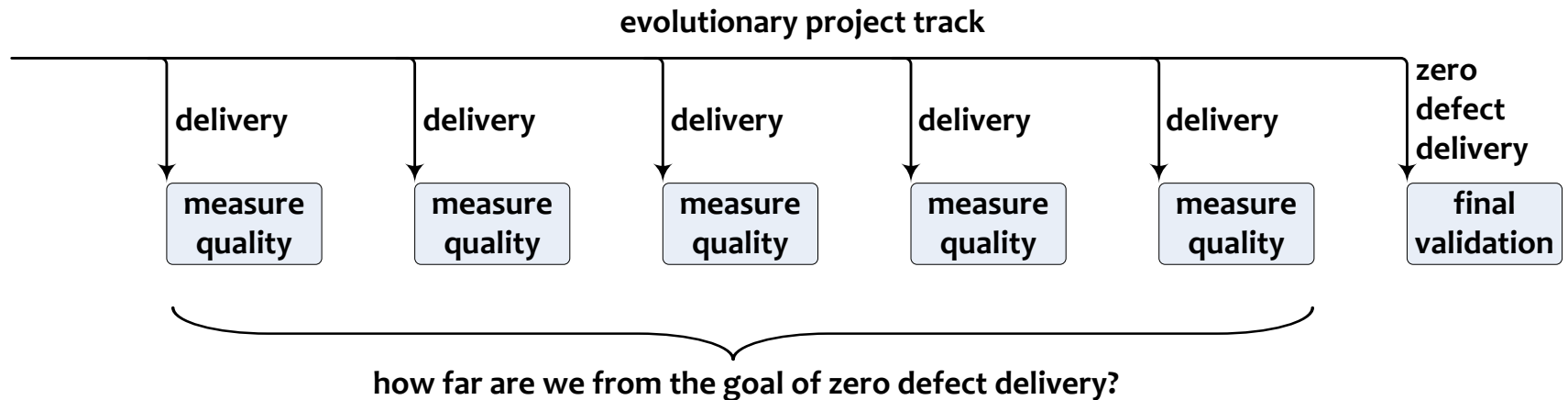
Developers are continuously optimizing

- **The product**
how to arrive at the most effective product (goal !)
- **The project**
how to arrive at the most effective product effectively and efficiently
- **The process**
 - **Finding ways to do better**
 - **Learning from other methods**
 - **Absorbing those methods that work better**
 - **Shelving those methods that currently work less**

Testers are continuously optimizing

- **The product**
how to arrive at the most effective product (goal !)
- **The project**
how to arrive at the most effective product effectively and efficiently
- **The process**
 - Finding ways to do better
 - Learning from other methods
 - Absorbing those methods that work better
 - Shelving those methods that currently work less

Evo Testing

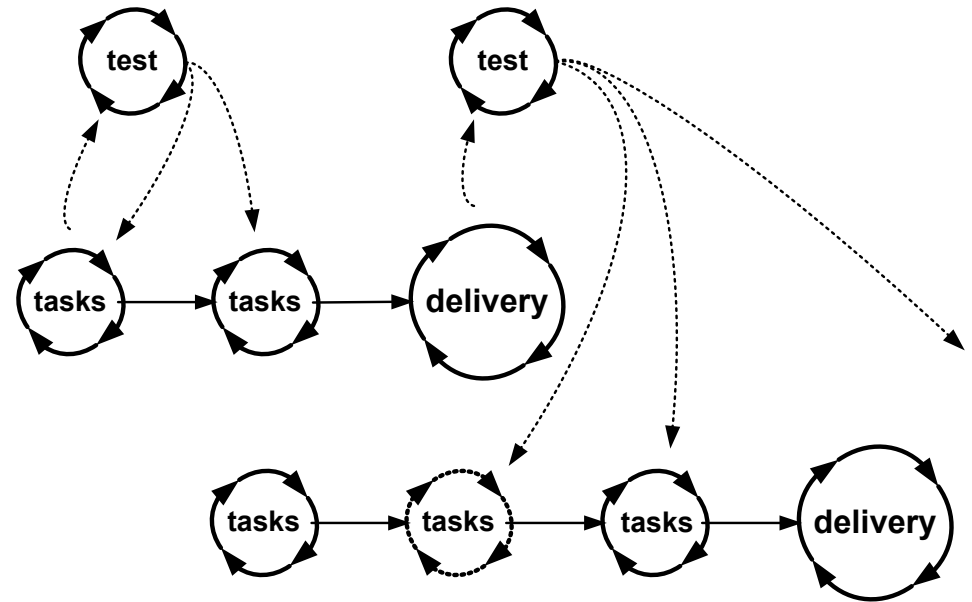


- **Final validation shouldn't find any problems**
- **Earlier verifications mirror quality level to developers: how far from goal and what still to learn**
- **Evo has no debugging phase !**
- **Checking is done *in parallel with development***
- **Checking doesn't delay the project**

Further Improvement

- **Tester's customer is "the developers"**
- **Finding defects is not the goal**
- **Project Success is**
- **Testers select and use any method appropriate**
- **Testers check work in progress even before it is finished**
- **Testing is organized the Evo way, entangling intimately with the development process**

Evo cycles for Testing



- Testers organize their work in weekly TaskCycles
- DeliveryCycle is the Test-Feedback cycle
- Testers use their own TimeLine, synchronized with the developers TimeLine
- Testers conclude their work in sync with developers
- Testers know what they are supposed to test
- Testers check work in progress even before it is finished

Testing Metrics

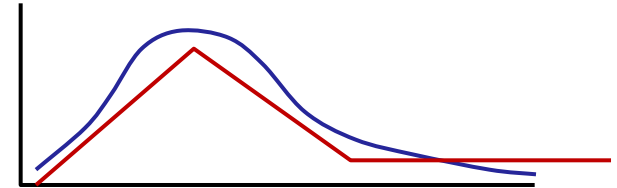
**Don't *improve* non-value-adding activities
- better *eliminate* them**

- **Defects per Page, Defects per ...**
Stop counting defects, it conveys a bad message. Prevent defects
- **Incoming defects per month** (by test, by user)
Don't count. Do something. Users shouldn't experience problems
- **Defect detection effectiveness or Inspection yield**
 - Yield is 30% ~ 80%; testers are human after all
 - Zero defects at user means zero defects before final test
 - Whether that is difficult is beside the point

More Testing Metrics

- **Cost to find and fix a defect**
 - The less defects the higher the cost per defect
 - This was a bad metric anyway
- **Closed defects per month**
 - Closing depends on prioritizing process, through Candidate Tasks List
- **Age of open customer found defects**
 - Purpose of many metrics seems to be policing: not trusting people to take appropriate action
 - In Evo we take appropriate action
- **Remaining defects**
 - Still useful as measure of Prevention success

When are we done with testing?



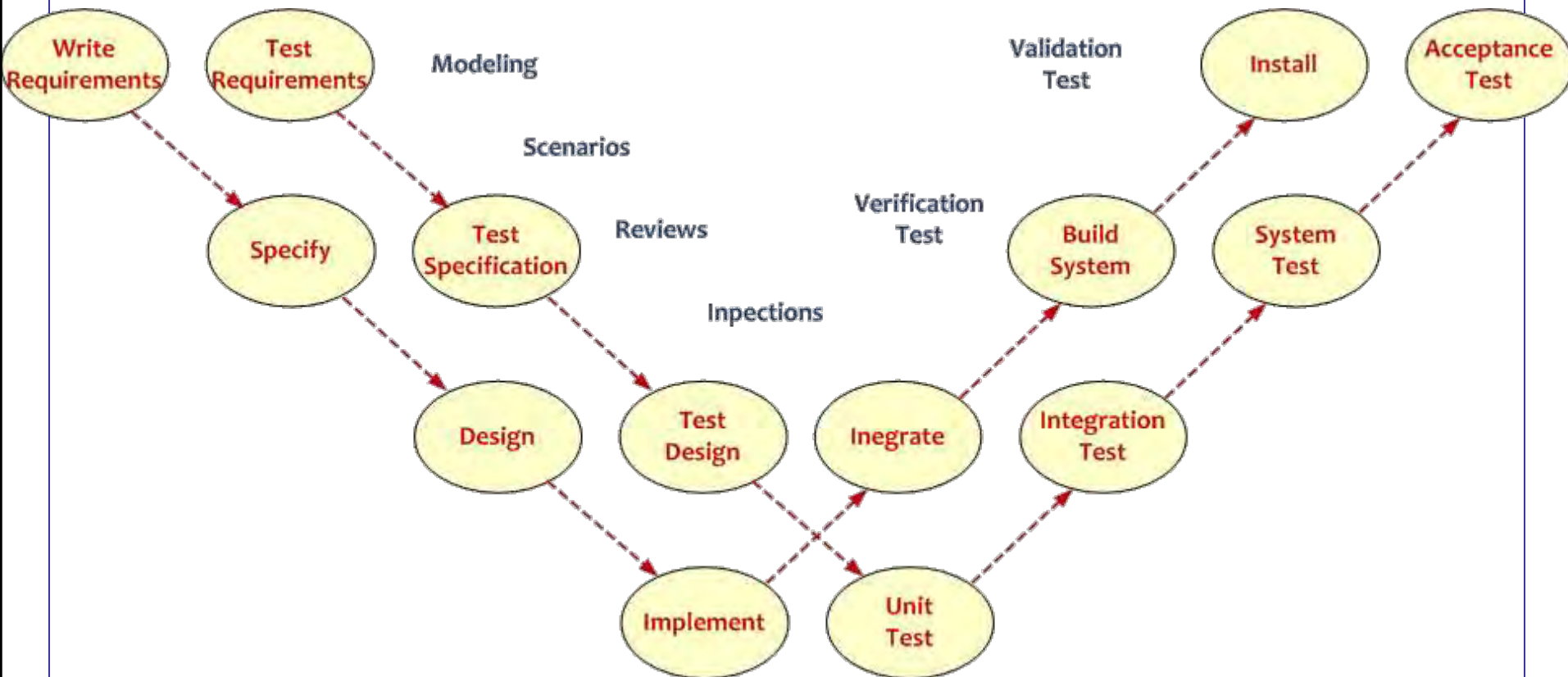
- **Conventional:**
 - Number of defects found per day less than n
 - Defect backlog decreased to zero
 - Prediction by curve fitting based on early found defect numbers
 - Using ~~historical~~^e data
 - Other?
- **Evo:**
 - The project is ready at the agreed date, or earlier
 - That includes testing

Defects typically overlooked

- **Functions that won't be used** (superfluous requirements)
 - Why to repair defects in the implementation of these requirements?
 - The only defect is that it has been implemented
- **Nice things** (not checked, not paid for)
Shouldn't be there in the first place
- **Missing quality levels** (should have been in requirements)
Checking the implementation of the documented requirements won't help
- **Missing constraints** (should have been in requirements)
Product could be illegal
- **Unnecessary constraints** (not required)
What would testing say about these?

Remember the W-model

but also remember: all models are wrong ...



Ways to achieve better quality ?

- **Hope ??**
- **Test ?**
- **Debug ??**
- **Review ?**
- **Walkthrough ?**
- **Inspection ?**

Prevention !!

CR/PR/RI Database



Focus on
Prevention

- **Change Requests**
CR: customer pays
- **Problem Reports**
PR: you pay
- **Risk Issues**
RI: prevention

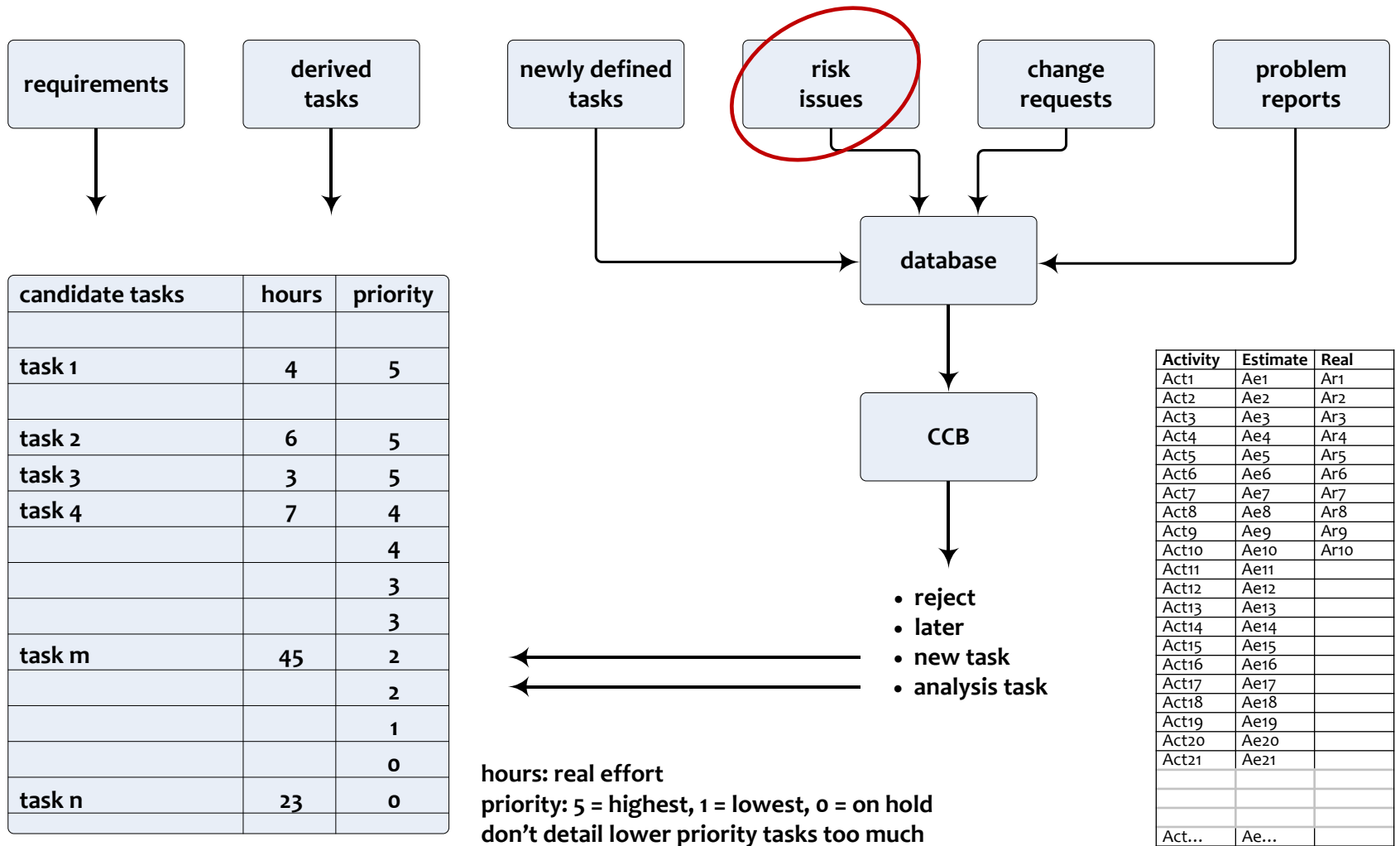
- **Where, what, when, who**
- **Urgency, severity**
- **Classification**
- **Status**



Focus on
“Repair”

- **Where caused and root cause**
- **Where should it have been found earlier**
- **Why not found earlier**
- **Prevention plan**
- **Analysis tasks defined and put on Candidate Task List**
- **Prevention tasks defined and put on Candidate Task List**
- **Check lists updated for finding issues easier, in case prevention doesn't work yet**

Anything we think must be done goes through the *Candidate Task Mechanism*



Reviews & Inspections

Are you reviewing?

*

Many types of Review to choose from

- **Informal Review**
- **Pair Programming**
- **Technical Review**
- **Walkthrough**
- **Formal Inspection (Fagan type)**
- **Cleanroom Inspection**
- **Formal Inspection (Gilb/Graham type)**
- **Agile/Extreme/Lean/Early Inspection**
- **Gate Review**
- **Unit Test**
- **Debugging**
- **Test**

Techniques

- **Can you look at this ?**
- **Over the shoulder**
- **Pair Programming**
- **E-mail**
- **Tool**
- **On Screen**
- **Projector**
- **On Paper**
- **Formal process**

Formal Reviews (vs Ad-Hoc)

- **Defined, repeatable process**
- **Measures effectiveness**
- **Continuous improvement**
- **Rules/checklists**
- **Feeds prevention process**

Typical documents

- **Wish specification**

Thank you, nice input

- **Business Case**

Why are we doing it

- **Requirements**

What the project agrees to satisfy

- **DesignLog**

Selecting the 'optimum' compromise and how we arrived at this decision

- **Specification**

This is how we are going to implement it

- **Implementation**

Code, schematics, plans, procedures, hardware, documentation, training

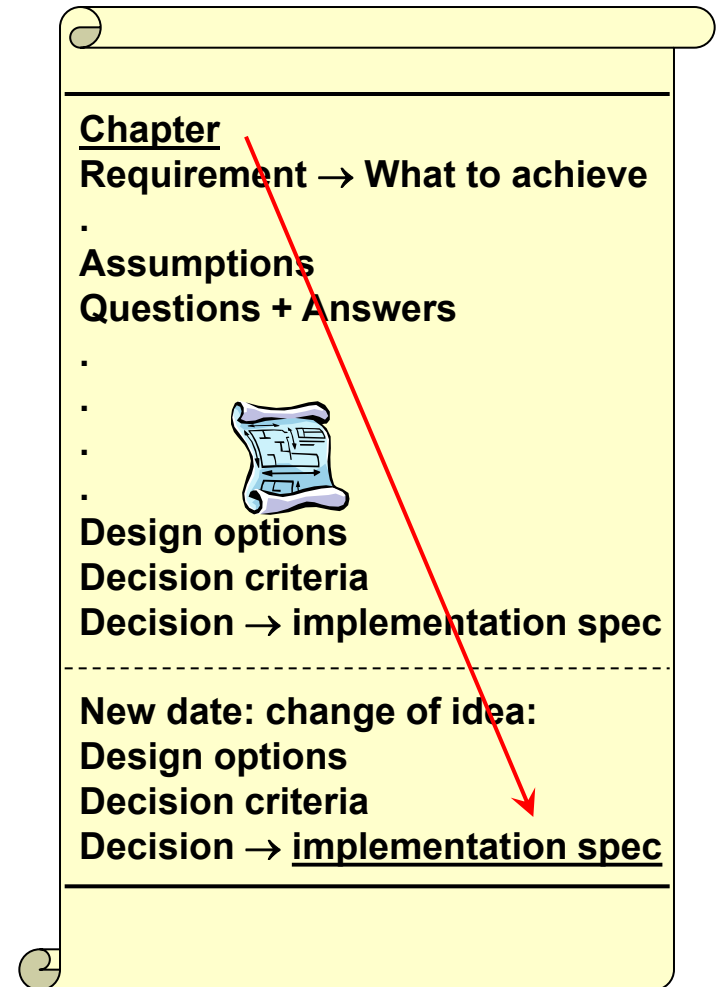
- **Process Log**

Describing how and why we arrived at which current practices

DesignLog

(project level)

- **In computer, not loose notes, not in e-mails, not handwritten**
 - Text
 - Drawings!
 - On subject order
 - Initially free-format
 - For all to see
- **All concepts contemplated**
 - Requirement
 - Assumptions
 - Questions
 - Available techniques
 - Calculations
 - Choices + reasoning:
 - If rejected: why?
 - If chosen: why?
- **Rejected choices**
- **Final (current) choices**
- **Implementation**



Did you ever do a Review ?

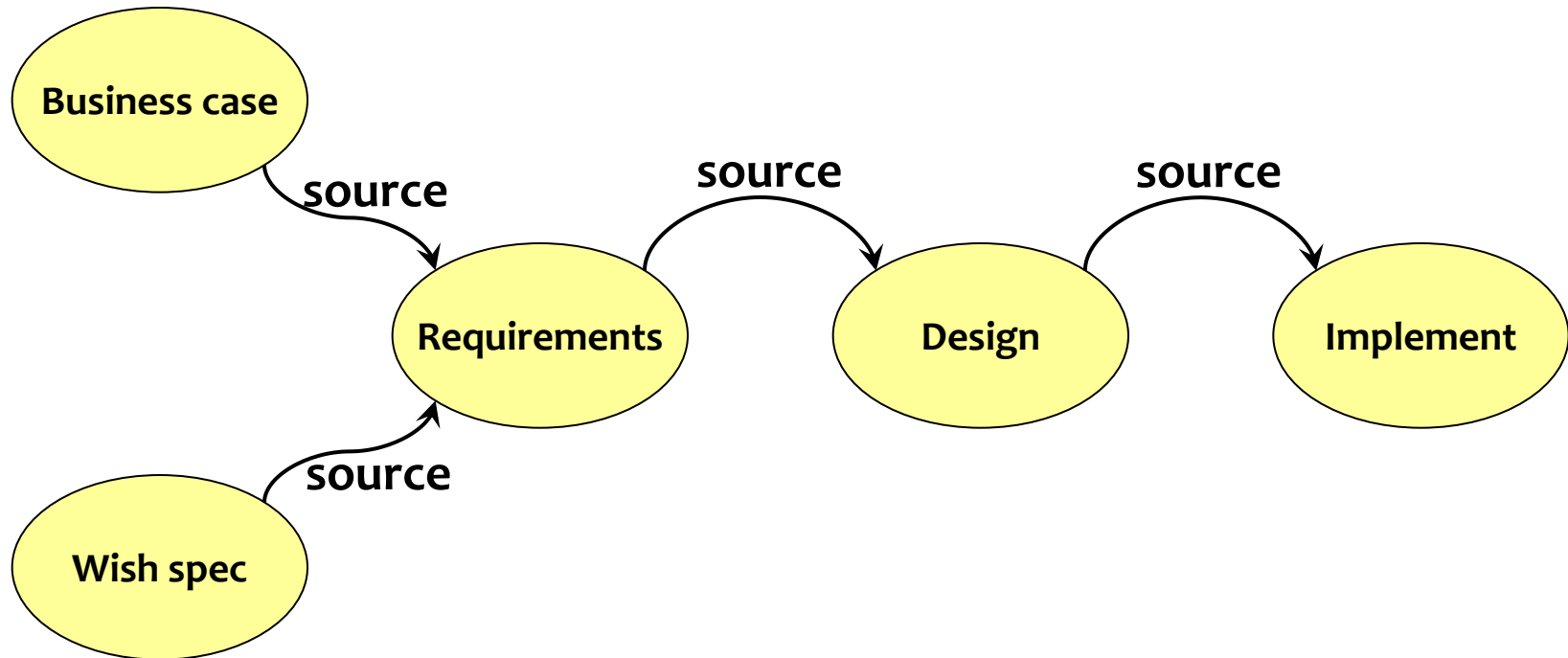
*

Let's review

- **Do we have a document ?**
- **Select one representative page**
- **Make some copies**
- **Review**
- **Then we'll discuss the result of the review**

Simple Rule for Reviews

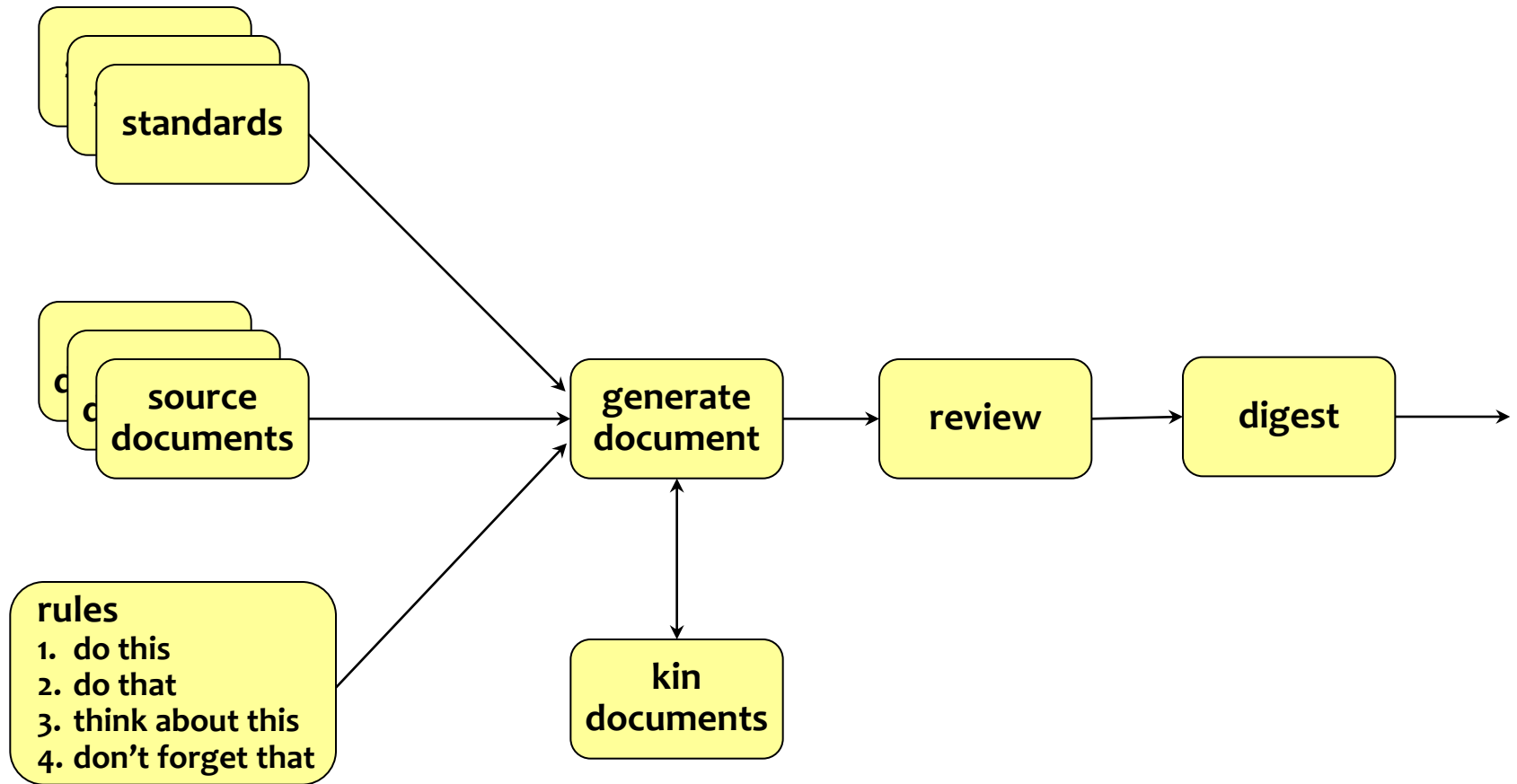
“We don’t review unless there is a source document”



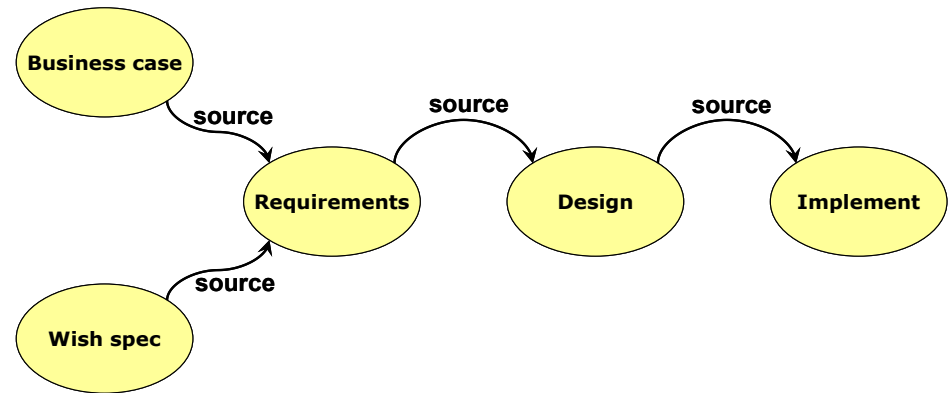
Now review again

- **Any difference ?**

Document generation



Rules



- **Any workproduct will be reviewed against**

- **Itself**
- **Kin documents**
- **Source documents**

If we don't have the source, how can we judge the workproduct?

- **We always update the source document first before changing the workproduct(s)**

- **First change the Design, then the Code and the Test**
- **First change the Requirement, then the Design, then the Code and the Test**

A typical Review ...

- The document to be reviewed is given out in advance
- Typically dozens of pages to review
- Instructions are "please review this"
- Some people have time to look through it
- Review meeting often lasts for hours
- Typical comment: "I don't like this"
- Much discussion, some about technical approaches, some about trivia
- Don't really know if it was worthwhile, but we keep doing it
- Next document reviewed will be no better

Inspection is different

- **The document to be reviewed is given out in advance**
not just product - rules to define defects, other docs to check against
- **Typically dozens of pages to review**
chunk or sample
- **Instructions are "please review this"**
training, roles
- **Some people have time to look through it**
entry criteria to meeting, may be not worth holding
- **Review meeting often lasts for hours**
2 hr max
- **Typical comment: "I don't like this"**
Best Practice rules - Rules are objective, not subjective
- **Much discussion, some about technical approaches, some about trivia**
no discussion, highly focused, anti-trivia
- **Don't really know if it was worthwhile, but we keep doing it**
exit criteria - continually measure costs and benefits
- **Next document reviewed will be no better**
most important focus is improvement in processes and skills

Inspection

- **Most rigorous form of review**
- **Pioneered by Fagan (IBM)** (paper 1976)
 - Locating all the defects in a work product
- **Inspection economics: Gilb/Graham** (Software Inspection, 1993)
 - Quantifying the defect density of a work product and preventing poor quality work from moving downstream
- **Is not the same as review**
- **Use:**
 - Walkthroughs for training
 - Technical Reviews for consensus
 - Inspections to improve the quality of the document and its process
 - Gate Reviews to decide what to do with it

**Would you like to base further work or decisions
on a document of unknown quality?**

Software Inspection

Tom Gilb

Dorothy Graham



ADDISON-WESLEY



A ready to use recipe ...

16 page Inspection Manual

www.malotaux.nl/nrm/pdf/InspManual.pdf

Inspection Manual

Procedures, rules, checklists and other texts
for use in Inspections

Version: 0.43 (Changed *Plan* into *Goal*)

Date: Oct 13, 2007

Owner: Niels Malotaux

Status: not inspected

Intended readership: anybody interested in or busy with inspections

Note: Most of these texts are originally taken from the book:

"Software Inspection" by Tom Gilb and Dorothy Graham

Addison Wesley, 1993, ISBN 0-201-63181-4, and from

web-sites, such as www.result-planning.com (Tom Gilb's web-site)

This is a starting point from which the procedures, rules, etc.

may be adapted to the local culture.

Basic Simple Requirements Inspection

- **Use these Rules:**
 1. Unambiguous to the intended readership
 2. Clear to test
 3. No Design
- **A Defect is a violation of a Rule**
- **Check for Major Defects**
 - Major means > 10 hours cost to find and repair if found later
- **Take one page**
- **How many Majors did you find on this page?**

Inspection goals and effects

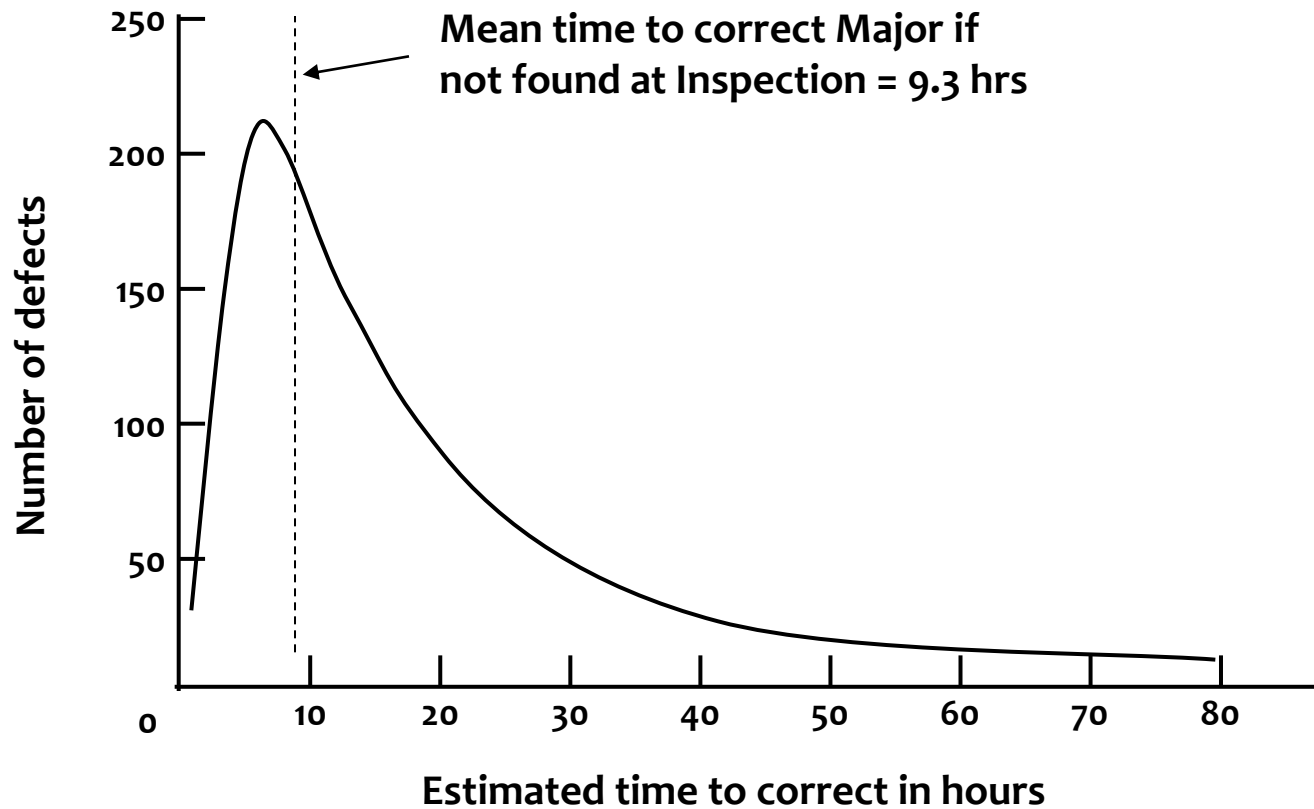
- **Identify and correct major defects**
- **Most important:**
Identify and remove the source of defects
- **Consequence:**
Education and interaction:
How should we generate documents in the first place?
- **Interesting side-effect:**
People get to know each others documents efficiently

Defect classes

- **Major defect**
 - Defect probably has significantly increased costs to find and fix later (test, field)
 - 10 engineering hours lost extra
 - Average time in work-hours to find, log and fix a major defect by Inspection is 1 hour (observed by many sources)
- **Minor defect**
 - Not major (no significant impact on result)
- **Super-major/critical**
 - Order of magnitude more costly than major
 - Project threat

Cost of Repair

ref SI, fig 14.6, p315



Rules

- **Rules are the law for documents**
- **Defect = Rule violation**
not: “I think this is wrong”
- **Rule:**
All quality requirements must be expressed quantitatively
- **Typical requirements found:**
The system should be extremely user-friendly
The system must work exactly as the predecessor
The system must be better than before

Generic Specification Rules

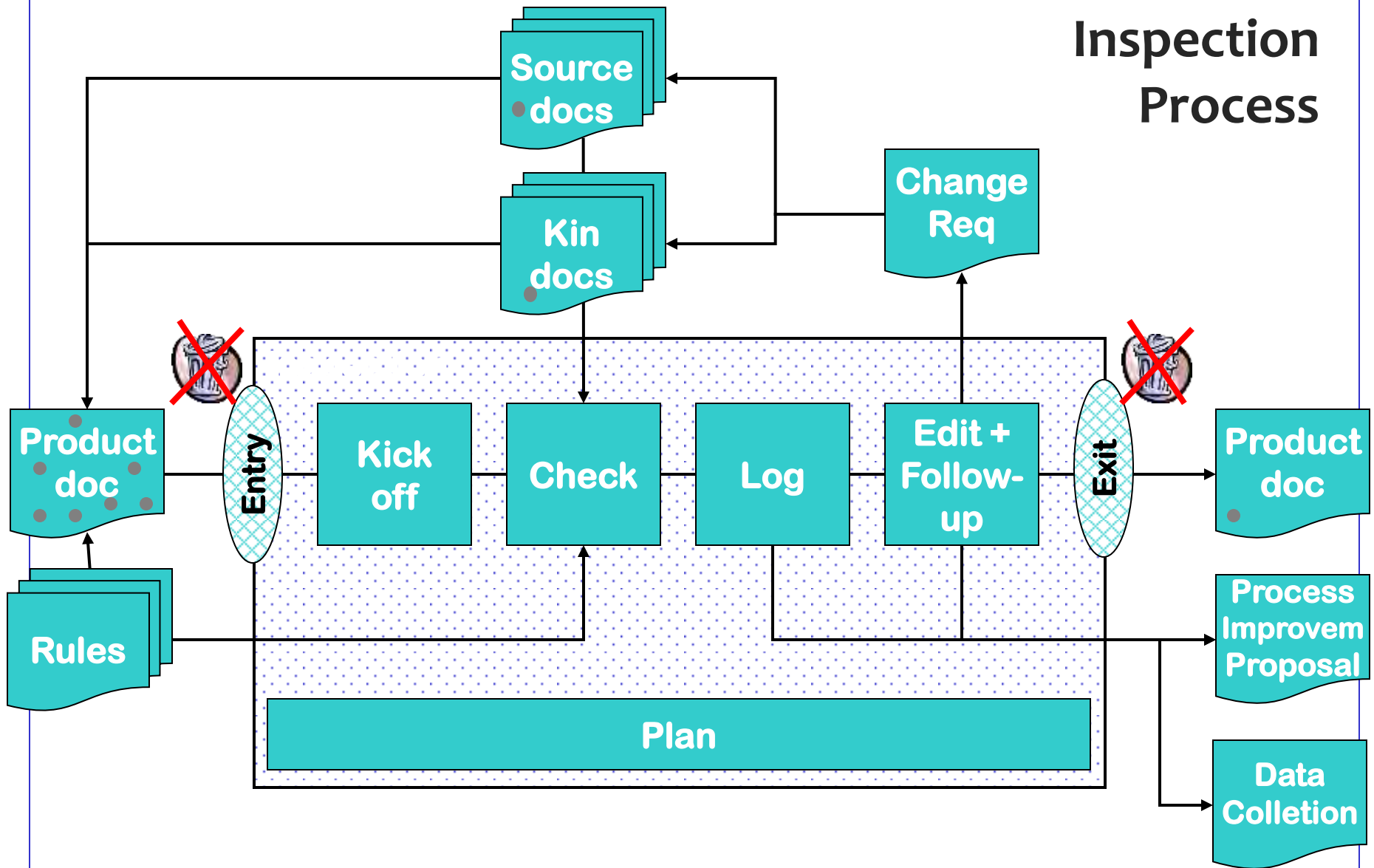
(see Inspection Manual)

- GE0 (def) Generic engineering specification rules apply to all engineering documents as required best practices
- GE1 (relevant) All statements should be relevant to the subject
- GE2 (complete) There should not be any significant omissions
- GE3 (consistent) Statements should be consistent with other statements in the same or related documents
- GE4 (unambiguous) All specifications should be unambiguous to the intended readership
- GE5 (note) Comments, notes, suggestions, not official part of document shall be clearly marked (“”, *ital*, ******)
- GE6 (brief) All specifications shall be as brief as possible, to support their purpose, for the intended readership
- GE7 (clarity) All specifications shall result in clarity to the intended readership regarding it’s purpose or intent (the burden is on author, not the reader)
Note: It is not enough that statements are unambiguous. They must contain clarity of purpose: why is it there?
- GE8 (elementary) Statements shall be broken into their most elementary form
Note: This is so that they each can be cross-referenced externally (Traceability)
- GE9 (unique) Specifications shall have a single instance in the entire project documentation
- GE10 (source) Statements shall have source info (spec ← source)
- GE11 (risk) The author should clearly indicate any information which is uncertain or poses any risk to the project, using indications like: {<vaguely defined>, ?, ??, 70% ±20, suitable comments or notes}
- GE12 (verifiable) All statements should be verifiable
- GE13 (true) The statement is simply not true

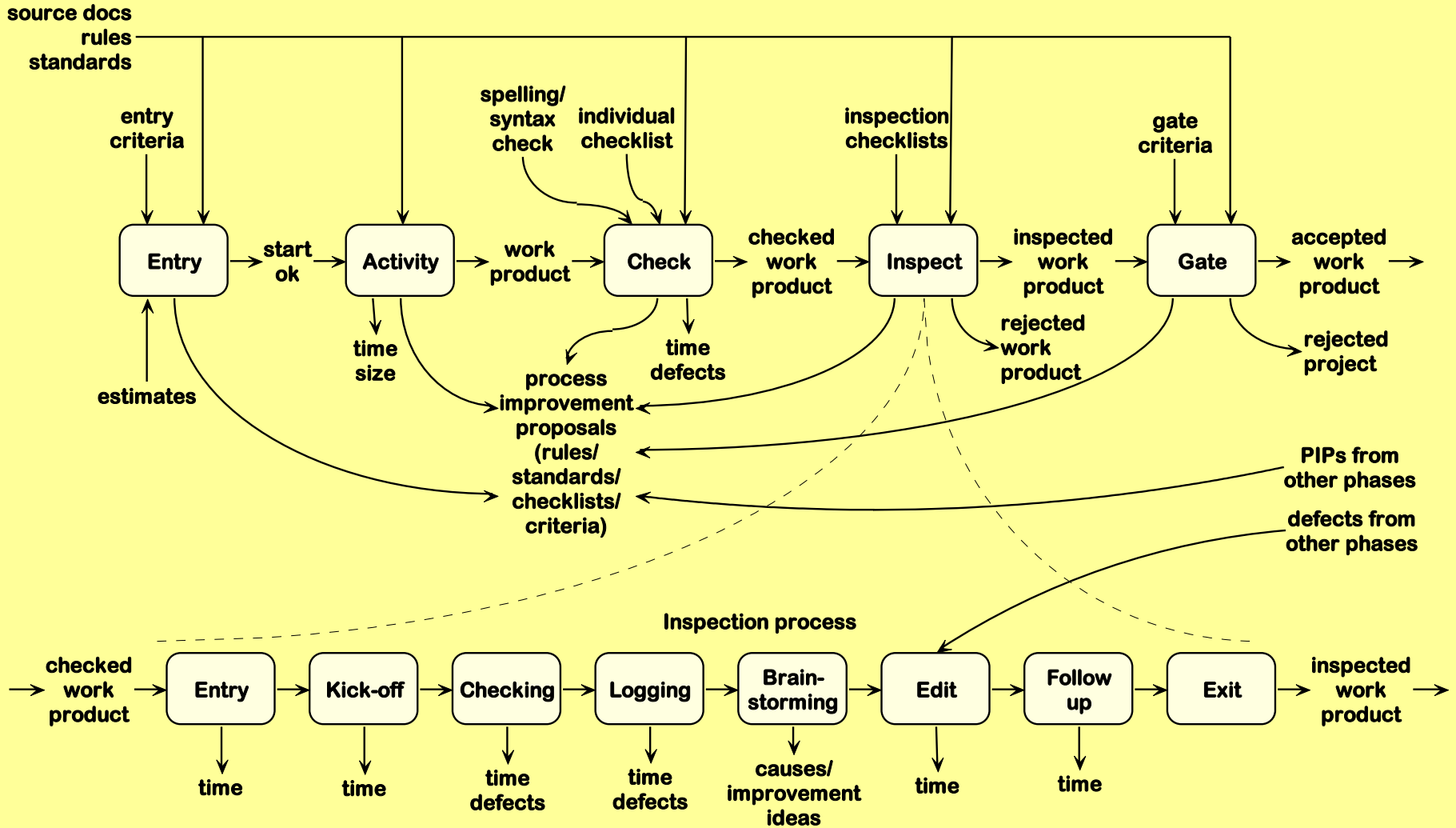
Check Lists

- **Checklists contain interpretations of Rules to help reviewers to find more issues**
- **Rules are “The Law” 法律 (?)**
- **Checklists provide “Jurisprudence” 法学 (?)**

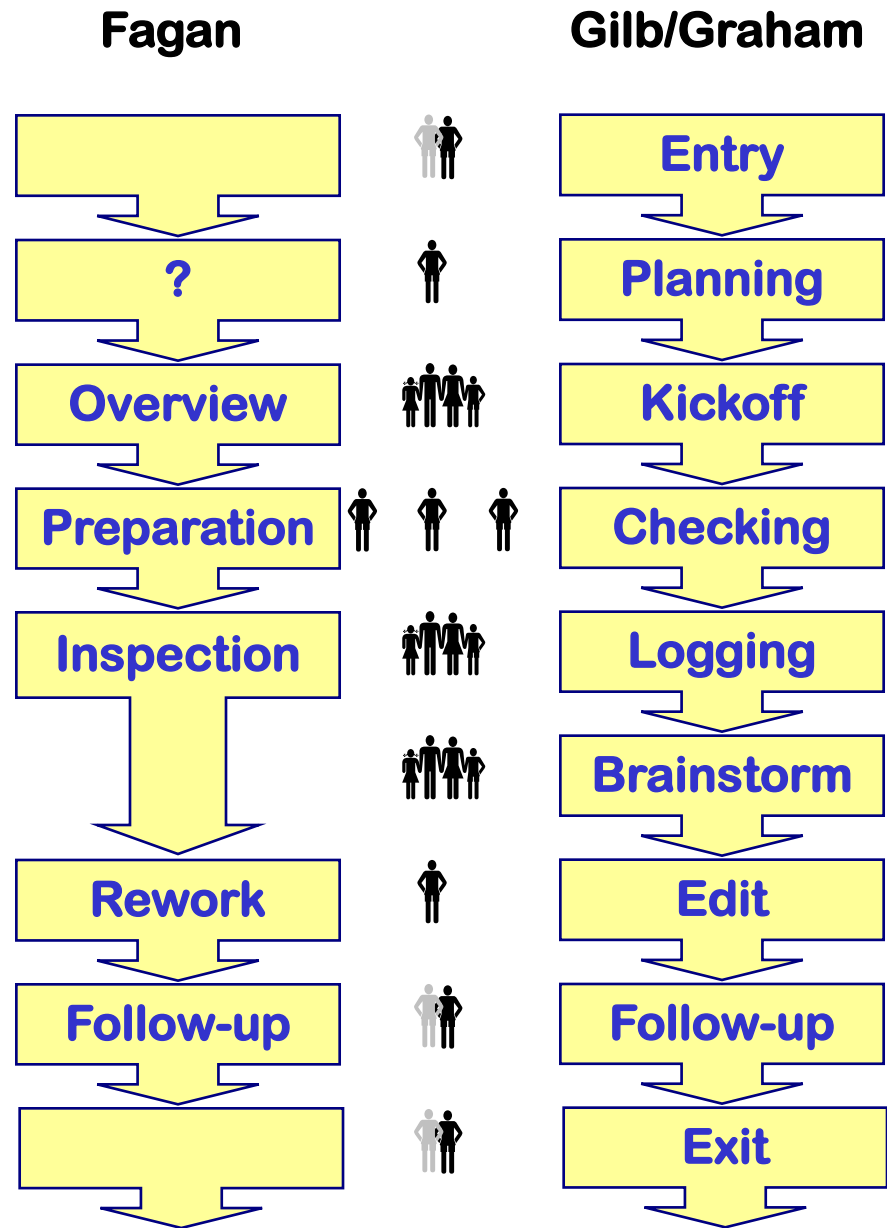
Gilb/Graham Inspection Process



Development project sub-process



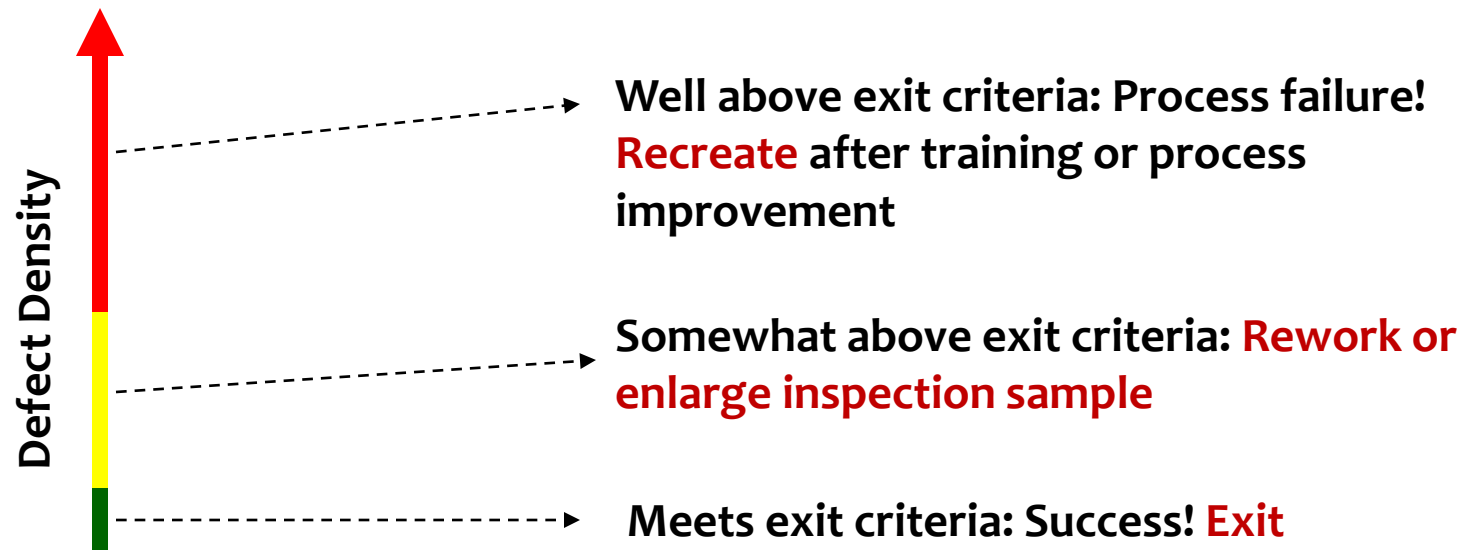
Inspection Process Steps



Gilb/Graham Concepts

Entry and Exit Criteria

Once the quality level of a specification is known, there are three possible paths forward:



Optimum Checking Rate

- The most **effective** individual speed for ‘checking a document against all related documents’ in page/hr
- Not ‘reading’ speed, but rather **correlation** speed
- Failure to use it, gives ‘bad estimate’ for ‘Remaining defects’

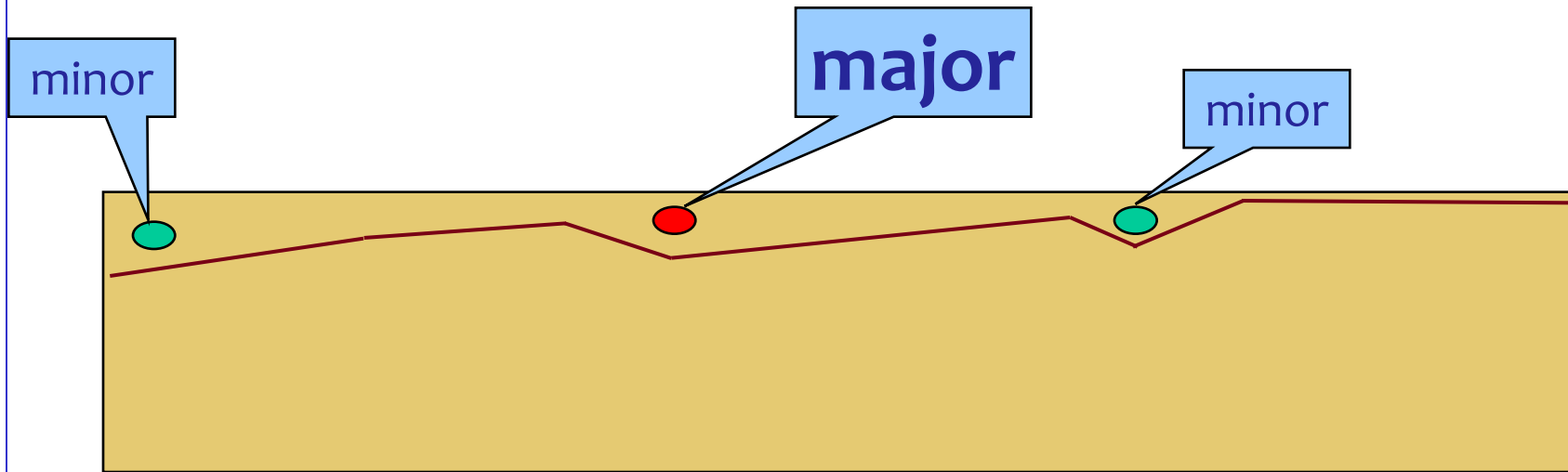
- 100~250 SLoC per hour
- 1 page of 300 words per hour (“logical page”)

Optimum checking rate



Here's a document: review this (or Inspect it)

Review “Thoroughness”?

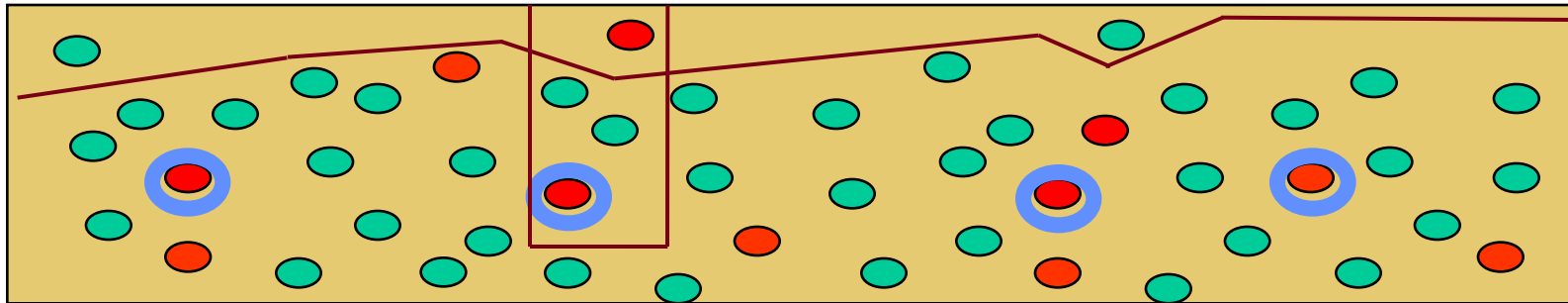


- **Ordinary review**

- Find some defects, one Major
- Fix them
- Consider the document now corrected and OK ...

Inspection Thoroughness

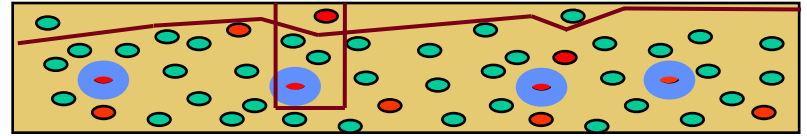
Ref. Dorothy Graham



- **Inspection can find deep-seated defects**
- **All of that type can be corrected**
- **Needs optimum checking rate**

- **In the above case we are clearly taking a sample**
- **In the “shallow” case we we’re also taking a sample, however, we didn’t realize it !**

Gilb/Graham Inspection

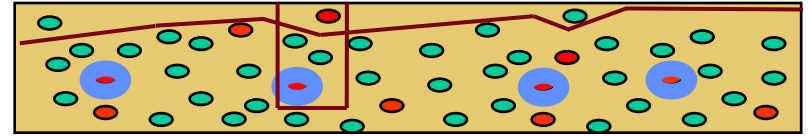


Gilb/Graham inspection differs from other types of inspection in some or all of these ways:

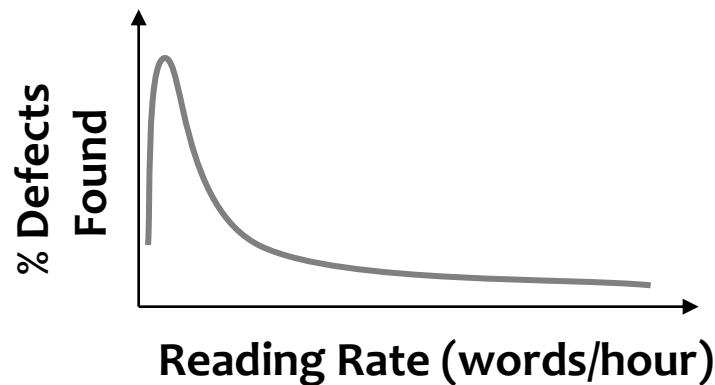
- **Purpose:**
Quantifying quality, not searching for all defects
- **Controlled reading rate:**
The material being inspected is read very slowly in order to identify as many defects as possible (deep vs shallow sample)
- **Sampling:**
Only samples are inspected to optimize time and effort investment while maintaining the reading rate
- **Entry/Exit Criteria:**
Quantified entry and exit criteria used to guide the inspection effort
- **Rules:**
Written rule sets used to locate and classify defects

Gilb/Graham Concepts

Reading Rate



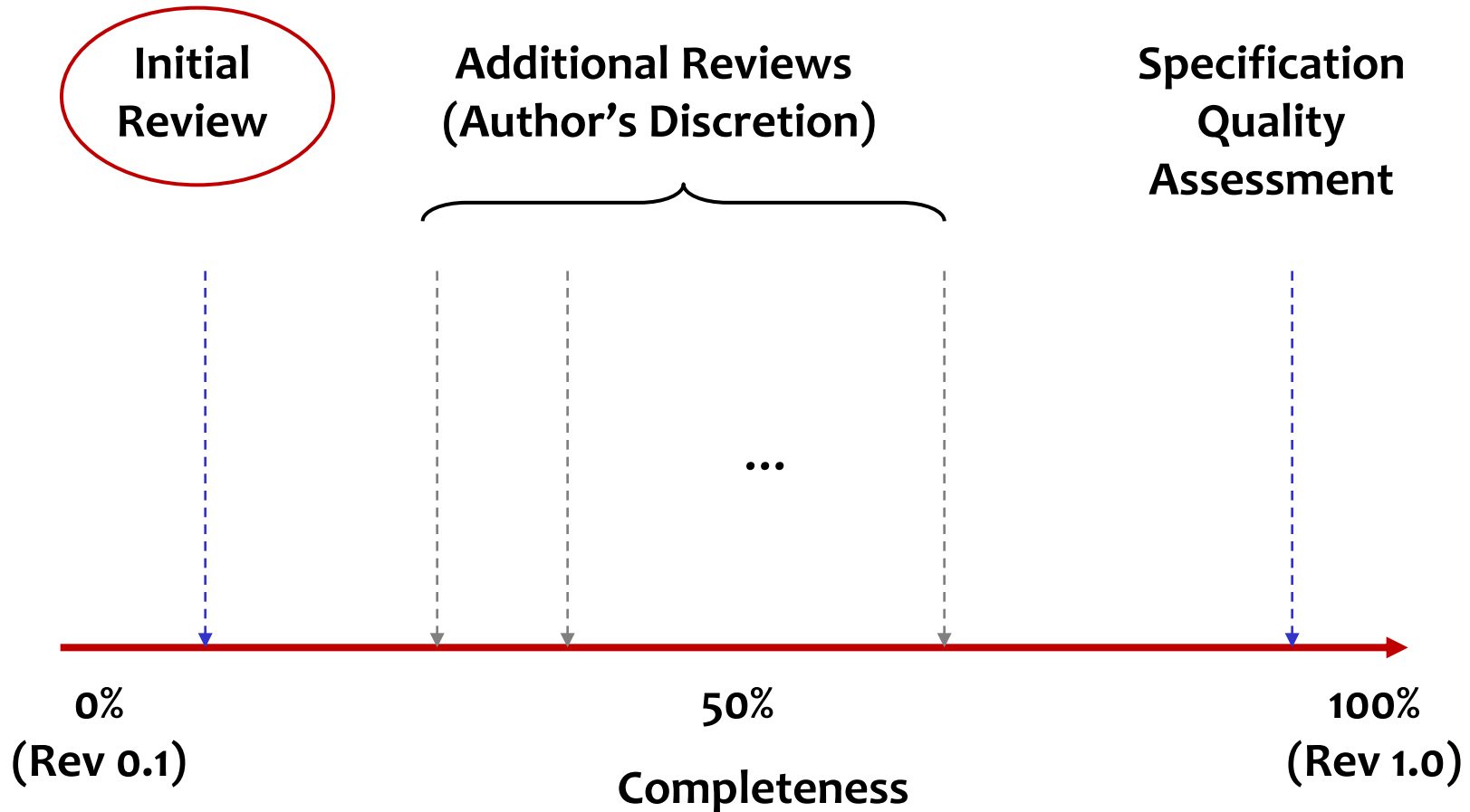
- **Default recommended reading rate is one logical page per hour, lower than in many other inspection methods**
- **This ensures adequate time to locate the vast majority of latent defects in the specification**
- **Supporting documents, rules, etc. can be read at any speed**



Read too fast and you will miss most of the defects!

Early Inspection

Prevention costs less than Repair



Initial Review

- Purpose:** Locating mistakes and tendencies that could lead to injecting major defects if not corrected
- When:** As soon as the author has completed a small representative portion of the specification, typically a few pages or 600-1200 words (e.g. few requirements)
- Who:** Individual or small team (1 or 2)
- Expertise in the subject matter
 - Expertise in generic principles (such as requirements engineering, design, specific language)
- What:** Detailed review of the specification against rules and checklists for known error conditions and dangerous tendencies; formal inspection may be used
- Duration:** Because the sample is small, the initial review takes only 1-2 hr

The earlier it's reviewed, the more defects we can prevent

Initial Review Checklist

- ✓ **Use a small team of experienced reviewers**
- ✓ **Schedule the review to minimize author waiting time**
- ✓ **Focus on issues that are or will cause major defects**
- ✓ **Avoid elements of style**
- ✓ **Be constructive at all times**
- ✓ **Focus on the work product, and never on the author**
- ✓ **Maintain confidentiality!**
The review is for the author's benefit

Reviewers: Your job is to make the author look like a hero

Case Study 1 - Situation

- **Large e-business integrated application with 8 requirements authors, varying experience and skill**
 - Each sent the first 8-10 requirements of estimated 100 requirements per author (table format, about 2 requirements per page including all data)
 - Initial reviews completed within a few hours of submission
 - Authors integrated the suggestions and corrections, then continued to work
 - Some authors chose additional reviews; others did not
 - Inspection performed on document to assess final quality level

Case Study 1 - Results

Average major defects per requirement in initial review	8
Average major defects per requirement in completed document	3

- **Time investment: 26 hr**
 - 12 hours in initial review (1.5 hrs per author)
 - About 8 hours in additional reviews
 - 6 hours in final inspection (2 hrs, 2 checkers, plus prep and debrief)
- **Major defects prevented: 5 per requirement in ~750 total**
- **Saved $5 \times 750 \times 10 \text{ hr} = 37500 \text{ hr} / 3 = 12500 \times \$50 = \$625000$**

Why Early Inspection Works

- **Many defects are repetitive and can be prevented**
 - Early review allows an author to get independent feedback on individual tendencies and errors
 - By applying early learning to the rest (~90%) of the writing process, many defects are prevented before they occur
 - Reducing rework in both the document under review and all downstream derivative work products

Case Study 2 - Situation

- **A tester's improvement writing successive test plans:**
 - Early Inspection used on an existing project to improve test plan quality
 - Test plan nearly “complete”, so simulated Early Inspection
 - First round, inspected 6 randomly-selected test cases
 - Author notes systematic defects in the results, reworks the document accordingly (~32 hrs.)
 - Second round, inspected 6 more test cases; quality vastly improved
 - Test plan exits the process and goes into production
 - The author goes on to write another test plan on the next project...

Case Study 2 - Results

First round inspection	6 major defects per test case
Second round	0.5 major defects per test case

- **Time investment: 2 hours in initial review, 36 hours total in inspection, excluding rework (2 inspections, 4 hrs each, 4 checkers, plus preparation and debrief)**
- **Historically about 25% of all defects found by testing, were closed as “functions as designed”, still 2-4 hrs spent on each**
- **This test plan yielded over 1100 software defects with only 1 defect (0.1 %) closed as “functions as designed”**
- **Time saved on the project: 500 - 1000 hrs (25% x 1100 x 2-4 hrs)**

Defect Prevention in action: First inspection of this tester's next test plan: 0.2 major defects per test case

Early Detection vs. Prevention

Denise Leigh (Sema group, UK), British Computer Society address, 1992:

An eight-work-year development, delivered in five increments over nine months for Sema Group (UK), found:

- 3512 defects through inspection
- 90 through testing
- and 35 (including enhancement requests) through product field use

After two evolutionary deliveries, unit testing of programs was discontinued because it was no longer cost-effective

Nice job! Early detection has big benefits - BUT...

How many of the 3512 defects found in end-of-line inspections could have been completely prevented by Early Inspection?

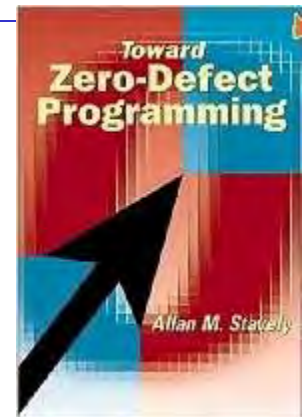
Cost-effective defect prevention is the bottom line

Cleanroom Software Development

- **Design** (Mathematical proof)
- **Verification** (by others)
- **Implementation**
- **Verification** (by others)
- **No unit test**
- **Only Integration Test** (by others)
(Test is Running Code)

- **Verification is for finding defects**
- **Testing is for not finding defects**

Cleanroom (ref Allan M. Stavelly: Toward Zero Defect Programming)



- **The purpose of Inspection is to eliminate defects**
- **Exit criterion for design:**
 - One design statement materializes as 3 to 10 code statements
- **Checklists of typical errors we make**
- **No Unit Test - Developer does not run software !**
- **Testing:**
 - Finding as many of the remaining defects as possible
 - Too many errors discovered
 - previous steps are not being done properly
 - redo previous steps (do not “repair”)

Testing in Cleanroom

- **Testing is an important part of the process, but it is done only after verification is successfully completed**
 - **Testing is done:**
 - **Primarily to measure quality**
 - **Secondarily to find defects that escaped detection during verification**
 - **Number of bugs per thousand lines of code <10 after verification, compilation and syntax checking**
 - **Very good teams produce 2.3 bugs per kLoC and reject code with 4 or 5 bugs per kLoC**
 - **No attempt is done to try to salvage rejected code by debugging**
 - **The code is sent back to the developers to be rewritten and reverified**
 - **Then it is tested as a completely new product**
 - **Usage based testing**
 - **Risk based testing**
- } **Statistical testing**

Rules for Code

Tick the Code Rule Set

(Miska Hiltunen, 2007)

Extra baggage rules

- DEAD** **Avoid unreachable code**
- DRY** **A comment must not repeat code**
- INTENT** **A comment must either describe the intent of the code or summarize it**
- ONE** **Each line shall contain at most one statement**
- UNIQUE** **Code fragments must be unique**

Tick the Code Rule Set

(Miska Hiltunen, 2007)

Missing info rules

DEFAULT A 'switch' must always have a 'default' clause

ELSE An 'if' always has an 'else'

MAGIC Do not hardcode values

PTHESES Parenthesize amply

TAG Forbidden: marker comments

ACCESS Variables must have access routines

HIDE Direct access to global and member variables is forbidden

Tick the Code Rule Set

(Miska Hiltunen, 2007)

Chaos-inducers

- CALL** Call subroutines where feasible
- NAME** Bad names make code bad
- RETURN** Each routine shall contain exactly one 'return'
- SIMPLE** Code must be simple
- FAR** Keep related actions together
- DEEP** Avoid deep nesting
- FOCUS** A routine shall do one and only one thing

Tick the Code Rule Set

(Miska Hiltunen, 2007)

Risky assumptions

- CHECK-IN** Each routine shall check its input data
- NEVERNUL** Never access a 'NULL' pointer or reference
- NULL** Set freed or invalid pointers to 'NULL'
- CONST 1ST** Put constants on the left side in comparisons
- ZERO** Never divide by zero
- PLOCAL** Never return a reference or pointer to local data
- ARRAY** Array accesses shall be within the array
- VERIFY** Setter must check the value for validity

Tick the Code Speed

(Miska Hiltunen, 2007)

Rule	Call	Check-In	Dead	Deep	Default	Dry	Else
Ticks/hr	46	82	45	76	11	53	322
Rule	Hide	Magic	Name	NotNull	Tag	Unique	
Ticks/hr	186	516	93	90	18	20	

- **Average number of ticks found per hour per rule**
- **Software developers could find this many violations in one hour in the code they produce**
- **144 developers checked for 108h to create the data**

Draft Rule Set for Java

(Sybren Stüvel, 2007)

- SIMPLE** Code should be as simple as possible, but not simpler
- DOCUMENT** Documentation should be such that a developer who's unfamiliar with the code can still understand the reasoning behind it
- CORRECT** Naming and documentation must be correct
- CONDITIONAL CORE** Core functionality of a method should be outside any conditional block
- EARLY RETURN** Return as soon as you can from a method. Assigning to a temporary variable and returning that variable usually results in overly complex code
- EXCEPTIONS** Use exceptions to signal an error condition
Don't return null to signify an error

Draft Rule Set for Java

(Sybren Stüvel, 2007)

- REUSE** Use common library functions where applicable
At least take a look at StringUtils and ListUtils (Spring framework) and ArrayUtils (Apache Commons)
Use XStream for parsing and generating XML
- EQUALS** To compare objects use their equals method
- MAGIC** Define constants in one place, and use them
- REFER** Use @see and @link in Javadoc to refer readers to relevant other locations
- READABLE** Ensure the code is easily readable
- SENSIBLE** Test values should be sensible
- TEST VALUES**
- EARLY JAVADOC** Write a method's JavaDoc before writing actual code. This gives a method its scope
- REVIEW TESTS** Start by reviewing the unit tests

MISRA C

- **MISRA: Motor Industry Software Reliability Association**
- **MISRA C (1998) has 127 rules**
- **Providing a set of guidelines to restrict features in the ISO C language of known undefined or otherwise dangerous behaviour**
- **Of these, 93 are required and the remaining 34 are advisory**
 - **Rule 104 (required): Non-constant pointers to functions shall not be used**

Version	Rules	Sections	Pages
MISRA C 1998	127	17	69
MISRA C 2004	141	21	111

MISRA C

Rule 59 (required): The statement forming the body of an "if", "else if", "else", "while", "do ... while", or "for" statement shall always be enclosed in braces

```
if (x == 0)
{
y = 10;
z = 0;
}
else
y = 20;
z = 1;
```



MISRA C

Rule 33 (required):
The right hand side of a
"&&" or "||" operator
shall not contain side effects

```
if ((x == y) || (*p++ == z))
{
/* do something */
}
```

```
if (x == y)
{
doSomething = 1;
}
else if (*p++ == z)
{
doSomething = 1;
}

if (doSomething)
{
/* do something */
}
```

MISRA C

Motor Industry Software Reliability Association

`a[i] = ++i;` happens once in every 7,000 lines in C

```
c == d;
```

```
if (c=d)
{
}
```

Put on checklist

Cleanroom Inspections

Cleanroom expectations

NASA Satellite control system

- 40kLoC FORTRAN
- Testing found 4.5 defect/kLoC
- 60% of programs compiled successfully first time

IBM decision support program

- 107kLoC various languages, 50 person team
- Testing found 2.6 defect/kLoC
- 5 of 8 components: no defects found, no defects found in use

IBM tape drive controller, real time data stream control

- 86kLoC, C-code, 50 person
- Testing found 1.2 defect/kLoC

Ericsson Telecom operating system

- 350kLoC, assembler and C, 70 person, 18 months
- Testing found 1 defect/kLoC

Cleanroom benefits

- **Zero failures in field use**
- **Short development cycles**
- **Long product life**

Quality is cheaper

Cleanroom plans

Software development plan

1. Project mission plan
2. Project organization plan
3. Work product plan
4. Schedule and resource plan
5. Measurement plan
6. Reuse analysis plan
7. Risk analysis plan
8. Standards plan
9. Training plan
10. Configuration management plan



Cleanroom specification processes

- **Requirements analysis process** **CMM-2**
 - Software requirements
- **Function specification process** **CMM-3**
 - Function specification
(black box, state box, clear box)
- **Usage specification process** **CMM-2**
 - Usage specification
- **Architecture specification process** **CMM-3**
 - Software architecture
- **Increment planning process** **CMM-2**
 - Increment construction plan

Cleanroom development processes

- **Software reengineering process** **CMM-3**
 - Reengineering plan
 - Reengineered software
- **Increment design process** **CMM-2**
 - Increment design
- **Correctness verification process** **CMM-3**
 - Increment verification reports
- **Architecture specification process** **CMM-3**
 - Software architecture

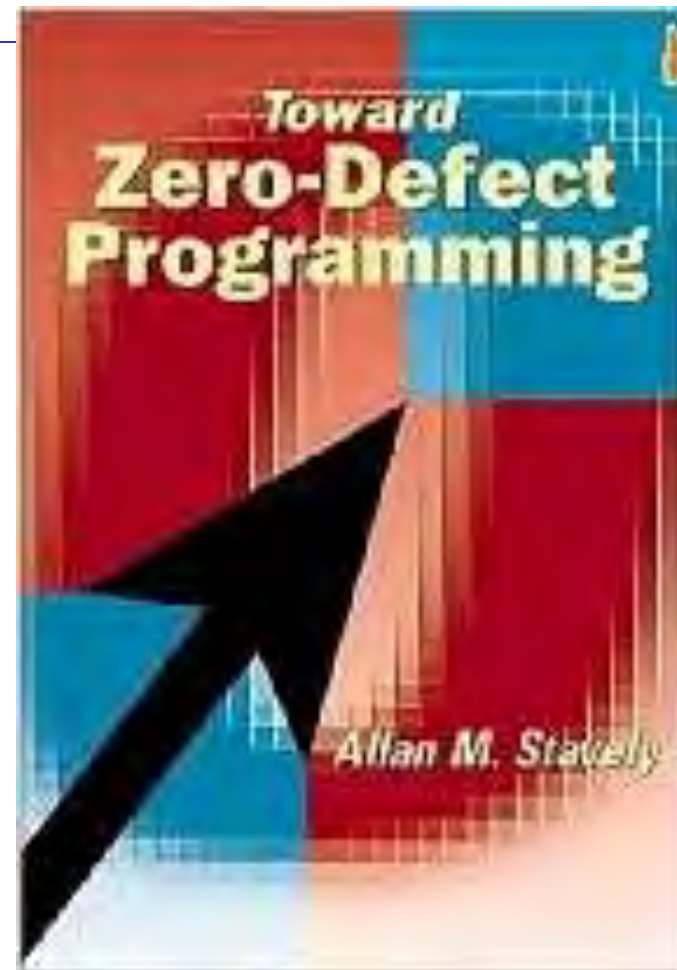
Cleanroom certification processes

- **Usage modelling and test planning process**
 - Usage models (abuse models)
 - Increment test plan
 - Statistical test cases
- **Statistical testing and certification process**
 - Executable system
 - Statistical testing reports
 - Increment certification reports

Cleanroom

Allan M. Stavely:
Toward Zero Defect Programming

There are more books, but Stavely explains it very pragmatic



Cleanroom Principles

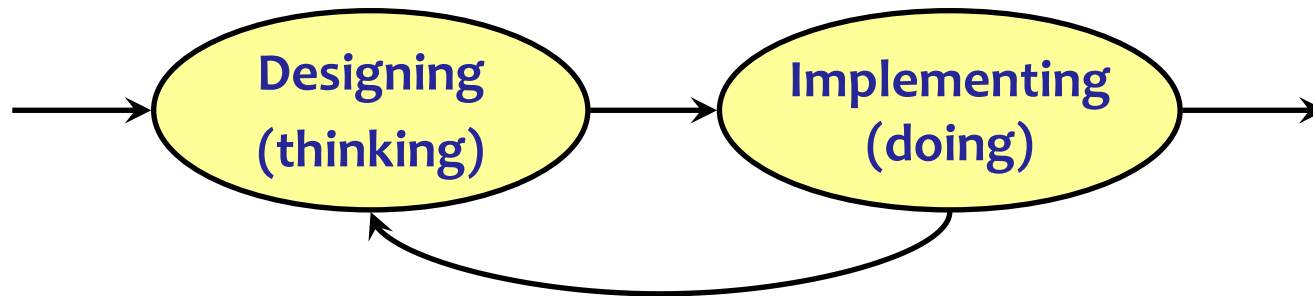
- **Incremental development**
 - User verifiable increments
- **Team organisation**
 - 4~8 people
- **Formal methods of specification and design**
 - Level of formalism varies even within project
- **Intense review**
 - Mathematical proof of correctness
 - Verifying individual control structures
- **No unit test**
 - No testing infinite number of paths, infinite combination of data
- **Statistical testing as reliability measurement**
 - Testing is not suitable for bug-hunting

Cleanroom Inspections

- **The purpose of Inspection is to eliminate defects**
- **Exit criterion for design:**
 - One design statement materializes as 3 to 10 code statements
- **Checklists of typical errors we make**
- **No Unit Test - Developer does not 'try' software !**
- **Testing:**
 - Finding as many of the remaining defects as possible
 - Too many errors discovered
 - previous steps are not being done properly
 - redo previous steps (do not "repair")

Getting stuck somewhere ?

- **Getting stuck in implementation? Back to the design !**

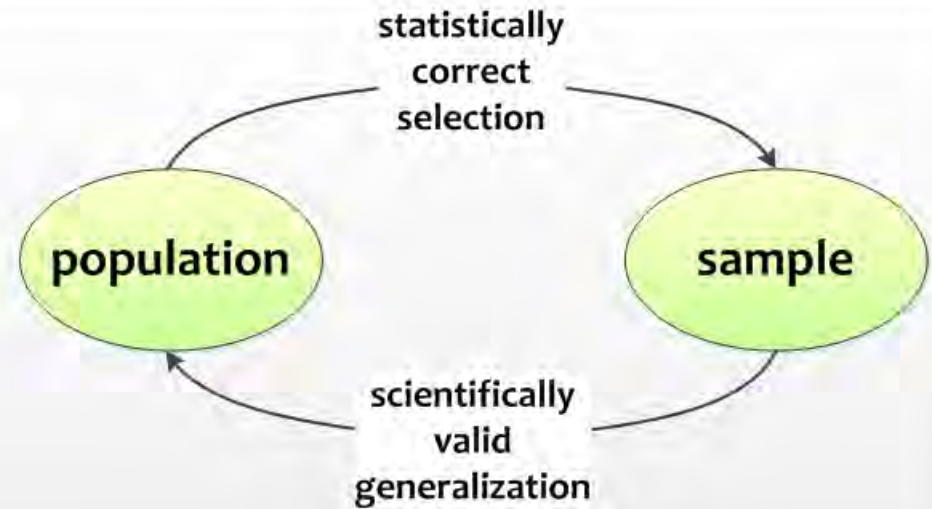


- **Getting stuck in Inspection? Back to the design !**
- **Getting stuck in Testing? Back to the design !**
- **Why do we get stuck ?**
- **Root cause analysis !**

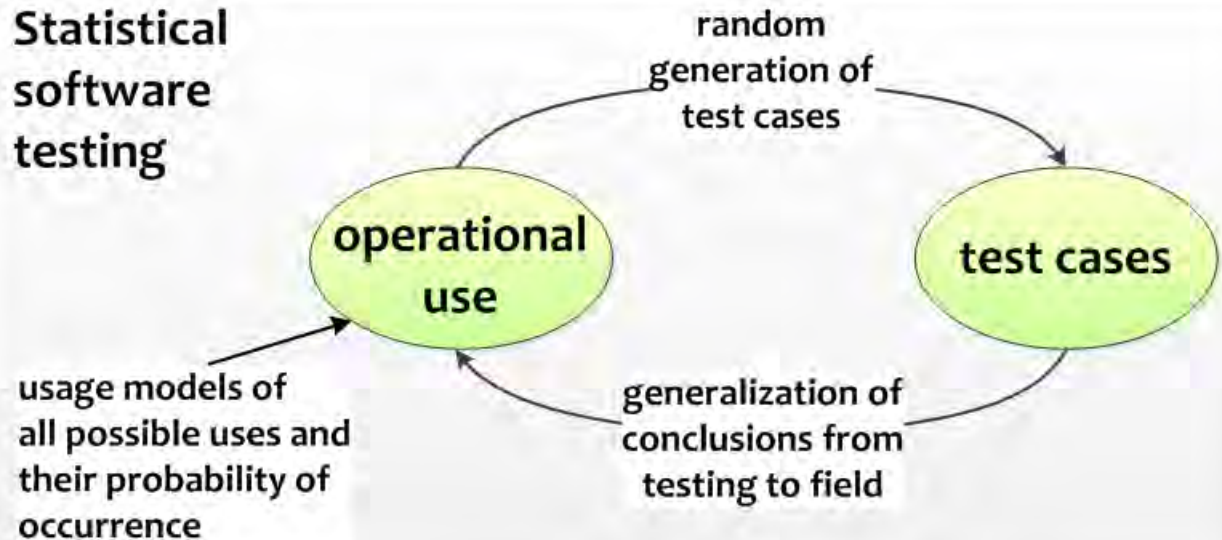
Statistical Testing

You need also other forms of testing!

Statistical experiment



Statistical software testing



Cleanroom fundamentals

- **Design principle**
 - Designers can and should produce systems free of defects *before testing*
- **Testing principle**
 - The purpose of testing is to *measure* quality
- **Main development model**
 - Incremental (Cleanroom)/evolutionary (Gilb)/cyclic (TSP)
 - Each increment is a working subset of the final product
 - Stable requirements for each increment
 - No eleventh hour integration

(1-4) Project Planning, Project Management, Performance Improvement, and Engineering Change

(5) Architecture Specification

Full Cleanroom Process Cycle

Analysis/Specification Process Cycle

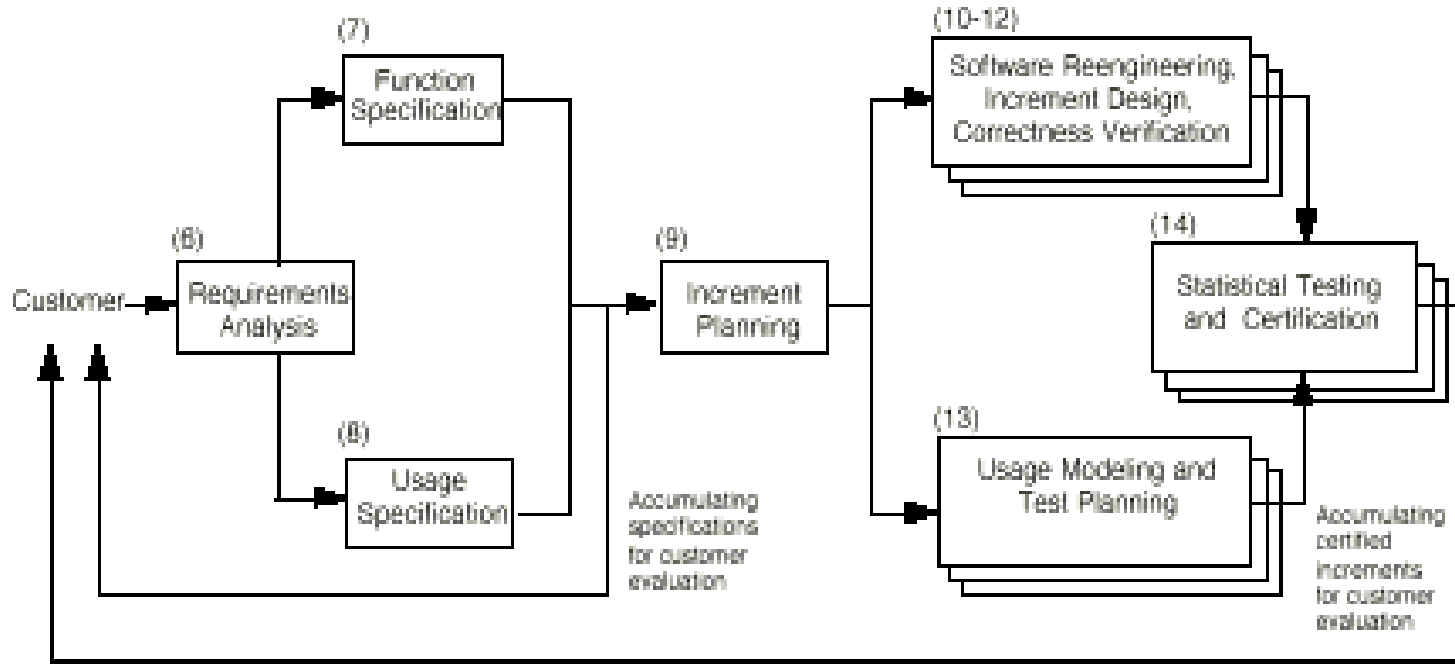


Figure 1. Cleanroom Process Flow

Philosophy behind Cleanroom

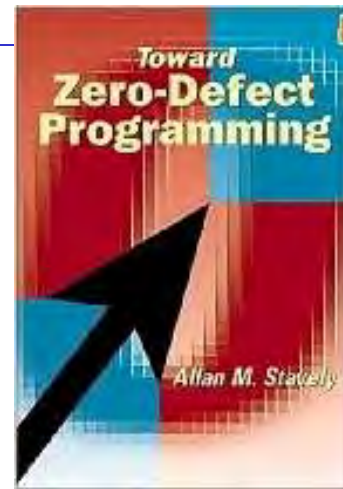
- To avoid dependence on costly defect-removal processes
- By writing code increments right the first time and
- Verifying their correctness *before* testing

(Linger, 1994)

Cleanroom Software Development

- Design (Mathematical proof)
- Verification (by others)
- Implementation
- Verification (by others)
- No unit test
- Only Integration Test (by others)
(Test is Running Code)

- *Verification* is for finding defects
- *Testing* is for *not* finding defects



Testing in Cleanroom

- Testing is an important part of the process, but it is done only after verification (by Inspection) is successfully completed
 - Testing is done:
 - Primarily to measure quality
 - Secondarily to find defects that escaped detection during verification
 - Number of bugs per thousand lines of code <10 after verification, compilation and syntax checking
 - Very good teams produce 2.3 bugs per kLoC and reject code with 4 or 5 bugs per kLoC
 - *No attempt is done to try to salvage rejected code by debugging*
 - The code is sent back to the developers to be rewritten and reverified
 - Then it is tested as a completely new product
 - Usage based testing
 - Risk based testing
- } Statistical testing

No Unit Testing in Cleanroom

- **We should avoid any kind of private testing, whether it is unit testing or some other kind**
- **We may experiment for various reasons, but we must resist the temptation to test our actual code**

Rules in Cleanroom

- **Inspect also for attributes like: efficiency, simplicity, clarity, generality, portability, ease of verification, maintainability, ...**
- **People can make suggestions for improvement of any aspect of the program. Valuable ideas will often emerge from the teams discussions**
- **The goal is to produce the best program possible: a program that can be verified with difficulty, but is more complicated than it needs to be, is not good enough**
- **If substantial revision appears necessary, the review process is stopped so that the team does not waste time verifying parts that will be changed anyway**
- **Usually, after some experience, this will rarely happen**
- **In a later meeting, the team will reverify the parts that were changed**

Cleanroom: Slowest reviewer sets the pace

- **Wrong: Does anyone consider this incorrect?**
(dreamers won't answer)
- **Better: Does everybody agree that this is correct?**
(attention is required)
- **A team does not consider a verification condition proven until the slowest person to respond has expressed agreement**

It is important to resist taking shortcuts here

Metrics

Useful Evo metric

Size of the smile on the customers face

- In many cases, the Evo attitude and techniques replace the need for metrics
- I did not say always
- In Evo, we consume metrics immediately for learning, rather than collecting them

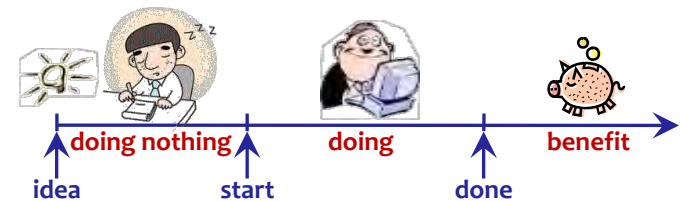
Why would we measure ?

- **Goal**
- **Question**
- **Metric**
- **Consequence**
- **Does it help ?**

Metrics used

- **Metrics for the project**

- Ratio Real time used / Estimated time
- Calibration factor
 $\Sigma \text{ Realized} / \Sigma \text{ Estimations}$
- Predicted date of what will be done when
Today plus Sum of Calibrated Estimations
- Ratio plannable / unplannable hours (default: 2/1)
- Available time, available budget (less is better)
- Cost of one day of delay
 - Cost-Of-Doing-Nothing
 - Project Cost + Lost Benefit



- **Metrics for the product**

- Quantified requirements (ref Planguage - Tom Gilb)
- Rate of improvement on quantified requirements (Impact Estimation)

Metrics techniques used

- **Just-enough metrics** (don't do unnecessary things)
 - Maximizing Return-on-Investment and Value Delivered
- **Consuming the metrics immediately**
 - Not putting them in Databases
 - Using immediately for learning and improving
 - Feeding intuition to come up with better estimations
 - Preventing failure
- **Time-boxing** (not Feature-boxing)
 - Minimizes the need for tracking
- **Calibration**
 - Coarse metrics provide accurate predictions (Law of Large Numbers)
 - Moving Sense of Urgency from the end towards now

Estimation techniques used

- **Just-enough estimation** (don't do unnecessary things)
 - Maximizing Return-on-Investment and Value Delivered
- **Changing from optimistic to realistic predictions**
 - Estimation of Tasks in the TaskCycle
 - Prediction what will be done when in TimeLine
- **0th order estimations** (ball-park figures)
 - For decision-making in Business Case and Design
- **Simple Delphi**
 - For estimating longer periods of time in TimeLine
 - For duration of several (15 or more) elements of work
- **Simpler Delphi**
 - Same, but for quicker insight
 - Recently added by practice
- **Doing something about it** (if we don't like what we see)
 - Taking the consequence
 - Saving time

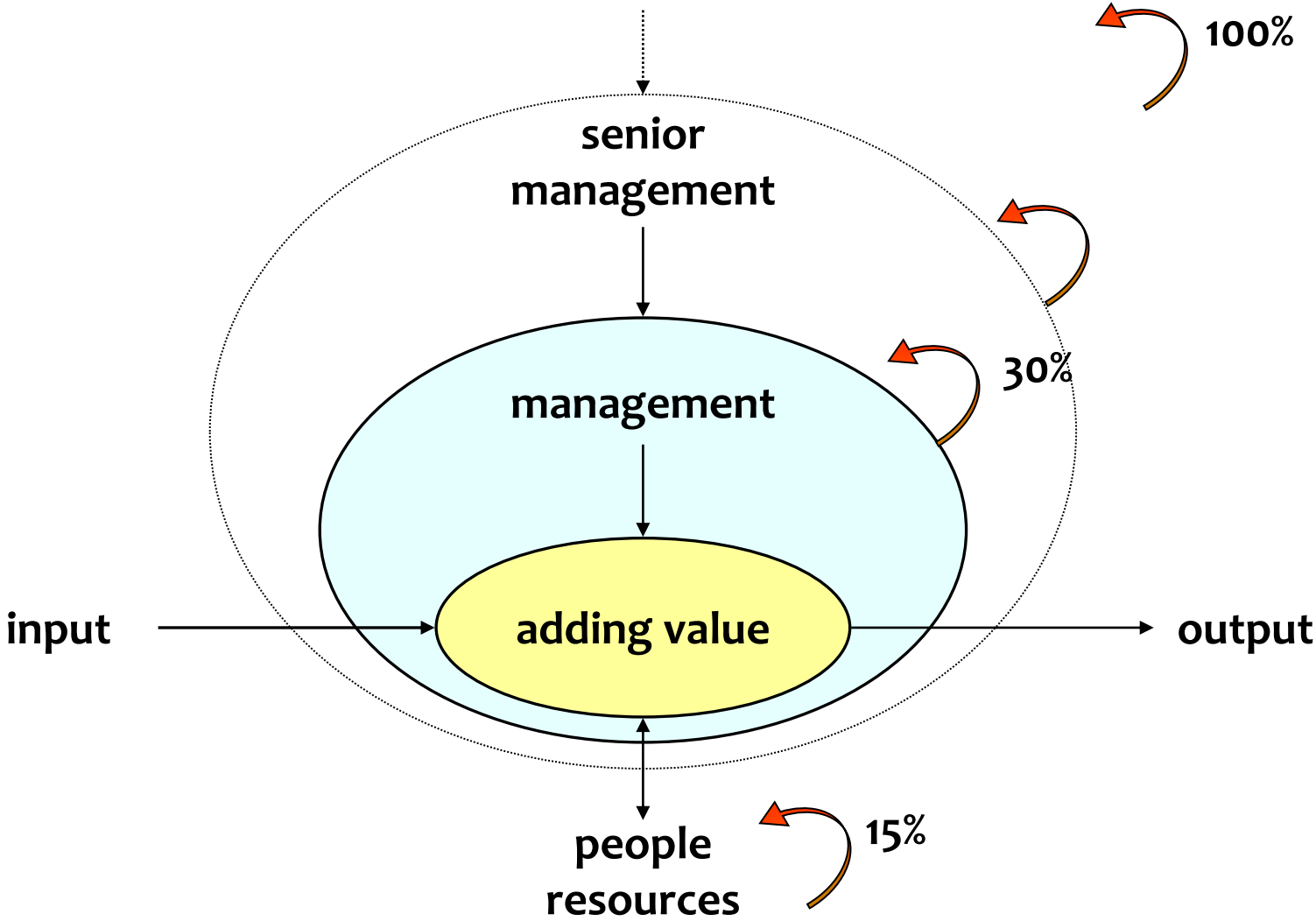


Management Issues

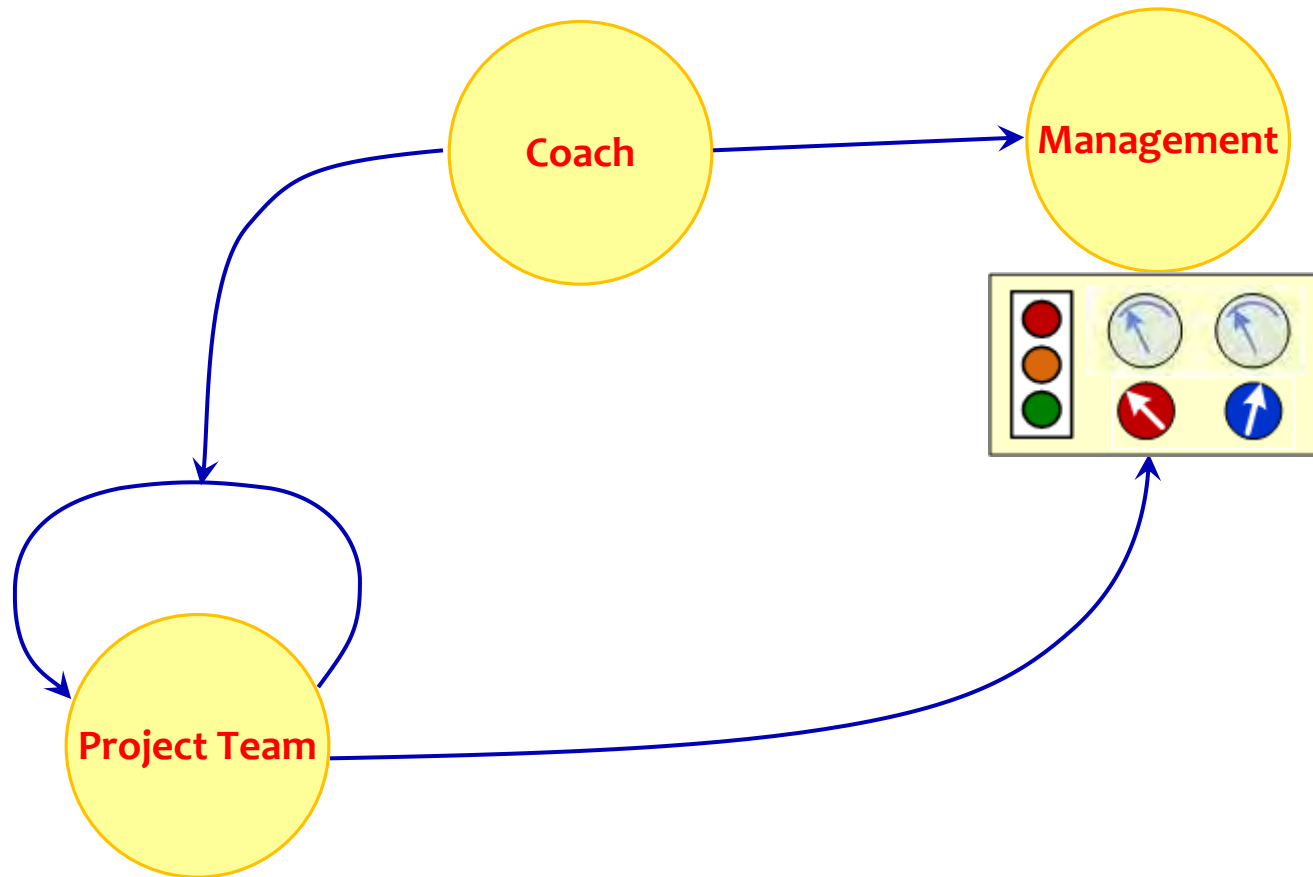
Managers have to learn

- **Managers should be coaches**
- **Not police**
- **Managers have to learn to understand the Evo approach**

Simple model of Management



Local Loop Principle



Management Questions on Tasks

- **Is the Project under Control?**
- **Show me !**
 - No “holes” in OK’s
 - All available, plannable time planned
 - TaskSheets used
 - Results used
 - Prompt explanation in case of discrepancies

Introduction

Issues

How Lean is Evo ?

- **Kaizen** - PDCA, reflection
- **Waste** - Only do the most important things, constantly seeking to do less, without doing too little. Reflection
- **Value** - Only produce Value, quantifying Value, increasing Value every Evo step
- **Pull** - Who's waiting for it? Defer to last moment
- **Poka Yoke** - Murphy
- **Etc**

- **Evo** fills in the Lean principles with pragmatic action
- **Lean** ← **Toyota** ← **Ford** ← **Benjamin Franklin**

Order of Introduction

- **Evo-day**
- **Tasks**
- **Deliveries**
- **TimeLine**
 - Sense of urgency
 - Value delivered
 - Better performance pays salaries
- **Requirements engineering details**
- **Reviews & Inspections**

Evolutionary start pattern

- **Evo day**
 - Explanation of the Evo approach
 - Organizing the work of the coming week
 - Goal: at the end of the day, people of the team know what they are going to work on, what not, and why
- **Weekly Evo day**
 - Execution of the 3-step procedure

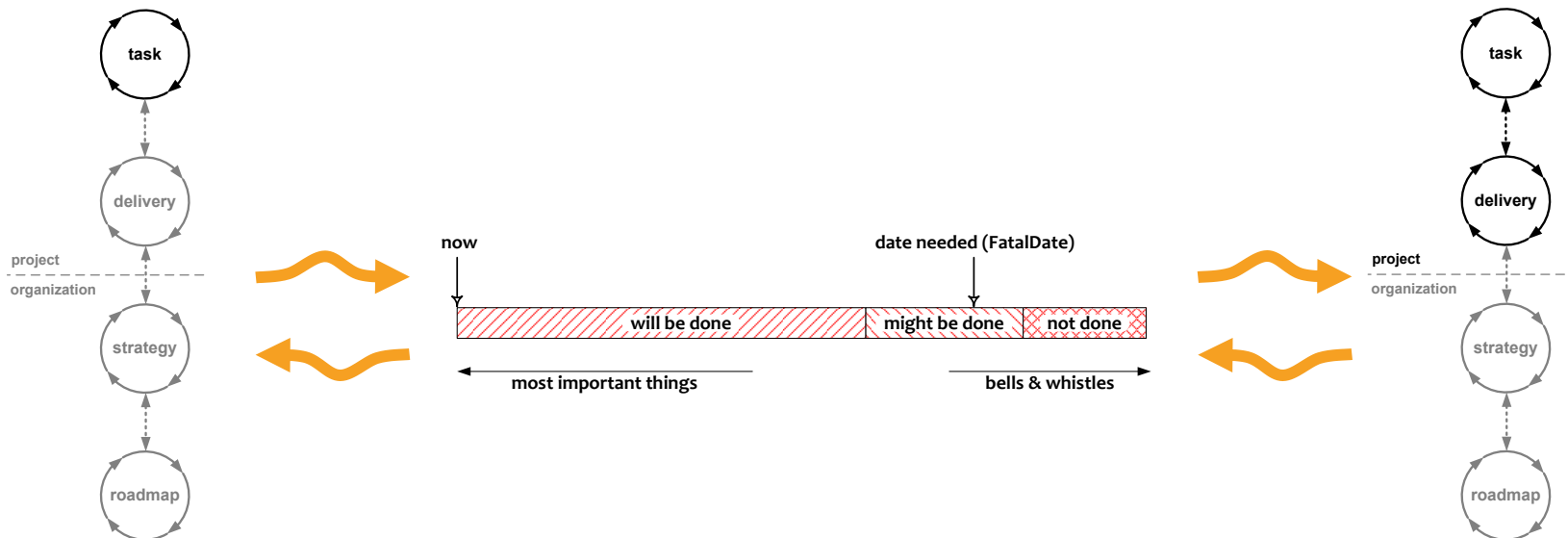
Evolutionary introduction pattern

- 1. Introducing Tasks**
How to organize the work
- 2. Introducing TimeLine**
The design of the project
- 3. Introducing Deliveries**
Focusing on Results

→ Short term view

→ Longer term view

→ Connecting long and short



Cases

Case 7: A “failure”

- **Seasoned project manager: “Good idea, but...”**
- **No emphasis on TimeBoxing**
- **Didn’t try to understand Delivery and TimeLine concepts**
- **Many “hero’s” in the team**
 - I can do whatever I want. I know so much, they won’t fire me anyway
- **No Sense of Urgency both in team and from management**
 - Management by fear
 - Management asks different things every week
 - Management asks impossible results

If you don’t apply Evo, Evo does not fail, the project does

Case 9: US company

- **Started with 10 people of a 40 people project** (don't over-eat)
- **Carefully *designed* the Evolutionary Introduction of Evo**
- **Now the whole team is routinely working the Evo way**
- **Including 8 people in India**
- **Didn't miss a milestone since**
(Average time overrun before Evo was 20%)
- **They still hardly believe this is possible**

Evo works with larger and distributed projects

Case 10: Managers

- **Managers asked**
“Can I use this for my own busy schedule?”
 - Write down what you have to do
 - Add effort hours
 - List in order of priority
 - Check how much time available this week
 - Draw line at 2/3 of the available time
 - Decide what to do and what not to do
- **Manager Reports:**
“This made me 40% more productive immediately!”

Finally

Magic words

- **Focus**
- **Priority**
- **Synchronize**
- **Why**
- **Dates are sacred**
- **Done**
- **Bug, debug**
- **Discipline**

Magic Sentences

- **Customer may never find out about our problems**
- **Evo metric: Size of the smile of the customer**
- **Delivery Commitments are always met**
- **People tend to do more than necessary**
- **Can we do less, without doing too little**
- **What the customer wants, he cannot afford**
- **Who is waiting for that?**

Why would the *product* need Evo ?

- **We don't know the real requirements**
- **They don't know the real requirements**
- **Together we have to find out** (stop playing macho!)
- **What the customer wants he cannot afford**
- **Is what the customer wants what he needs?**
- **People tend to do more than necessary especially if they don't know exactly what to do**

**If time, money, resources are limited,
we should not overrun the budgets**

Why would the *project* need Evo?

- **Are we effective?** (producing Results)
- **Are we efficient?** (optimally using the available time)
- **Are we actively learning from our mistakes?** (PDCA)
- **How do we estimate, plan and track progress?**
- **How do we handle interruptions?**
- **Did we learn from feedback per project?** (project evaluation)

When would we **not** need Evo

- **Requirements are completely clear, nothing will change: use waterfall** (= production)
- **Requirements can be easily met with the available resources, within the available time** (Still, Evo can make it faster)
- **Everybody knows exactly what to do**
- **Customer can wait until you are ready**
- **Management doesn't know what to do with the time saved**
- **No Sense of Urgency**

Use Evo only on projects you want to succeed

My project is different

- **On every project somebody will claim:**
**“Nice story, but my project is different.
It cannot be cut into very short cycles”**
- **On every project, it takes less than an hour (usually less than 10 minutes) to define the first short deliveries**
- **This is one of the more difficult issues of Evo
We must learn to turn a switch
Coaching helps to turn the switch**

www.malotaux.nl/Booklets

More

- 1 **Evolutionary Project Management Methods (2001)**
Issues to solve, and first experience with the Evo Planning approach
- 2 **How Quality is Assured by Evolutionary Methods (2004)**
After a lot more experience: rather mature Evo Planning process
- 3 **Optimizing the Contribution of Testing to Project Success (2005)**
How Testing fits in
- 3a **Optimizing Quality Assurance for Better Results (2005)**
Same as Booklet 3, but for non-software projects
- 4 **Controlling Project Risk by Design (2006)**
How the Evo approach solves Risk by Design (by process)
- 5 **TimeLine: How to Get and Keep Control over Longer Periods of Time (2007)**
Replaced by Booklet 7, except for the step-by-step TimeLine procedure
- 6 **Human Behavior in Projects (APCOSE 2008)**
Human Behavioral aspects of Projects
- 7 **How to Achieve the Most Important Requirement (2008)**
Planning of longer periods of time, what to do if you don't have enough time
- 8 **Help ! We have a QA Problem ! (2009)**
Use of TimeLine technique: How we solved a 6 month backlog in 9 weeks
- RS **Measurable Value with Agile (Ryan Shriver - 2009)**
Use of Evo Requirements and Prioritizing principles

www.malotaux.nl/nrm/Insp

Inspection pages

What now ?

Schedule

October	Wed 13	Thu 14	Fri 15
09:00~10:30		1:30	
break		0:10	
10:40~11:40		1:00	
break		0:10	
11:50~12:50		1:00	
lunch		0:40	
13:30~14:30		1:00	
break		0:10	
14:40~15:40		1:00	
break		0:10	
15:50~16:50		1:00	