Six Sigma: A Journey Toward an Empirical and Experimental Approach to Software Process Improvement

Dr Radouane Oudrhiri, Systonomy Limited

radouane@systonomy.com

11th Annual CMMI Technology Conference and User Group
November 14-17, 2011 - Denver, Colorado

Topics

- **Systonomy - company overview and approach**
- **Empirical and Experimental approach**
- **Introduction to Six Sigma**
- **Application of Six Sigma to Software process improvement**
  - Six Sigma an Empirical Approach
  - Case Study
- **Conclusions**

Systonomy background

Founded in April 1999, Systonomy is dedicated to the application of Empirical and Experimental Software engineering, Six Sigma and DFSS for IT and Software Development from real-time and embedded systems to Management Information Systems (MIS) including the implementation and integration of COTS, EAI, ERP systems, CRM, Financial Systems etc.

Systonomy has devised a unique Six Sigma and DFSS framework for IT and Software Engineering that is at the forefront of current knowledge and is investing heavily in research into new methods. Our training has been designed from the ground up as an IT/Software Six Sigma and DFSS training programme and is not a superficial modification of manufacturing or transactional Six Sigma. Our adaptive approach offers our clients an innovative and low risk move from defensive strategies to those of growth.

Our Change Managers, Advisors, Engineers, Black Belts, Master Black Belts and Instructors are IT professionals first and statisticians second

Software Engineering Culture

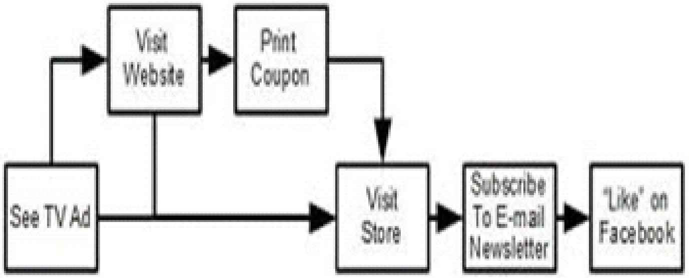
What do these have in common?

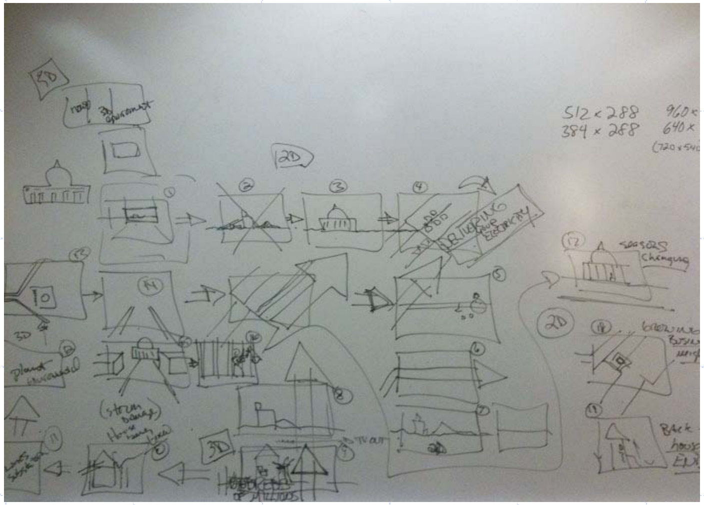
Fiction

Reality

Room

Process

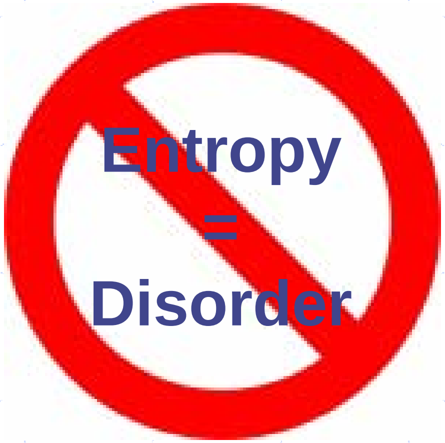




Entropy... Energy dispersal

Spontaneous Processes and Entropy

- **The idea that Entropy = Disorder is an obsolete and misleading concept**



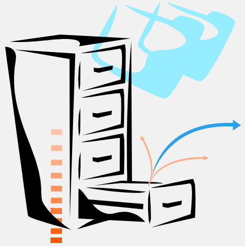
Entropy = Disorder

- **Entropy is about probability of all possible choices, … not whether a given arrangement looks neat or messy.**

- **Spontaneous processes go towards state with the most possible options**
  - "Driven" by simple statistics
  - Random process, not actually driven
- **Entropy is the number of available options**
- **Statistical Definition of Entropy**
  - *S= k ln(W)*
  - *W = # of available states of equal energy*
  - *(Entropy = Delocalization of energy)*

3 Shirts: Red, Blue, Green

Shirts limited to pile in drawer

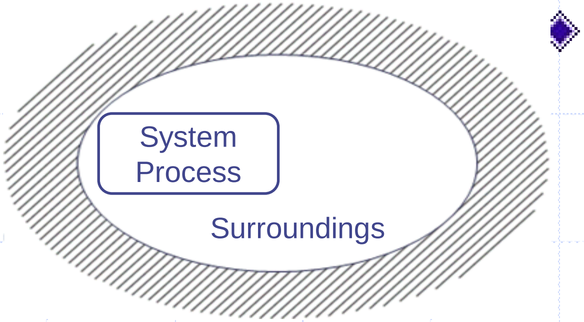


W = 6
red / blue / green,
blue / red / green, etc.

Shirts anywhere in room

W = Lots
red on chair / blue on floor, …

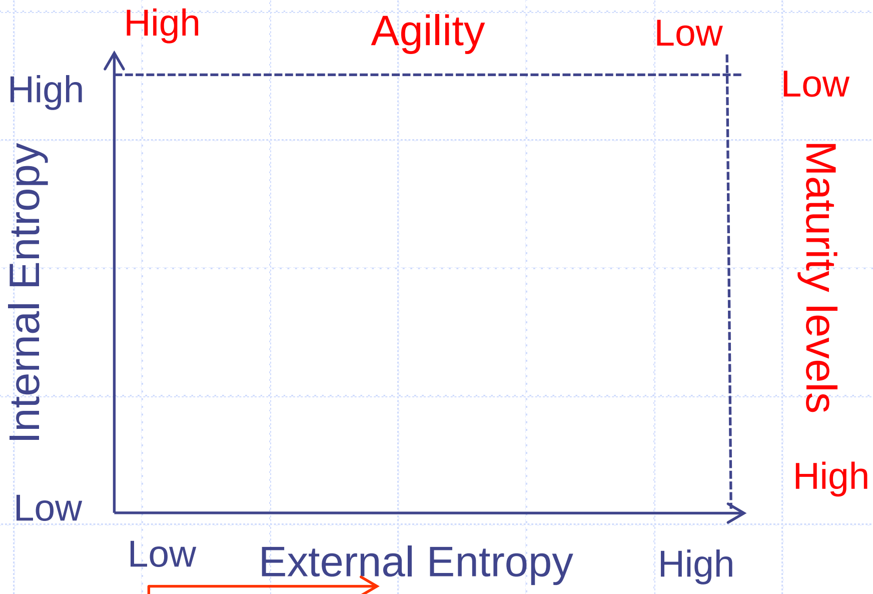
Internal and External Entropies



- **Two types of entropy:**
  - Internal entropy
  - External entropy

- $\text{Entropy}_{\text{Solid}} < \text{Entropy}_{\text{Liquid}} < \text{Entropy}_{\text{Gas}}$
- Increased Rigidity → Decreased Entropy
- Increased # Atoms (elements) → Increased Entropy
- Increased Mass (Complexity) → Increased Entropy

Maturity-based models value more the control of Internal Entropy



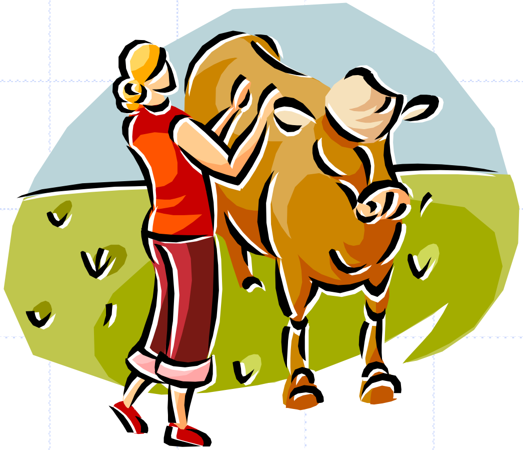
Agile methods value more the understanding of External Entropy

Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

Maturity versus Capability demystified

The Cow Maturity Model^{SM}

The Milking Process



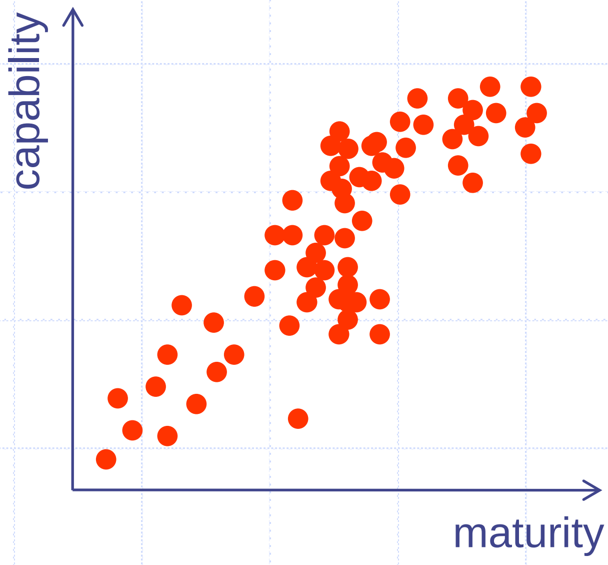
X’s:

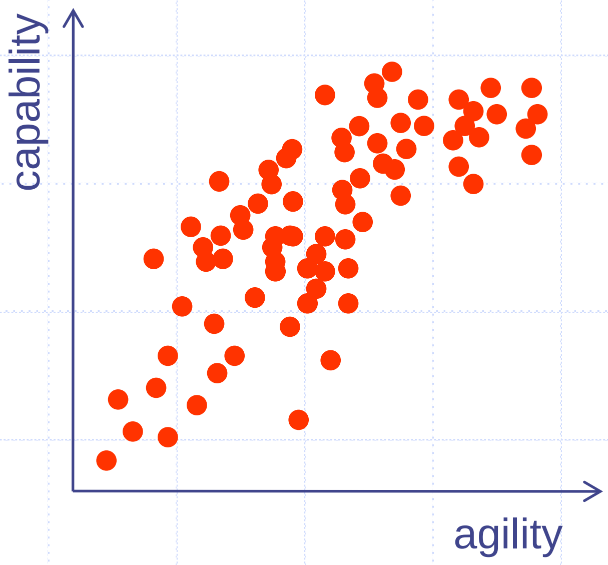
- Quality of the Cows
- Grass, Food
- Massaging cows
- Discipline
- Sophistication

Y’s:

- Litre/Cow/Day
- Density of cream/litre
- Defective milk

Maturity and agility versus Capability





- **Maturity-based models, agile methods and other methods claim some kind of positive relationship between their intrinsic property and Capability (performance)**
  - Maturity and Capability
  - Agility and Capability
  - etc

Superstition is merely the confusion of correlation and causality

- Superstitious behaviour: believes that the action has an immutable cause-and-effect to the outcome, whereas the action might or might not be functional

- "*I behave this way, and I achieve results. Therefore, I must achieve results because I behave this way*."

- Superstition: information accepted on faith, without personal knowledge or examination.

- People pass along "everyone knows" data without questioning it, and others accept the superstition as undeniably true

- Confidence isn't knowledge; .... confidence can *prevent* knowledge and innovation from happening, Unquestioned belief means you never measure, never test, never look at alternatives

You Tube

http://www.youtube.com/watch?v=vGazyH6fQQ4
http://www.youtube.com/watch?v=TtfQlkGwE2U&NR=1

- Skinner Experiment
  - He deprived pigeons of food for a period of time to ensure high responsiveness to anything which resulted in food.
  - Placed them in a cage which delivered a food pellet every 15 seconds, regardless of what the pigeon did or did not do.
  - The pigeon cannot do anything to obtain the food or ensure the supply would continue.
  - The pigeons began recalling their actions before the food was delivered. Some pigeons turned in circles, others tilted their heads, others tapped their feet, others swayed their bodies, and others tossed their heads.
  - The pigeons associated these particular actions with food delivery and began repeating them in the manner of a ritual
- Skinner drew the following conclusions:
  - The pigeon behaves as if there were a causal relation between its behaviour and the supply of food, although such a relation is lacking.
  - A few accidental connections between a ritual and favourable consequences suffice to set up and maintain the behaviour in spite of many unreinforced instances
  - Such a stimulus has reinforcing value and can set up superstitious behaviour.
  - The experiment might be said to demonstrate a sort of superstition.
  - There are many analogies with human behaviour

Examples of Superstitions in software engineering

- **Unfounded opinions and beliefs**
  - Java is better than C or C++ or C#
  - COBOL is an extinct language and not useful for anything
  - Open source languages are free!
  - CMMi is old fashion, Agile is great
  - Agile is not a proper method

- **Requirements Gathering:**

  The requirements gathering process has an inevitable speculative element to it.

Superstitions

Common superstition is that all requirements must be clearly defined before the project can start.

We also know it is the worst time to define all the requirements because it is the furthest point away from the time of use..

- **Project Planning**

  Planning is another inherently speculative activity. In order for planning activities to have any bearing on reality they must be closely and interactively allied to actual outcomes.

Superstition

One of the most pervasive superstition: the best plan is a complete plan, the more detail the better, the more accurate we make our forecasts, the more realistic our plan will be.

The project becomes schizophrenic, it has two independent realities. One is the “reality” of the plan, effort is expended trying to force-fit what really happened into a shape that we pretend it was the same as we thought would happen.

SE - Software Engineering discipline

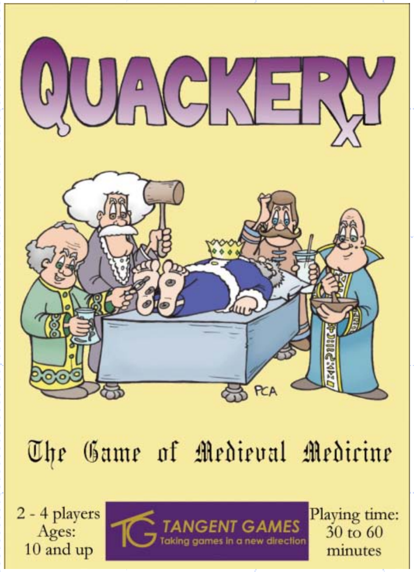
- **The term "Software Engineering" was coined the first time in 1968 at a NATO conference**

- **The discipline of SE is, unfortunately, still in its infancy**

- **SE has reached a stage that is more resembling to *quackery* than engineering**

- **The *modus operandi* of ideas adoption within SE is similar to *fashion industry* rather than true Engineering**
  - The certainty of ideas in SE are judged by whether people use the idea

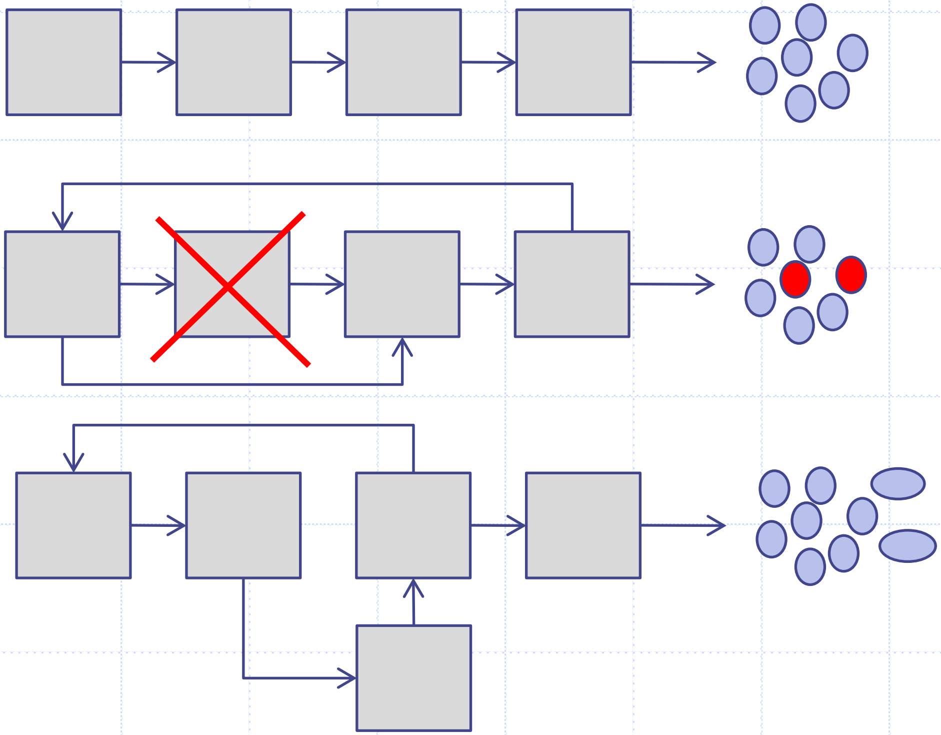
- **SE may be a field whose progress is threatened by analogies and beliefs**
  - Are we sure that our beliefs are true?
  - Which claims made by the software community are valid?
  - Under which circumstances are they valid?



Empirical and Experimental Software Engineering

Empirical Methods

- Empirical methods provide us with insights into how software engineering works in practice and how changes to the process can results in changes to the outcomes of the process (improvements)



[SEA 07]

Empirical Software Engineering

- **Empirical Software Engineering is a discipline which attempts to understand phenomena and at the same time try to change those phenomena in order to improve them.**

- **It is, therefore, about contemplation and action; it has two aims:**
  - Understand how software is actually developed and maintained
  - Understand what improvements should be made to software development (engineering) and how these improvements should be implemented

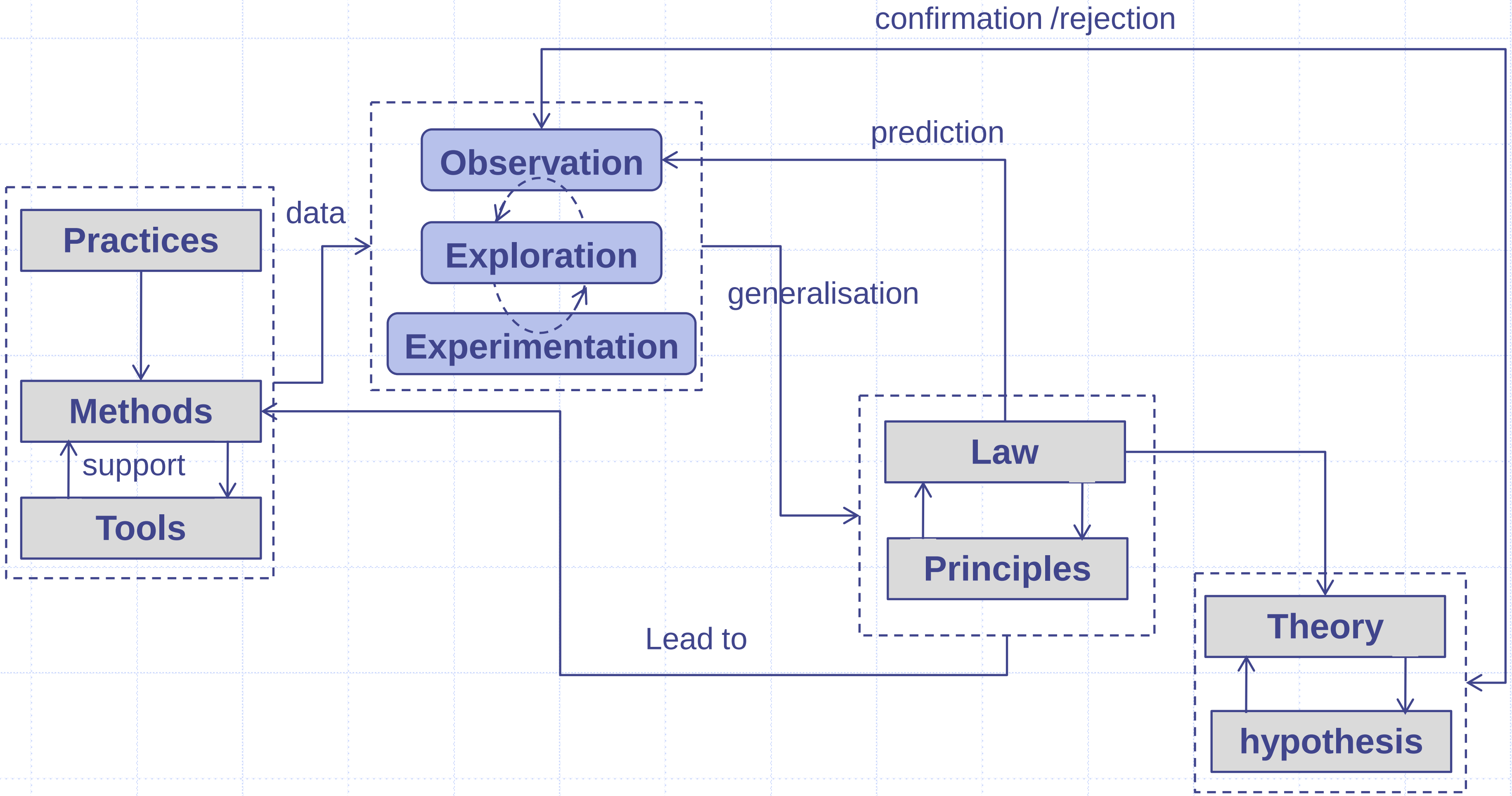
- **It promotes empirical evidence as the primary source of reliable knowledge to achieve these two goals**
  - Exploratory – forming hypothesis
  - Experimental - involves planned experiment designs in to prove causation and not only correlation

Empirical = Exploratory + Experimental

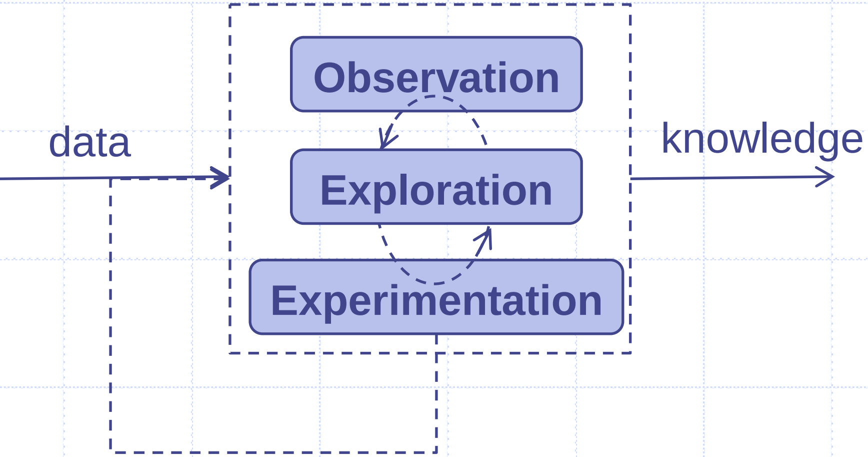
- **It is a necessary technology or a natural approach for goal-oriented, sustained process improvement**

[RAI 07]

Empirical Cycle
Closing the gap between theory and practices

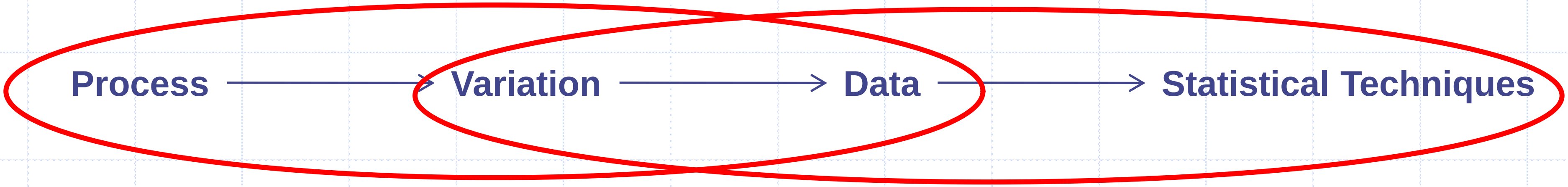


Empirical and Experimental approaches often rely on Statistical Thinking



- **Statistical Thinking**
  - All work is process
  - Variation exist in any process
  - Understanding and reducing variation are keys to success

- **Five fundamental habits are operating simultaneously:**
  - The need for data
  - The importance of data production
  - The omnipresence of variability and uncertainty
  - The measuring and modelling of variability and uncertainty
  - Interpreting results in a context



Statistical Thinking

Statistical Methods

Learning…and Knowledge acquisition

- **The purpose of Empirical and statistical Thinking is Learning and Knowledge acquisition**

> “*Learning is not compulsory…. neither is survival* ”
>
> **W. Edwards Deming**

Empirical and Experimental software engineering is for all professionals who have (and want to keep) a child’s mindset and ask the question *why?*

Software Six Sigma

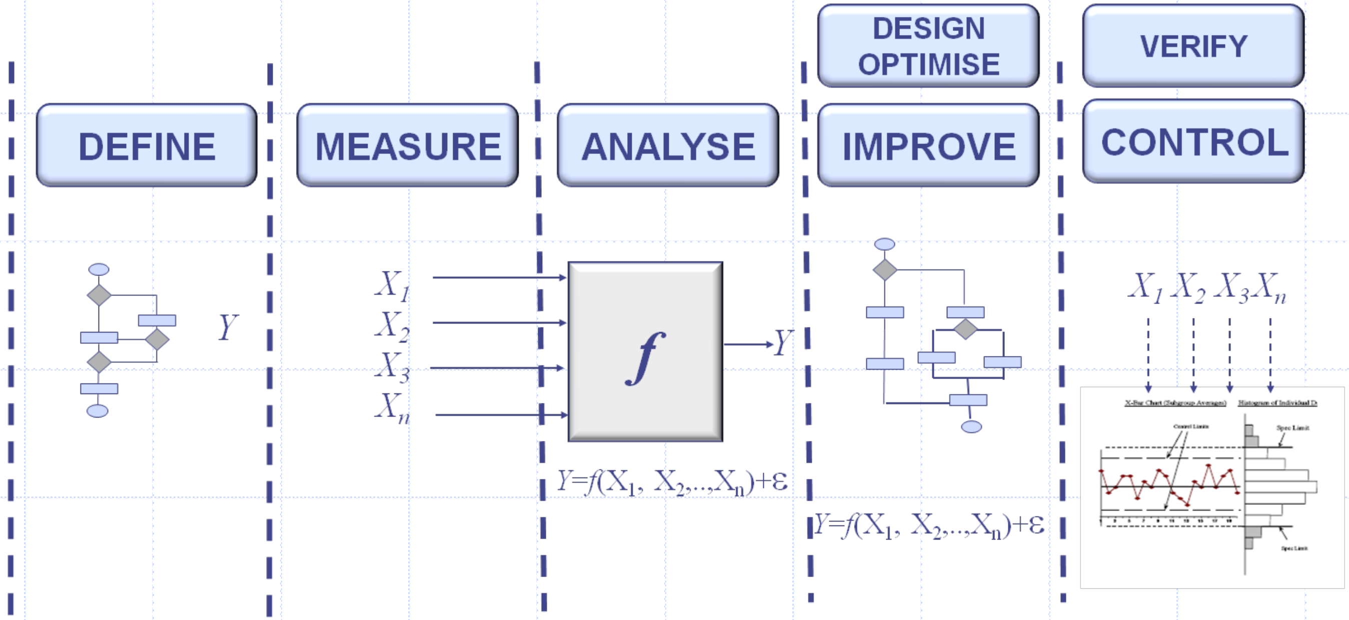
A Problem Solving Methodology

Is Six Sigma another fad?

- **YES... If**
  - It is used as a façade
  - It is used as a label or a branding
  - It is used for “compliance” purposes
  - Statistics are (ab)-used to justify early decisions

- **NO... If**
  - It is used as a real problem solving methodology
  - Improvements solutions are linked to the organisations goals, values and tangible benefits
  - it used to gain insights on our practices

Problem Solving Methodology

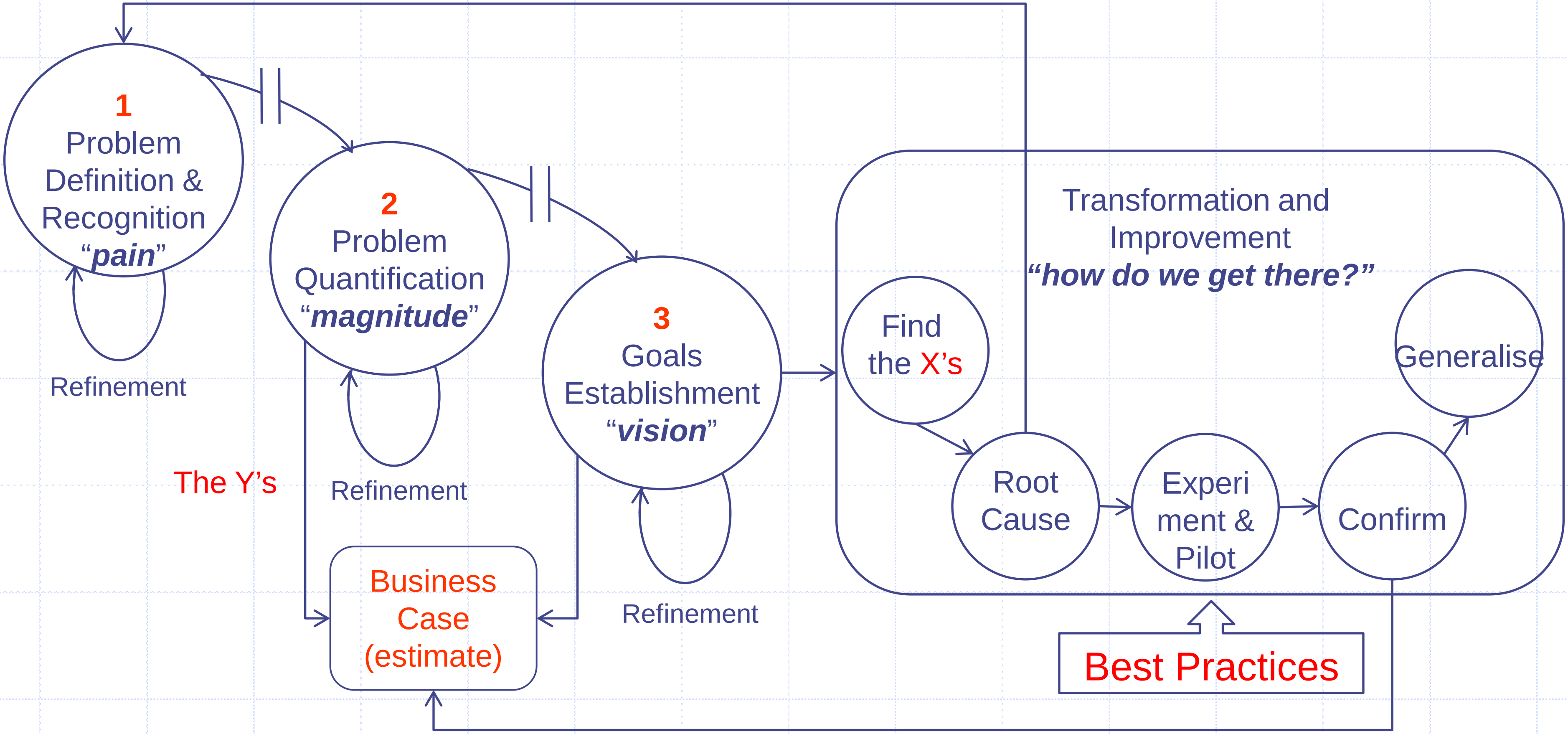


Six Sigma is a pragmatic approach to Empirical and Experimental Process Improvement

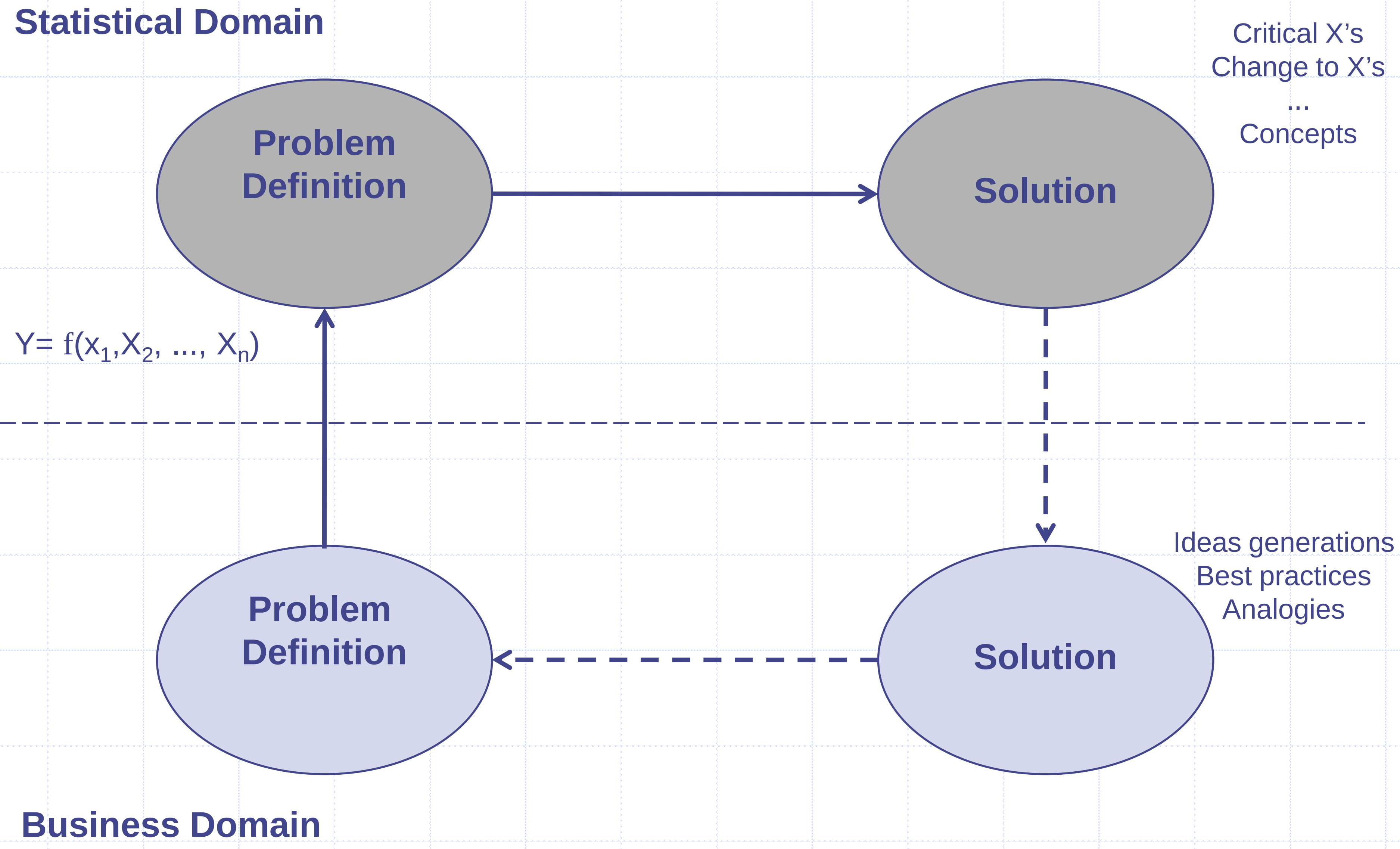
KEYWORDS:

- **Pragmatic: Six Sigma is about solving problems. The focus is on business problems that cause "pain" and extra costs to the organisation**
- **Experimental: Six Sigma is not a catalogue of best practices or methods. Every organisation is different and so are the problems they face. Six Sigma rejects pre-defined solutions and investigates problems to the level of their root causes**

Problem Solving Methodology
A rigorous approach to process improvement



Six Sigma Problem Solving cycle



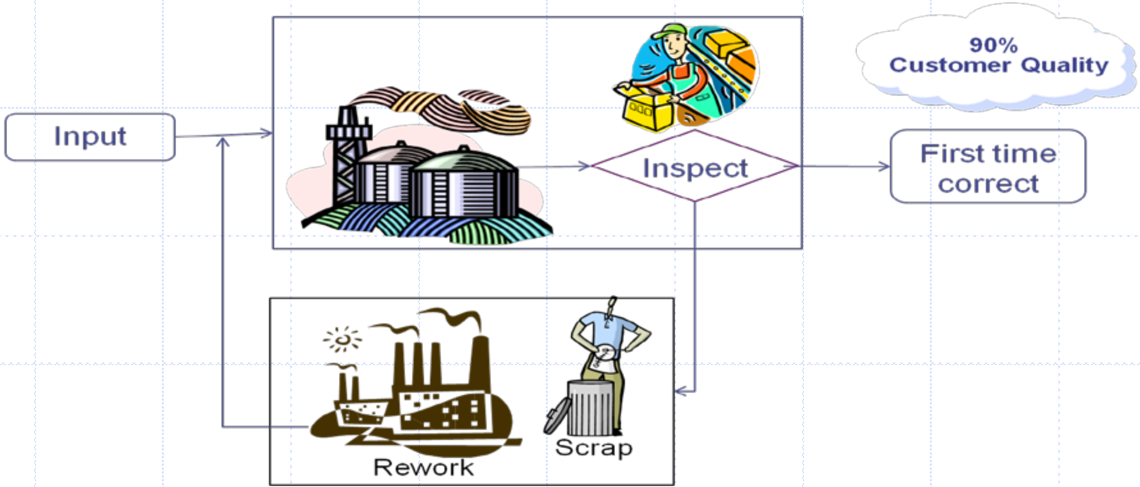
Six Sigma vs. Other SPI approaches

Six Sigma	Catalogue Based Improvement models
Focus on Problems	Focus on Best Practices (Solutions)
Emphasis on measurement of process capability	Emphasis on assessment of process maturity
Business results oriented	Quality improvement oriented
More prescriptive in nature	More descriptive in nature
Improvement “is by experiment”	Improvement is “by the book”
Provides the “how to”: solve a process problem	Provides the “how to”: manage the process according to best practices

The two approaches complement and reinforce each other!

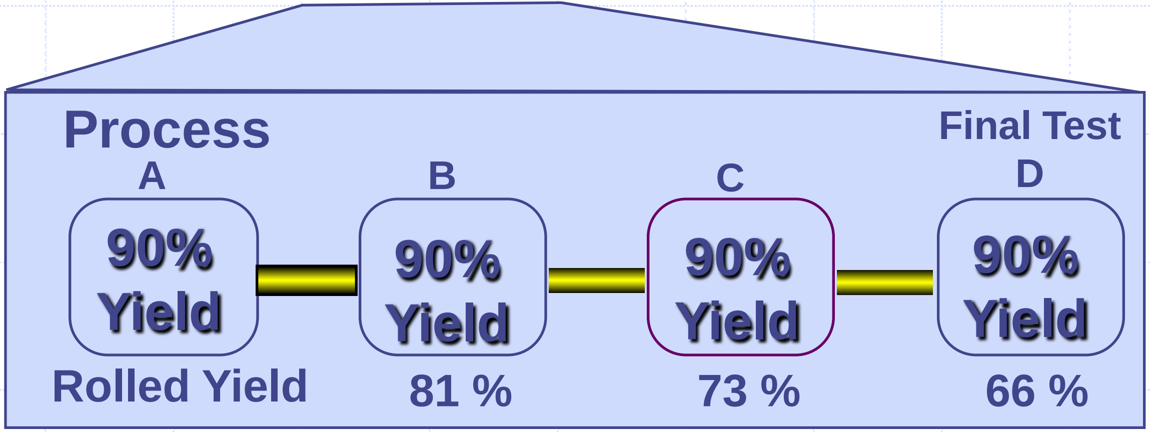
Six Sigma integrates pragmatism into Empirical approaches without loosing the scientific rigour

Key Six Sigma Concept: Hidden Factory



Manufacturing Variation Causes a "Hidden Factory" Increased Cost - Lost Capacity

- **Wasted Time**
- **Wasted Money**
- **Wasted Resources**
- **Wasted Floor space**



Rolled-Throughput Yield

66% ≠ 90%

Classical First-Time Yield

DPMO forces you to look at the “hidden factory” where expediting, rework and delays occur, but would likely not show up in classical yield metrics. The resulting detail from DPMO determinations can then help to prioritize where improvements can be made.

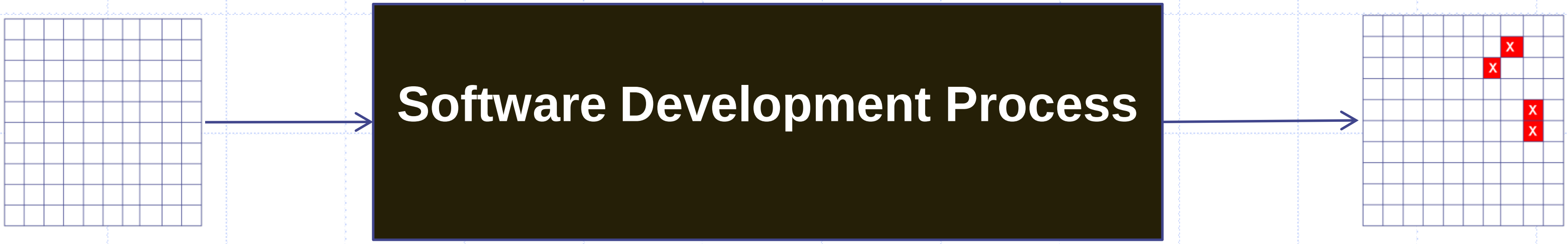
The hidden Factory in Software Process development

Unit of work

- Function Points
- use case points
- . widgets

Delivered unit of work

- Function Points
- use case points
- . widgets

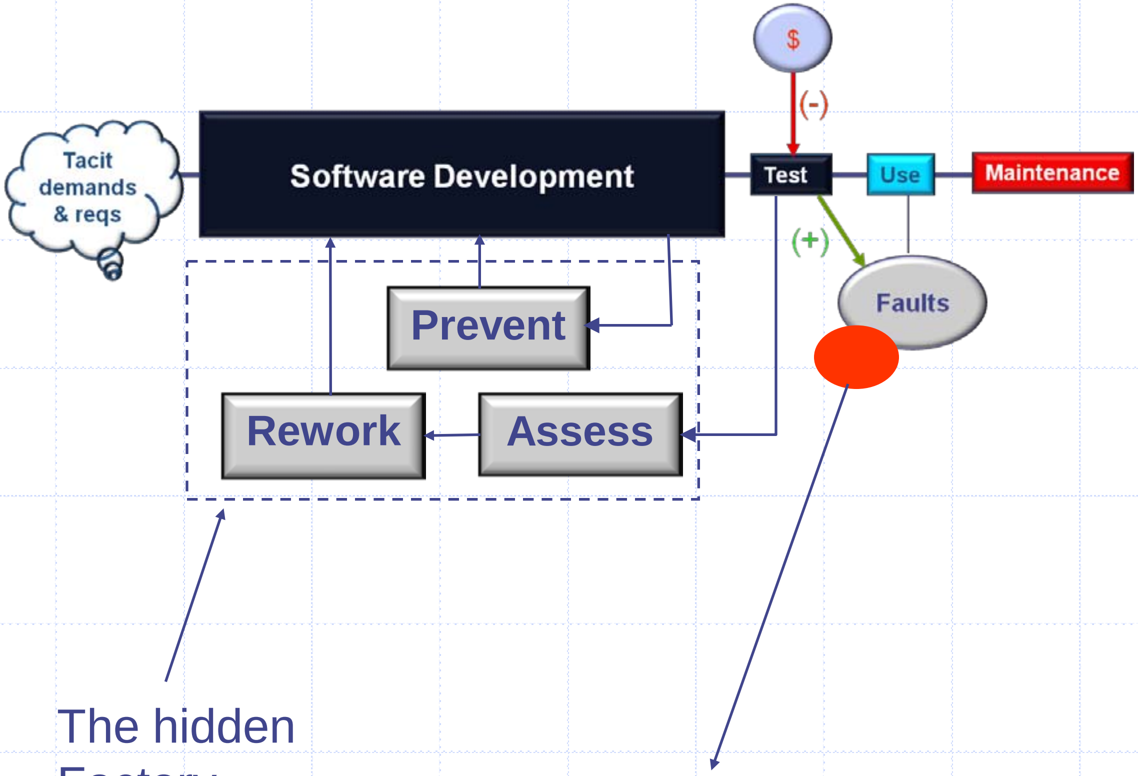


Software Development Process

Area of opportunities (x10^6)

Delivered defects per
area of opportunities (x10^6)

The hidden Factory in Software Process development

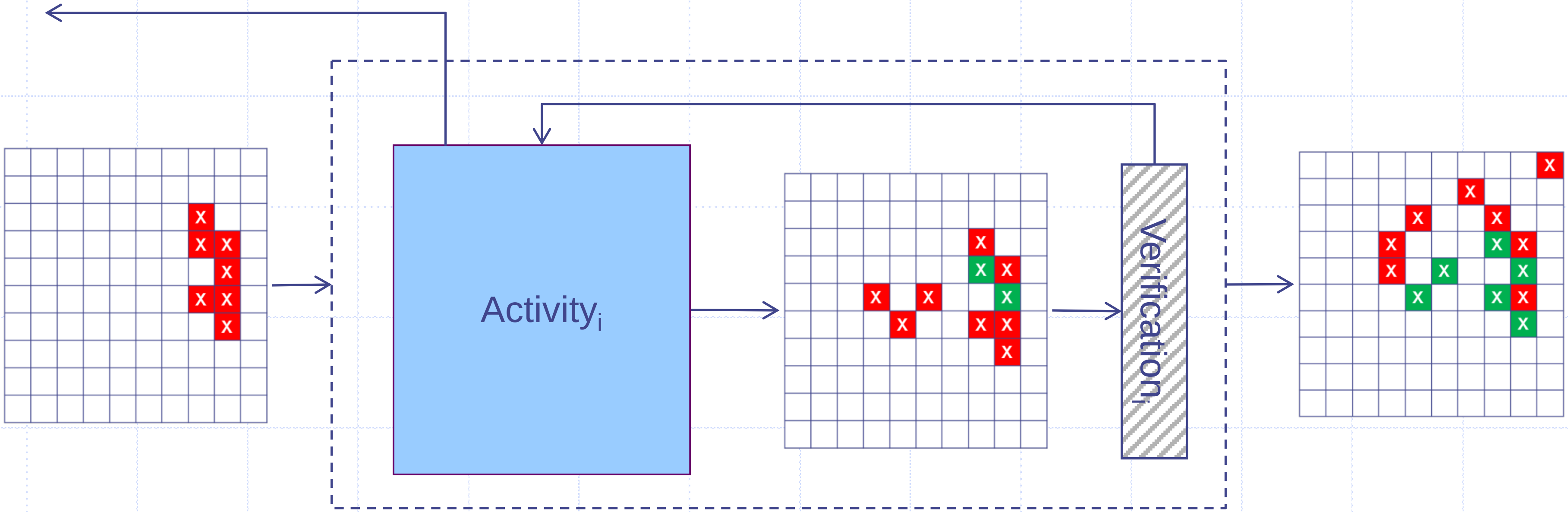


The hidden Factory

$$Yield = \frac{n_{system}}{(n_{system} + n_{leaked})}$$

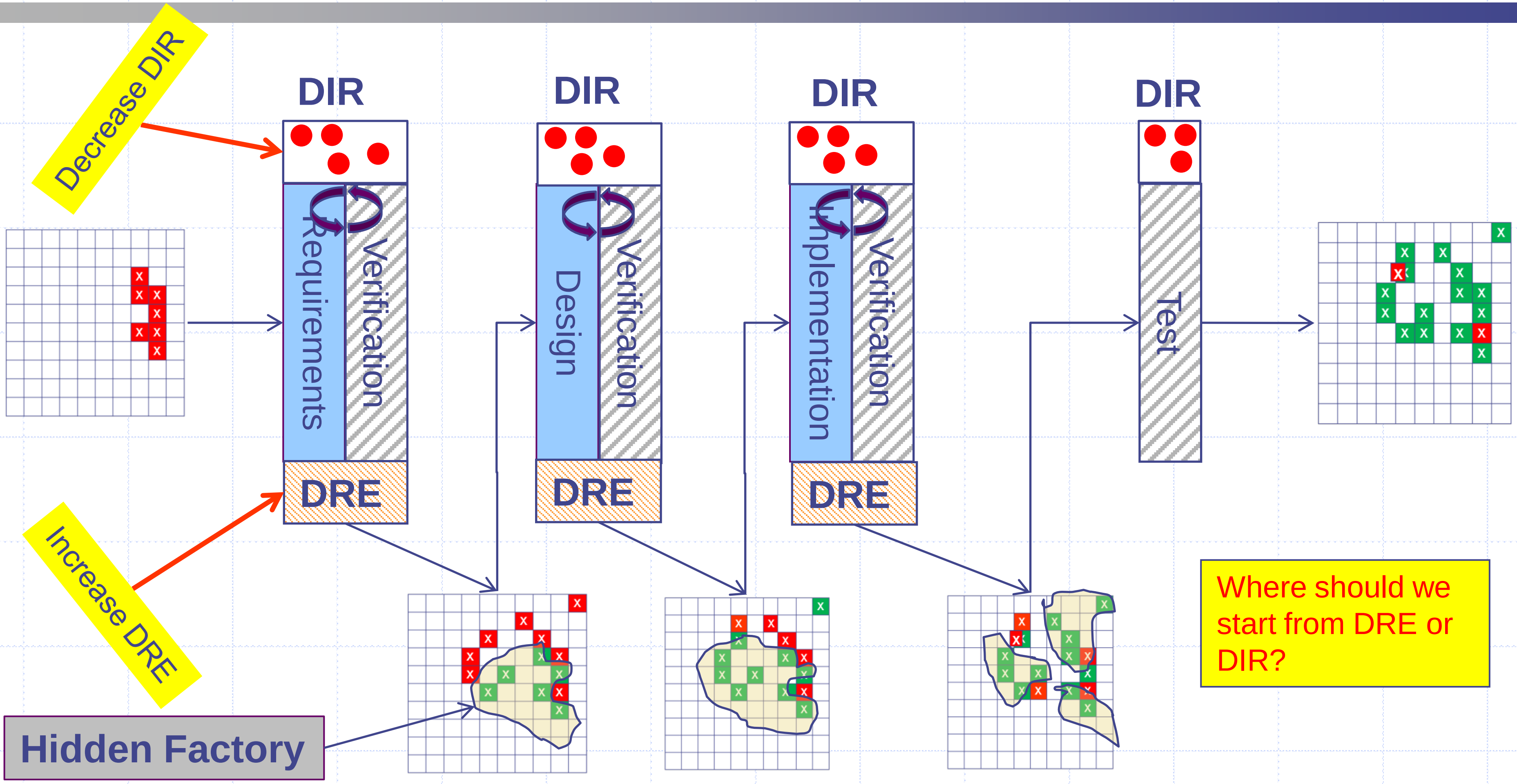
- The *Hidden Factory*
- Defects are not recorded prior to system test
- We are not recording the True Yield
- The Box called Software Development is a black box and hides other defects and reworks that could be avoided.

The hidden Factory in Software Process development



- **Activity$_i$ inherited a widget with 7 defects**
- **Activity$_i$ introduced 3 new defects, but also detected 2 existing defects to be fixed by Activity$_i$ or Activity$_{i-1}$, ...**
- **Verification$_i$ detected 4 defects, but introduced 3 new defects**
- **We have two types of Yields:**
  - Ability to produce non-defective units (equivalent to the classical Yield)
  - Ability to detect defects present at a given stage (mainly related to verification & validation activities)

The hidden Factory in Software Process development



Where should we start from DRE or DIR?

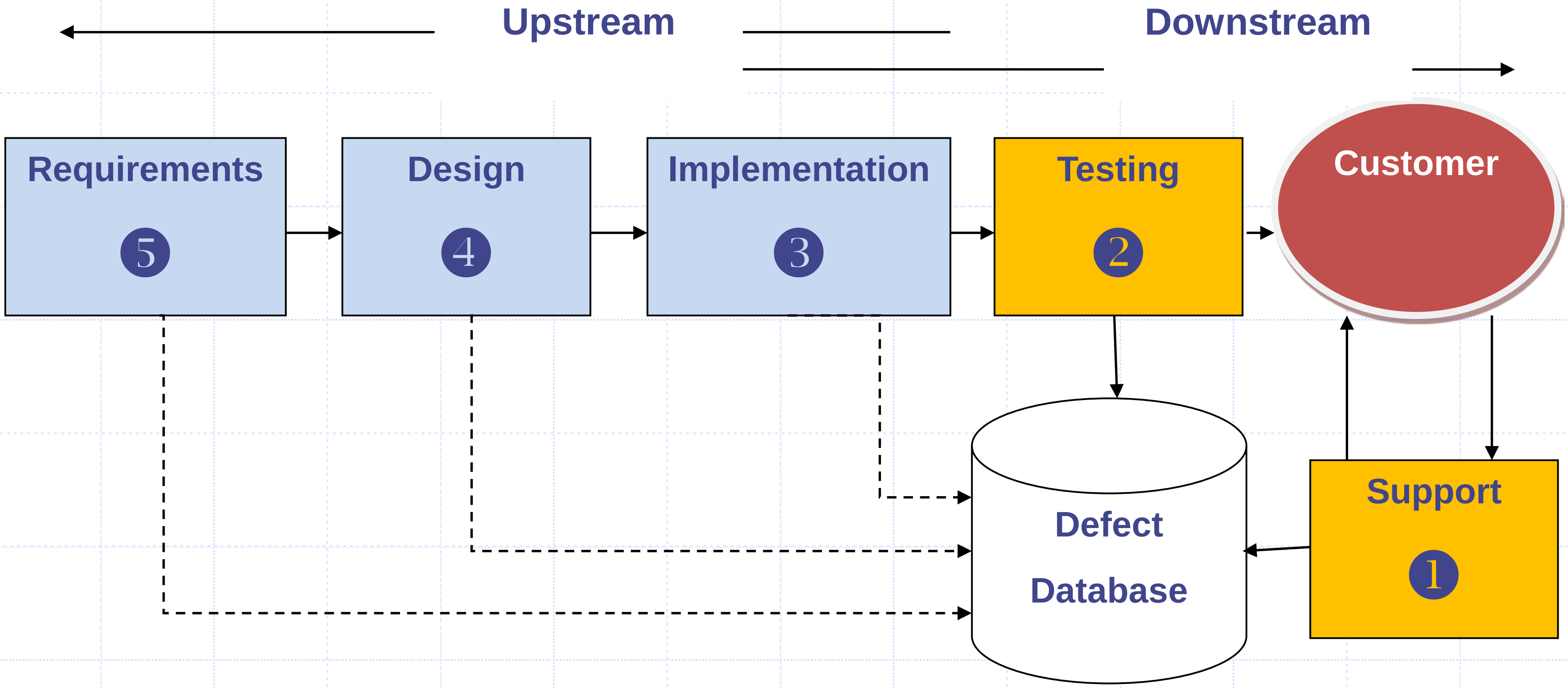
- **The activity yield and the total process yield depends on two parameters:**
  - The Defect Injection Rate (DIR)
  - The Defect Removal Effectiveness (DRE)

Reality of Six Sigma and Experimental approaches

Our observation

- **Understanding and characterising defects provides insight for process improvement**
  - Help prioritising effort
  - Provide a quantification of the “pain” (how big): Cost!
  - Type of techniques to prevent, contain or remove
- **The defects data (frequency and cost) is very much contextual**
- **Not many organisations are conscious of the cost of their defects**
- **The theory and even logic may dictate that we should start from Requirements and DIR**
- **In reality it is very difficult to define what is a defect for requirements and for Design... Because that means we know what is a “good” Requirement and a “good” design. Therefore we should:**
  - Start from defects that are close to the field
  - Characterise and profile defects to learn about the process
  - Start from problems related to effectiveness first and efficiency second

Stop the bleeding first...



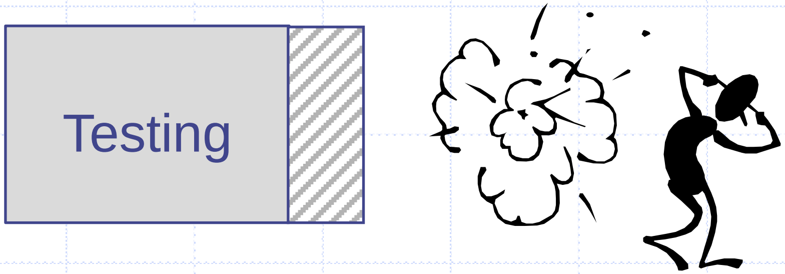
In theory, there is no difference between theory and practice.... but in practice there is.

attributed to Jan L. A. van de Snepscheut/Yogi Berra

Six Sigma
Case Study

Case Study

- **Objective of the case study is to show how:**
  - The Six Sigma DMAIC roadmap is a continuous learning process
  - The problem perception and formulation keep changing throughout the entire DMAIC
- **The project started as: "*We have a problem with the testing process*"**

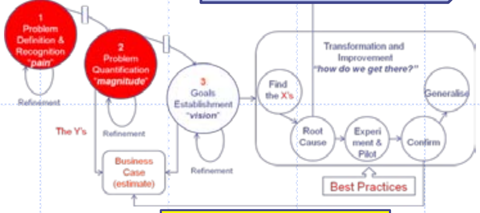


- **Questions to the audience:**
  - What are the typical problems for a testing process?
  - Think of a "pain" and why would that be a pain..
  - How would you quantify the problem ("pain")?

Problem Definition – “Pain”

DEFINE

- A typical dialogue…
  - -- *Our Testing process is not capable*
- What do you mean by “not capable”? Are there too many defects reaching the customer?
  - -- *YES*
  - -- *Actually NO… Customers are happy. However we spent too much time on the testing process.*
- Do you know how long your test process takes?
  - -- *NO… 35% to 45% of the development cycle*
- But that’s another problem it is not capability (effectiveness)… It is efficiency
  - -- *Yes, we want to reduce the testing cycle*
  - -- *May be we are testing too much*
  - -- *We were thinking about automating the test cases*
- Yes but these are solutions…



Exploratory

Is this normal? acceptable?

Questions to the audience:

- What would be your approach?

Problem Definition – “Pain”

DEFINE

Exploratory

- Maybe there are (too) many defects leaking from previous phases?
  - Yes... But we have already tried code reviews and design reviews...

    The developers said they do not work.

    They said that we find only trivial defects and code style errors. These can be found automatically by code analysers

    ...

Maybe your code review is inadequate?

- Anyway, you agree that one potential reason that your testing process is taking too long or is expensive is the fact that we have too many defects leaking from the previous phases

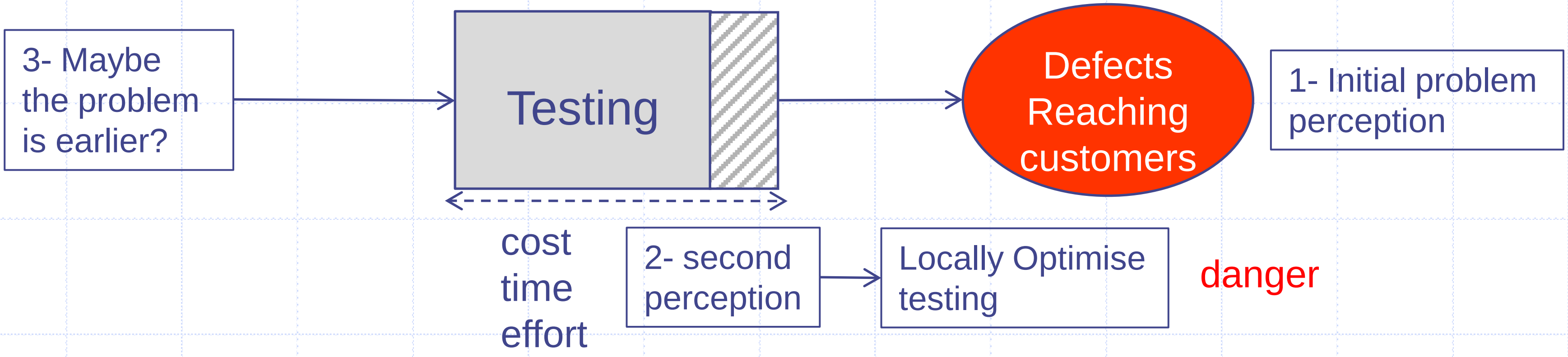
  The only way to know is learn more about the number of defects, the type of defects found during testing, how much they cost...

- Do we have this data?
  - Yes, not all of it

Problem Definition – “Pain”

DEFINE

Exploratory

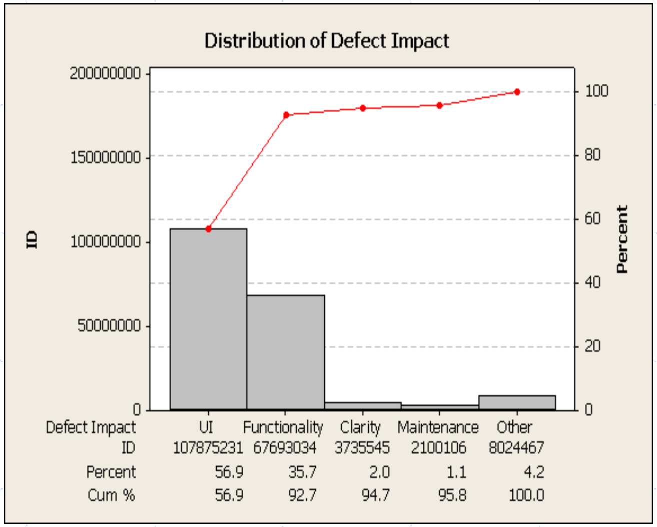

3- Maybe the problem is earlier?
Testing
Defects Reaching customers
1- Initial problem perception
cost
time
effort
2- second perception
Locally Optimise testing
danger

- **Initial Problem formulation:**

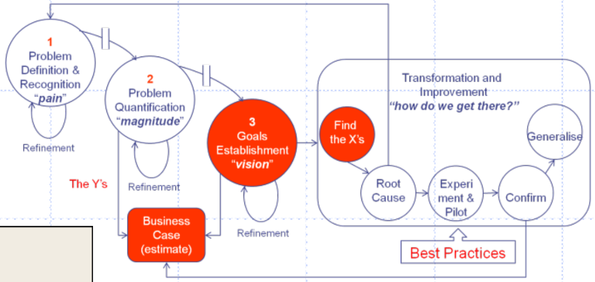
  Low process yield before QC (testing). This results in high number of defects that are discovered by the QC team relatively to the total number of defects found in process. The majority of the projects (76%) find between 70% to 100% of defects during the QC phase.

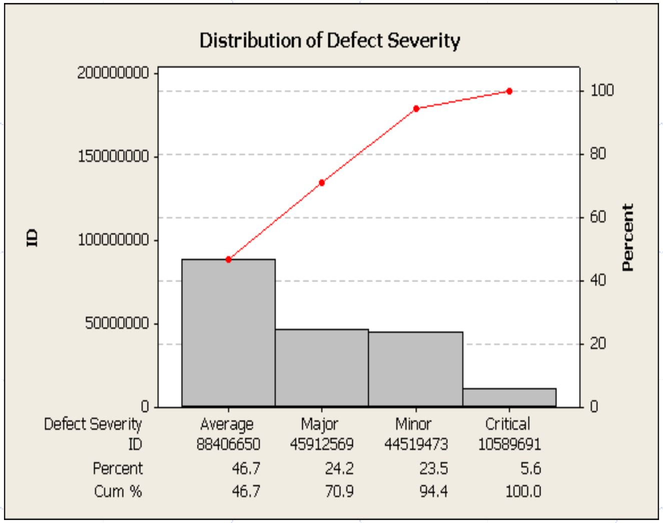
Defects Characterisation

MEASURE

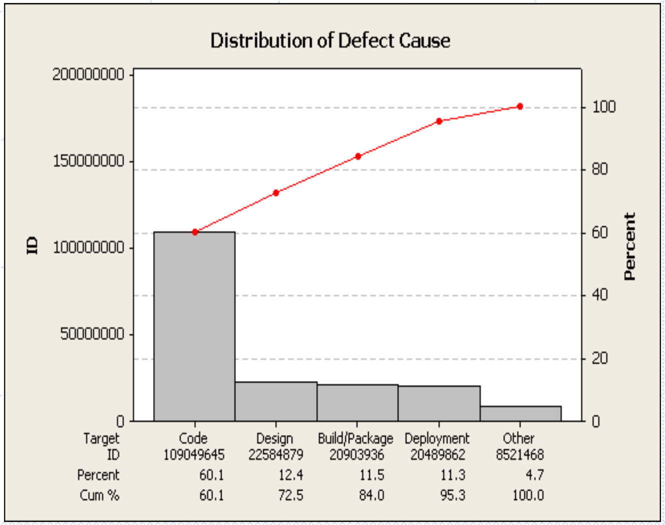


Defect Impact	UI	Functionality	Clarity	Maintenance	Other
ID	107875231	67693034	3735545	2100106	8024467
Percent	56.9	35.7	2.0	1.1	4.2
Cum %	56.9	92.7	94.7	95.8	100.0





Defect Severity	Average	Major	Minor	Critical
ID	88406650	45912569	44519473	10589691
Percent	46.7	24.2	23.5	5.6
Cum %	46.7	70.9	94.4	100.0



Target	Code	Design	Build/Package	Deployment	Other
ID	109049645	22584879	20903936	20489862	8521468
Percent	60.1	12.4	11.5	11.3	4.7
Cum %	60.1	72.5	84.0	95.3	100.0

92% of defects found during testing are either UI or Functional

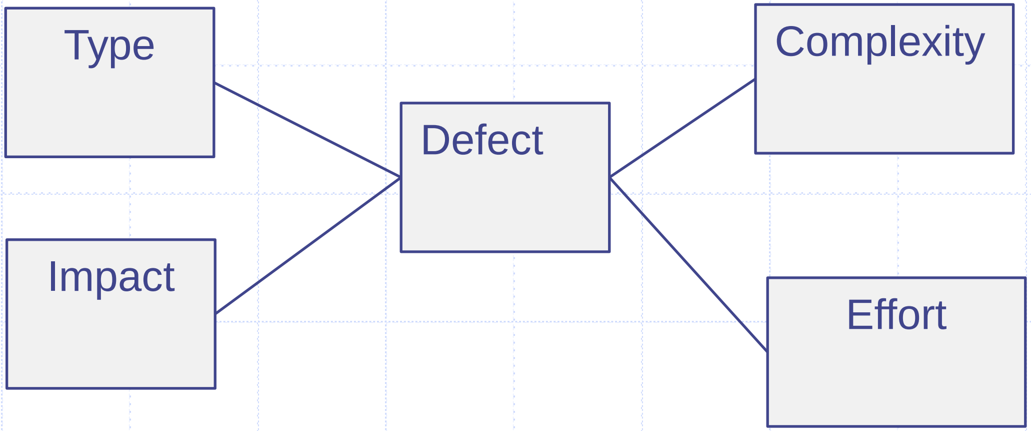
They both affects the User but may be originated from different sources

We can also question the quality of the defects categorisation

Elements of cost related to Defects

MEASURE

- Assess the defects on multiple dimensions



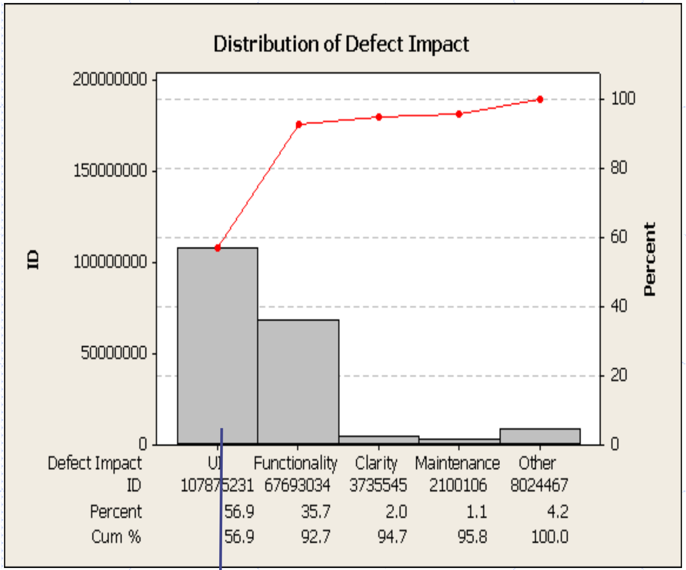
-Low, M, H
-Pairwise comparison

-Testing (common)
- Discovery (test)
- Reporting
-Fixing
- re-test

- Rough Estimate for the business case
  - Effort use Median with 5 estimation (97% probability that the median is within the min and max of the 5 values)
- Detailed estimate
  - Calibration phase
  - Estimation phase
  - Experimentation for assessing the capability of the defect categorisation (measurement)
    - Attribute Agreement Analysis

Root causes

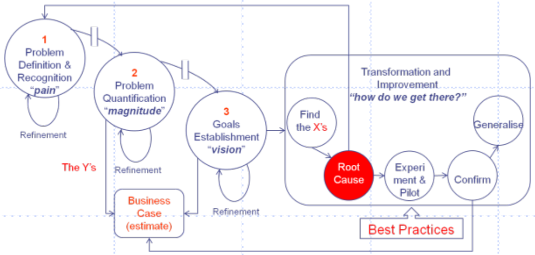
ANALYSE



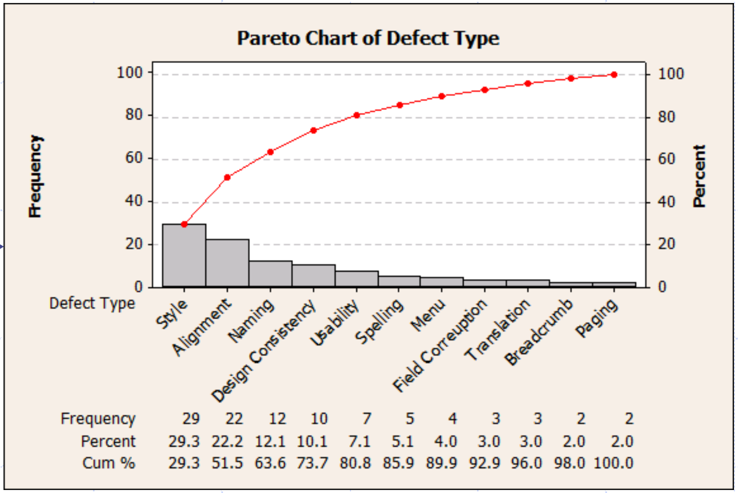
UI and Functional defects have probably different origins

You don’t address and you don’t verify UI defects and Functional defects the same way.

Two different streams

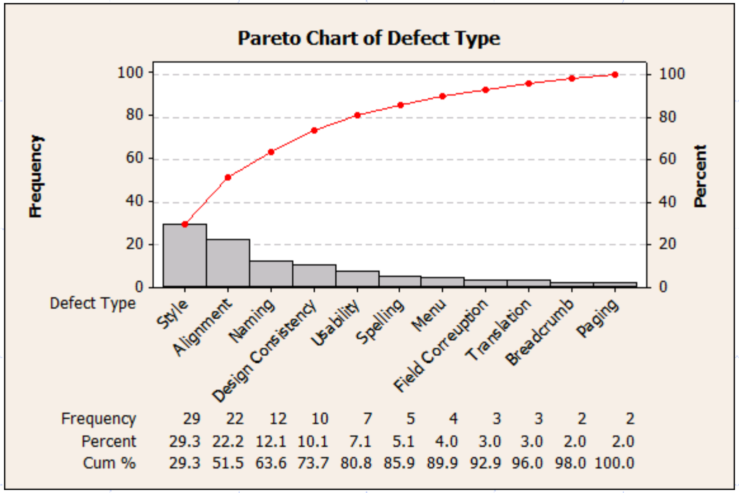


1	Alignment	Code
2	Design consistency	D
3	Field Corruption	F
4	Functionality	Ignored in UI
5	Naming	N
6	Menu	M
7	Paging	P
8	Translation	T
9	Spelling	S
10	Style	Y
11	Usability	U
12	breadcrumb	B



Root causes – problem focus

ANALYSE

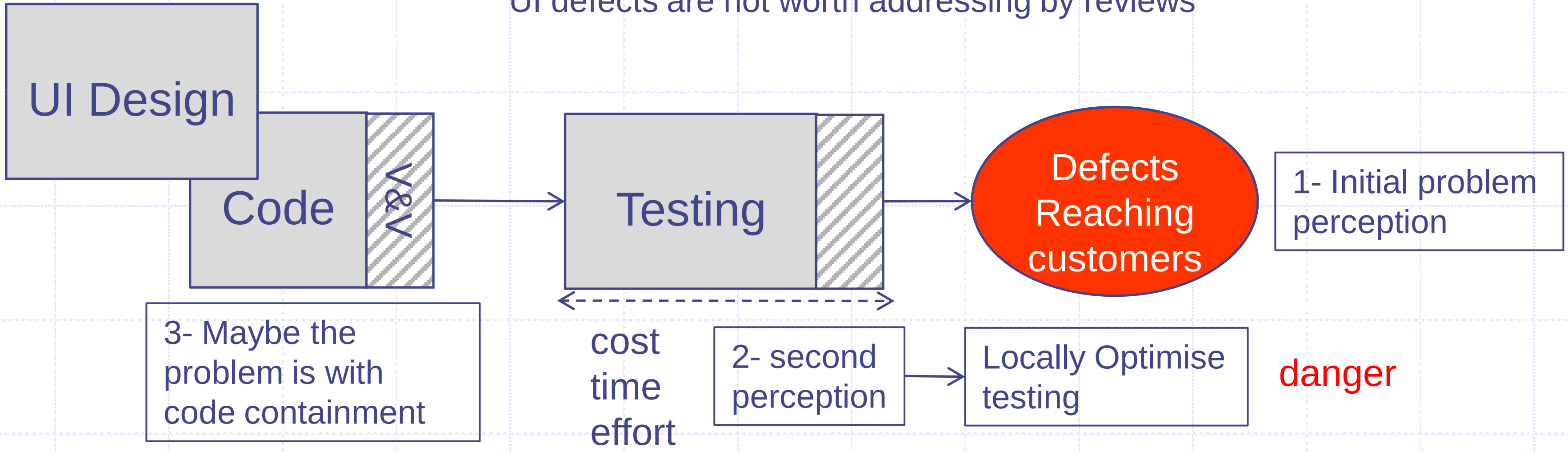


Defect Type	Style	Alignment	Naming	Design Consistency	Usability	Spelling	Menu	Field Correuption	Translation	Breadcrumb	Paging
Frequency	29	22	12	10	7	5	4	3	3	2	2
Percent	29.3	22.2	12.1	10.1	7.1	5.1	4.0	3.0	3.0	2.0	2.0
Cum %	29.3	51.5	63.6	73.7	80.8	85.9	89.9	92.9	96.0	98.0	100.0

4- the problem is likely in the UI design process

?

These types of defects will not be caught by review (maybe visual inspection)
UI defects are not worth addressing by reviews



1- Initial problem perception

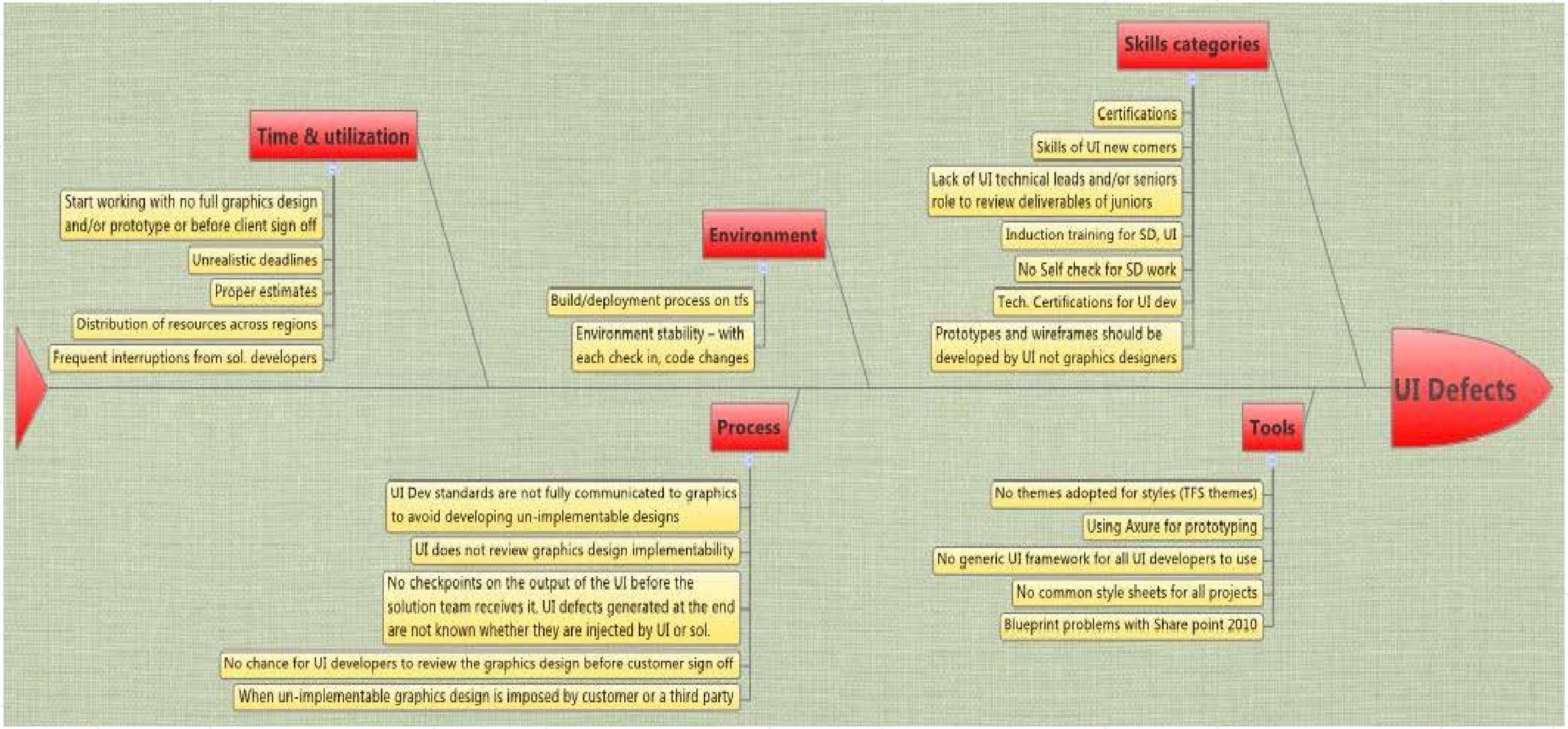
2- second perception

Locally Optimise testing

danger

3- Maybe the problem is with code containment

Root causes – problem focus



UI Defects

Time & utilization

- Start working with no full graphics design and/or prototype or before client sign off
- Unrealistic deadlines
- Proper estimates
- Distribution of resources across regions
- Frequent interruptions from sol. developers

Environment

- Build/deployment process on tfs
- Environment stability – with each check in, code changes

Skills categories

- Certifications
- Skills of UI new comers
- Lack of UI technical leads and/or seniors role to review deliverables of juniors
- Induction training for SD, UI
- No Self check for SD work
- Tech. Certifications for UI dev
- Prototypes and wireframes should be developed by UI not graphics designers

Process

- UI Dev standards are not fully communicated to graphics to avoid developing un-implementable designs
- UI does not review graphics design implementability
- No checkpoints on the output of the UI before the solution team receives it. UI defects generated at the end are not known whether they are injected by UI or sol.
- No chance for UI developers to review the graphics design before customer sign off
- When un-implementable graphics design is imposed by customer or a third party

Tools

- No themes adopted for styles (TFS themes)
- Using Axure for prototyping
- No generic UI framework for all UI developers to use
- No common style sheets for all projects
- Blueprint problems with Share point 2010

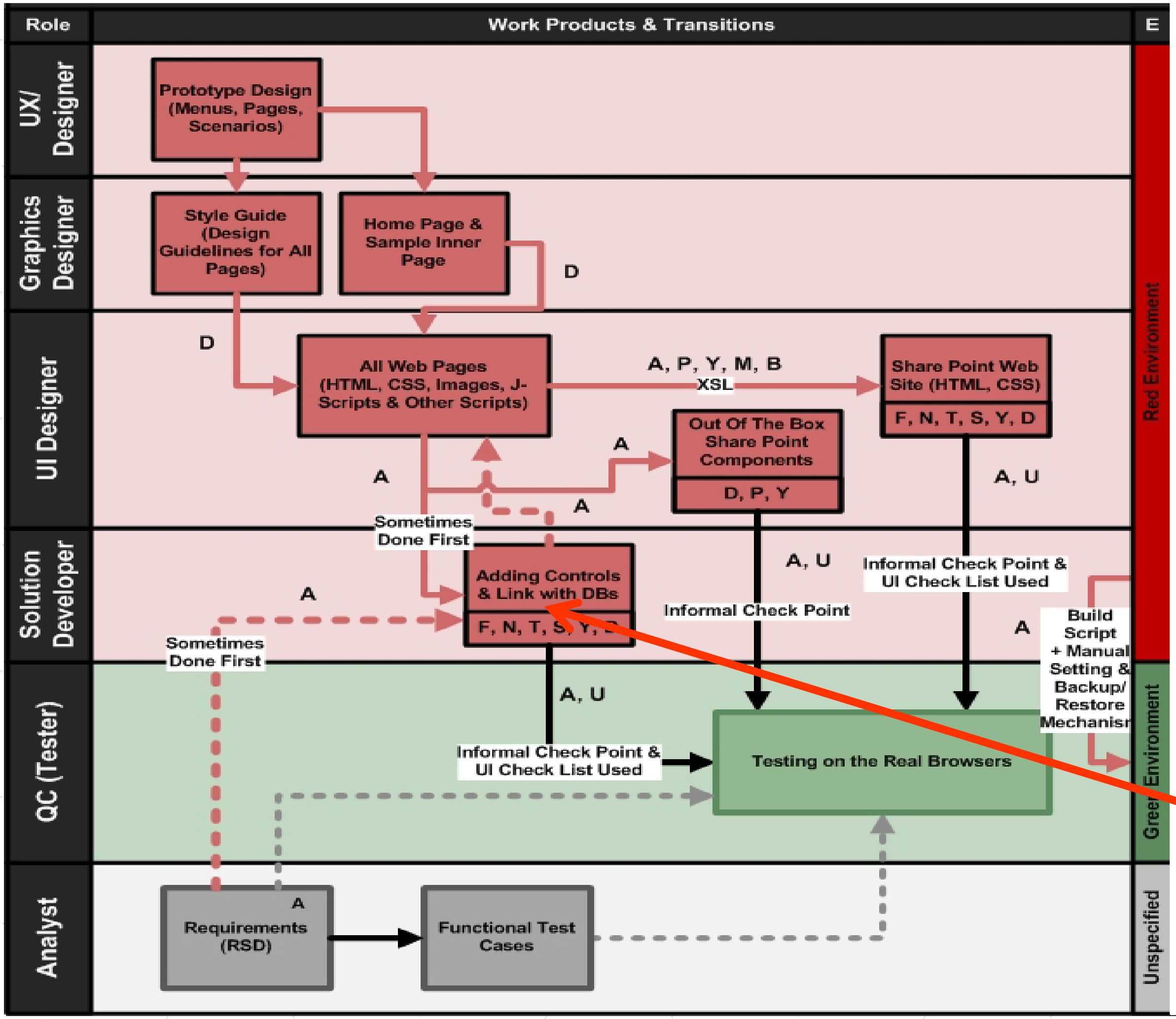
Detailed Process Map x Defects

Gemba

現場

ANALYSE

Observation



Role
Work Products & Transitions
E
UX/ Designer
Graphics Designer
UI Designer
Solution Developer
QC (Tester)
Analyst
Prototype Design (Menus, Pages, Scenarios)
Style Guide (Design Guidelines for All Pages)
Home Page & Sample Inner Page
D
D
All Web Pages (HTML, CSS, Images, J-Scripts & Other Scripts)
A, P, Y, M, B
XSL
Share Point Web Site (HTML, CSS)
F, N, T, S, Y, D
Out Of The Box Share Point Components
D, P, Y
A
A
A
A, U
Sometimes Done First
A
Adding Controls & Link with DBs
F, N, T, S, Y, D
A, U
Informal Check Point & UI Check List Used
Informal Check Point
A
Build Script + Manual Setting & Backup/ Restore Mechanism
Sometimes Done First
A, U
Informal Check Point & UI Check List Used
Testing on the Real Browsers
A
Requirements (RSD)
Functional Test Cases
Red Environment
Green Environment
Unspecified

- **Defects occur either at:**
  - Handover from one level to another (V)
  - the transition from one tool/environment to another
  - Within the same activity
- **Which type of defects occur and at what level of interaction?**

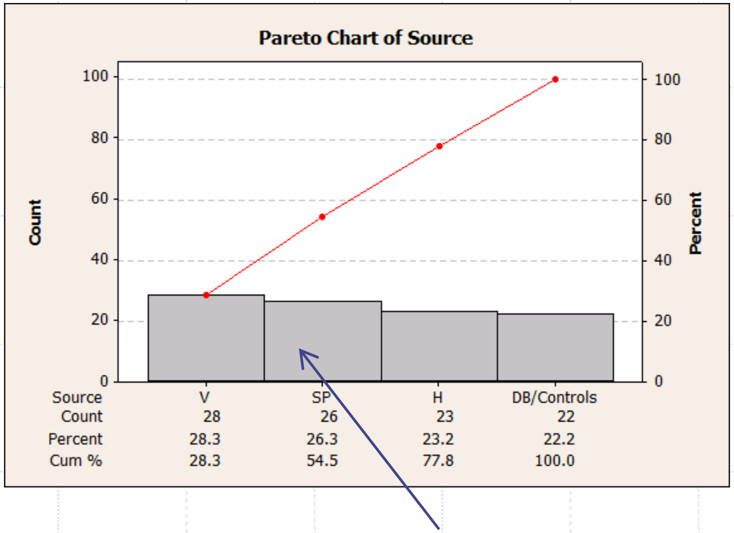
Other factors influencing this step:
-- Complexity
-- Size
-- Number of components/widgets
-- Skills
-- Tools

systonomy

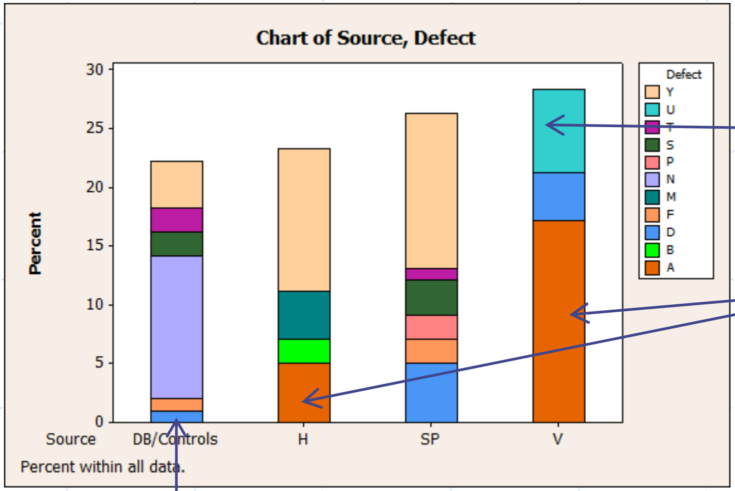
ANALYSE

Data Analysis

- **Defects occur either at:**
  - Handover from one level to another (V)
  - the transition from one tool/environment to another
  - Within the same activity
- **Which type of defects occur at what level of interaction?**



Large number of defects due to the technology/environment



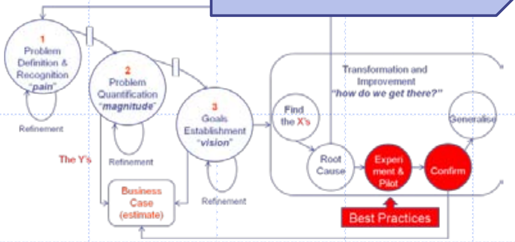
Usability problems at handover

Large number of alignment defects (trivial defects) due to handover

Human activity, consists in adding controls, links to DB, etc.

Solutions Identification and Evaluation

IMPROVE



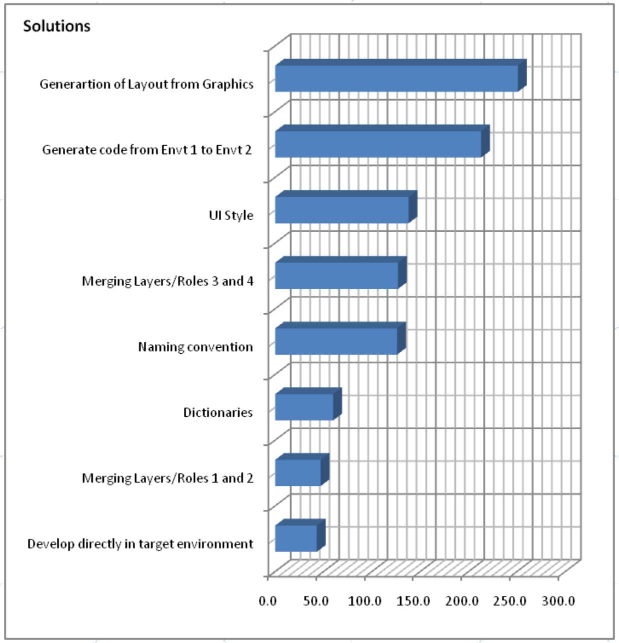
1 Problem Definition & Recognition "pain"
Refinement
2 Problem Quantification "magnitude"
The Y's
Refinement
Business Case (estimate)
3 Goals Establishment "vision"
Refinement
Transformation and Improvement "how do we get there?"
Find the X's
Root Cause
Experi ment & Pilot
Confirm
Generalise
Best Practices

- **Sources:**
  - Handover
  - Transition
  - SP Technology
  - DB/Controls

Hypothesis

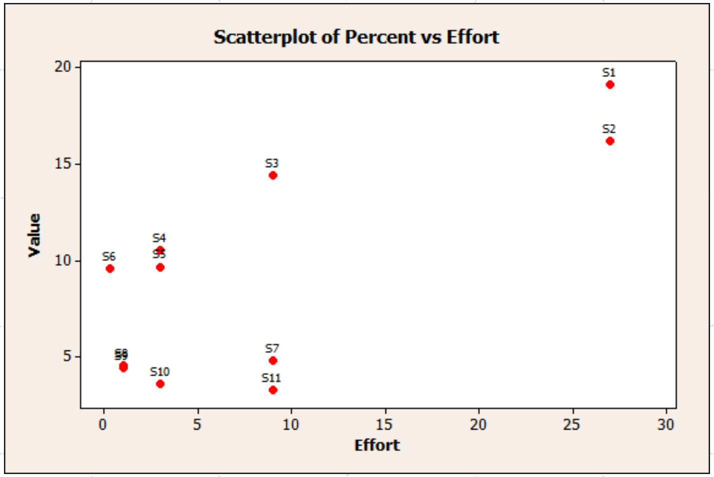
Solutions

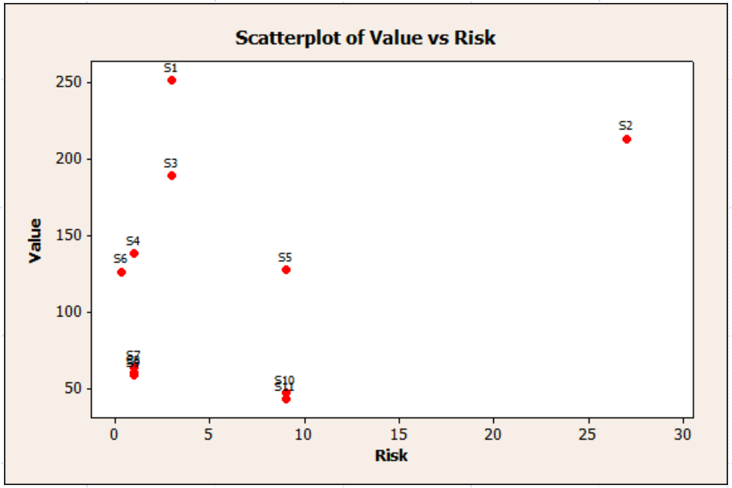
UI Defects	Importance	Generartion of Layout from Graphics	Generate code from Envt 1 to Envt 2	UI Reviews	Define UI Style	Merging Layers/Roles 3 and 4	Naming convention	Training/Education	Dictionaries	Establish Checklists	Merging Layers/Roles 1 and 2	Develop directly in target environment	Total	%
Field Corruption	3.0	L	M									M	**21.0**	**02**
Breadcrumb	2.0	L	M					L				M	**16.0**	**01**
Paging	2.0	H			L	M		M			M	H	**56.0**	**04**
Style	29.0			M	M	M				L			**290.0**	**22**
Alignment	22.0	H	H										**396.0**	**30**
Naming	12.0			M	L		H		M	L			**204.0**	**16**
Design Consistency	10.0	M			M	M	L	M		L	M	L	**180.0**	**14**
Usability	7.0			H	L			M		L	L		**105.0**	**08**
Spelling	5.0						L		M				**20.0**	**02**
Menu	4.0					L		L			L		**12.0**	**01**
Translation	3.0			L			L		M				**15.0**	**01**
Total		**251.0**	**213.0**	**189.0**	**138.0**	**127.0**	**126.0**	**63.0**	**60.0**	**58.0**	**47.0**	**43.0**		
%		**19**	**16**	**14**	**10**	**10**	**10**	**05**	**05**	**04**	**04**	**03**		
Cost/Effort		VH	VH	H	M	M	VL	H	L	L	M	H		
Risk		M	VH	M	L	H	VL	L	L	L	H	H		

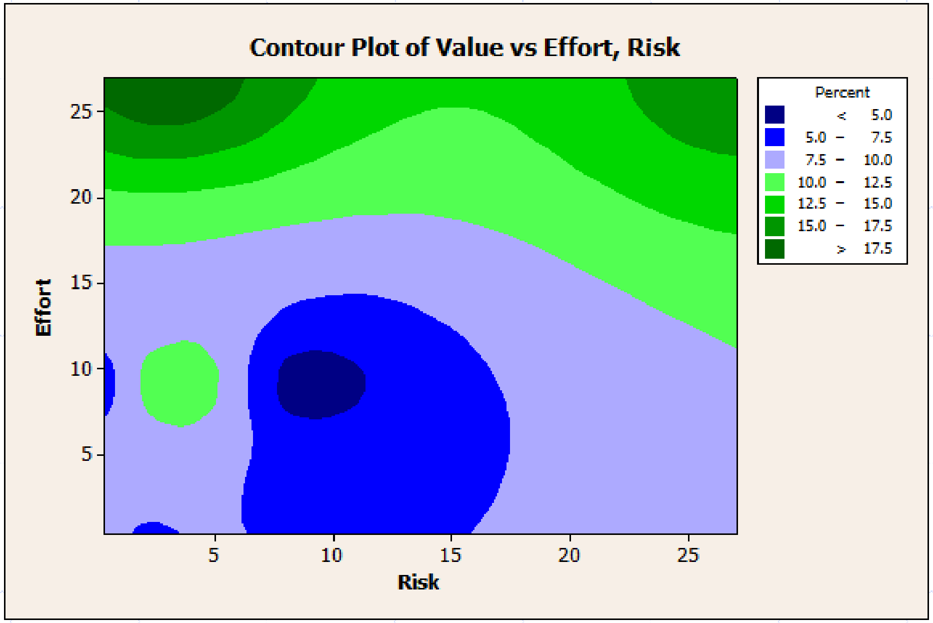


Solutions Identification and Evaluation

IMPROVE

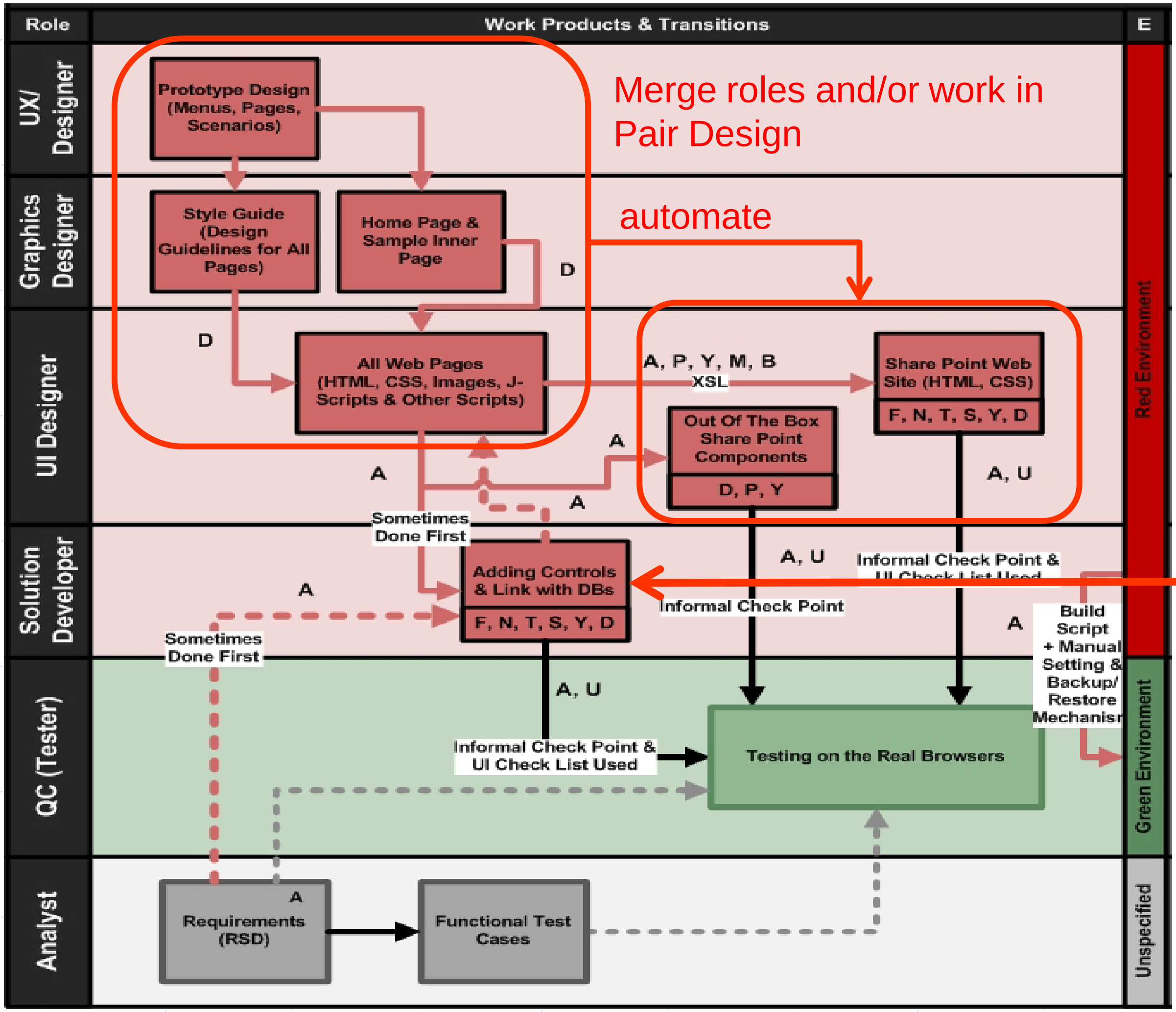






Process alterations and experiments

IMPROVE



Role | Work Products & Transitions | E

UX/ Designer

Graphics Designer

UI Designer

Solution Developer

QC (Tester)

Analyst

Prototype Design (Menus, Pages, Scenarios)

Style Guide (Design Guidelines for All Pages)

Home Page & Sample Inner Page

D

D

All Web Pages (HTML, CSS, Images, J-Scripts & Other Scripts)

A, P, Y, M, B XSL

Share Point Web Site (HTML, CSS)

F, N, T, S, Y, D

Out Of The Box Share Point Components

D, P, Y

A

A

A

A, U

Sometimes Done First

Adding Controls & Link with DBs

F, N, T, S, Y, D

A, U

Informal Check Point & UI Check List Used

Informal Check Point

A

Sometimes Done First

A

Build Script + Manual Setting & Backup/ Restore Mechanism

A, U

Informal Check Point & UI Check List Used

Testing on the Real Browsers

A

Requirements (RSD)

Functional Test Cases

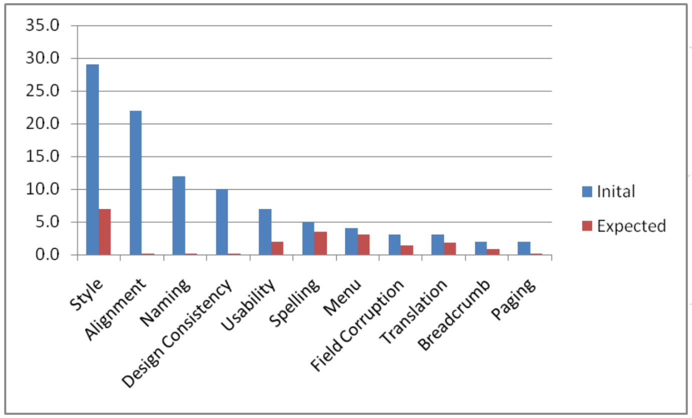
Red Environment

Green Environment

Unspecified

Merge roles and/or work in Pair Design

automate

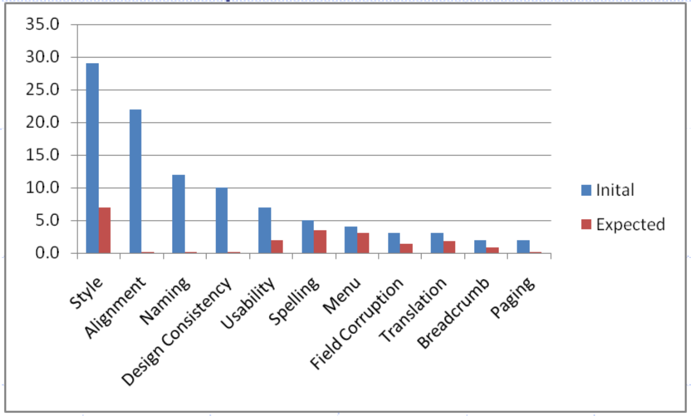


Checklists
Reviews/Usability Reviews
Education
Role merge is difficult
Due to cultural barriers

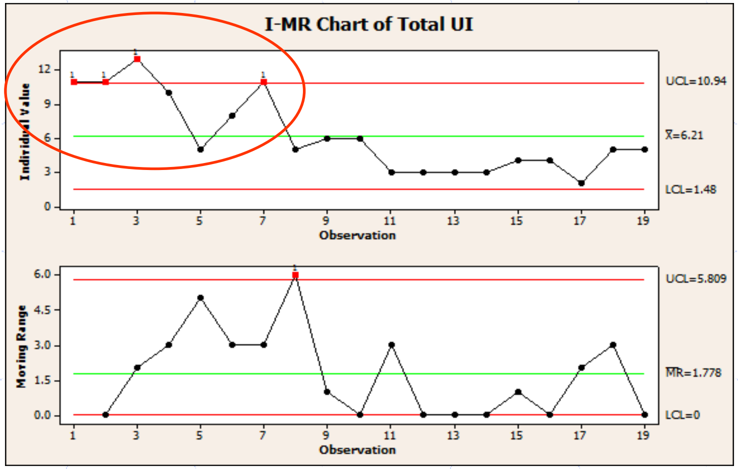
Improvement associated to all actions and not to single actions

CONTROL: show and validate improvement

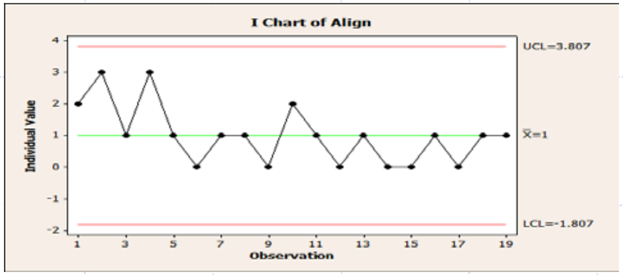
- A quick visual summary can show the actual improvement in effectiveness

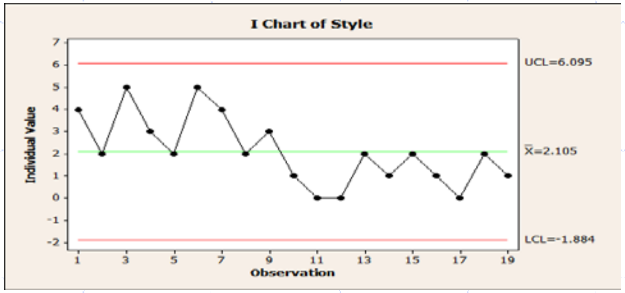


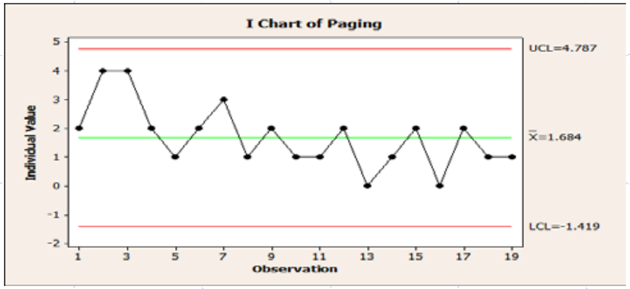
Learning and adaptation phase



Total UI defects per test cycle







UI defects per type per test cycle

Summary

- **The mother of all Six Sigma tools is *naive* questioning**
- **We don't take time to observe our processes**
- **Studying defects provide lot of insights on process improvements and applicable techniques**
- **Data is A-Political**
  - Test two alternatives and see what the data will say...
  - Drive things by consensus (even if the solution seems obvious)
- **Empirical software engineering is practical with lot of pragmatism**
- **Economics of software engineering is a key ingredient to Empirical Software Engineering**

Other key takeaways...

- Do not focus on the tools, but rather on the principles
- Do not apply the method by the books
- Do not just Copy and Paste a practice or a technique
- Do not just dismiss a practice because someone has, understand why?
- Adopt the method to your business, context and environment
- Do not follow the process for the sake of the process
- Recognise that there is no perfect project
- Recognise that human are not perfect... and engineers and customers are humans
- Recognise the importance of human factors... and culture

Never forget the Business

References

- [End 03] A. Enders, D. Rombach, A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories, Pearson Addison-Wesley, 2003.
- [Harry 00] M Harry, and R Schroeder “Six Sigma: The Breakthrough Management Strategy Revolutionizing the World’s Top Corporations”, 2000
- [RAI 07] A. Rainer, “The Value of Empirical Evidence for Practitioners and Researchers”, in Empirical Software Engineering Issues, Critical Assessment and Future Directions, Internal Workshop, Dagstuhl Castle, Germany June 26-30, 2006 Revisited Papers, Springer-Verlag, pp. 24,2007.
- [OUD 05] Oudrhiri: Six Sigma and DFSS for IT and Software Engineering, TikciT Magazine, Q2’05 issue, 2005
- [OUD 06] Oudrhiri R, Pellizzetti F, Using maturity models for Six Sigma and strategy execution, IQPC 7 Annual conference, Amsterdam, 2006
- [SEA 07] C. B. Seaman, “Empirical Paradigm: Position Paper”, in Empirical Software Engineering Issues, Critical Assessment and Future Directions, Internal Workshop, Dagstuhl Castle, Germany June 26-30, 2006 Revisited Papers, Springer-Verlag, pp. 23,2007.
- [Weinberg 92-97] Weinberg G., Quality Software Management (all- volumes 1-4), Dorset Publishing, 1992-1997