Systems Engineering, Quality Control, and Quality Assurance

Advanced SE QC and QA

FOR SWISSED 2022, INVITED KEYNOTE
Monday 12th SEPTEMBER 2022, Zurich

By Tom @ Gilb . Com, www.Gilb.com, www.Twitter.com/imtomgilb
Near Oslo, Norway

https://www.dropbox.com/sh/enpdevyu2hnbur1/AAAp16d143VkDVWTWnEKng9Ma?dl=0
Swiss SE Slides Folder

Swissed Slides folder

My view of Oslofjord from our summer cabin 7th Sept 2022
Park your yacht, or seaplane and have a beer.
PS Norway has mountains too!

Advanced SE QC and QA

Advanced SE QC and QA

- a 2022 Update on V-Model Thinking:
- Specific Practical Methods for shifting the QA burden to the ‘left'
  - (better requirements and design, defect prevention processes),
- and for focussing testing as an extreme agile feedback cycle,
- using ‘Competitive Engineering/Planguage”
- and methods of Musk and Bezos.
- An overview of
  - basic ideas,
  - practices,
  - case study experiences and
  - measures of effectiveness.

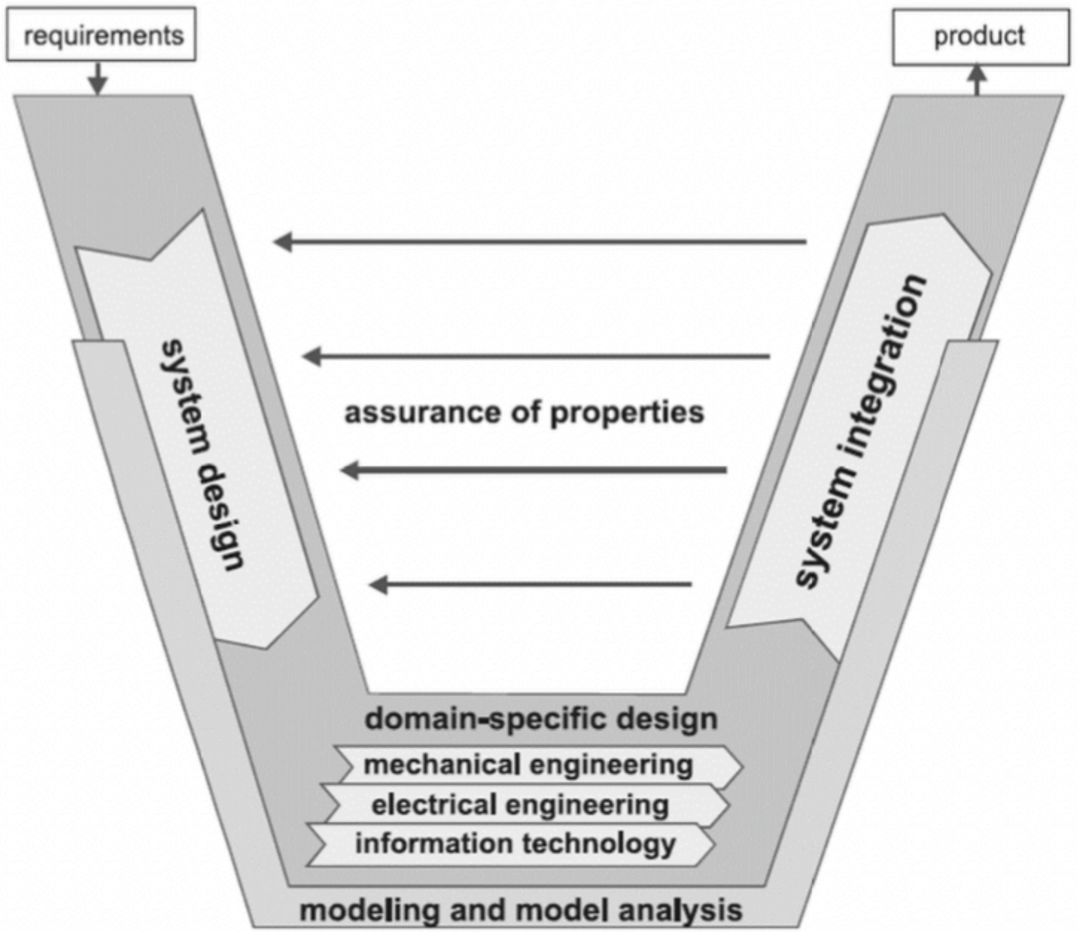
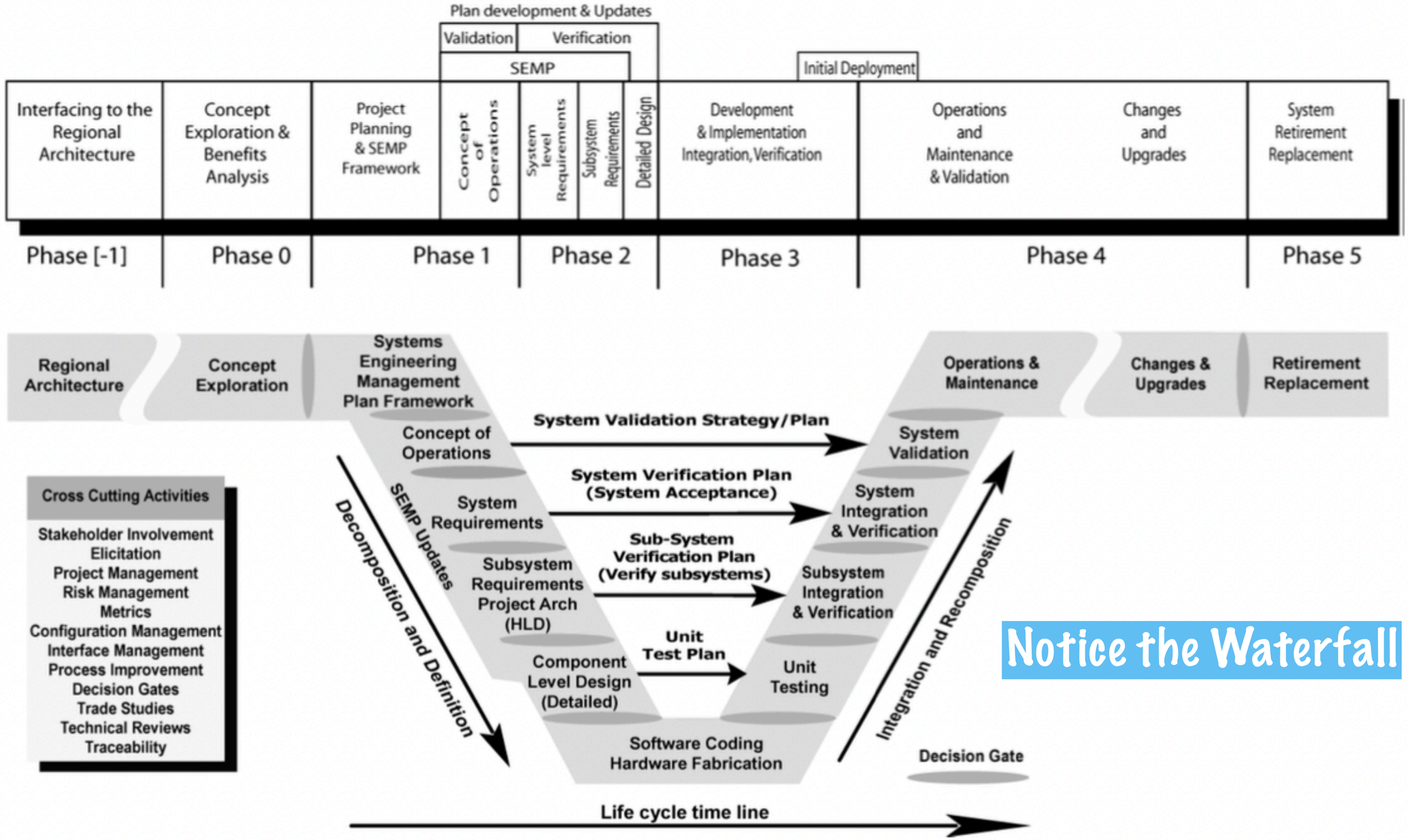


Figure 1. V-model of the VDI 2206:2004 guideline "Design methodology for mechatronic systems" (VDI, 2004)

www.researchgate.net/publication/325137200_V-MODELS_FOR_INTERDISCIPLINARY_SYSTEMS_ENGINEERING



Notice the Waterfall

Figure 2. V-model of the US Department of Transportation (US-DoT) (U.S. Department of Transportation, 2009)

Shift Left means

Reduce Rework
50% -< 3%

- Agile is an option
- QC at specification level, discover defects early
- QA to PREVENT defects, upstream (left-V)
- Advanced specification languages: intelligible, digital
- Not 'elimination of testing'
  - But big reduction in defects found in test (10x, 100x)
- = faster delivery, fewer defects, early value delivery

QA, Spec QC, Design

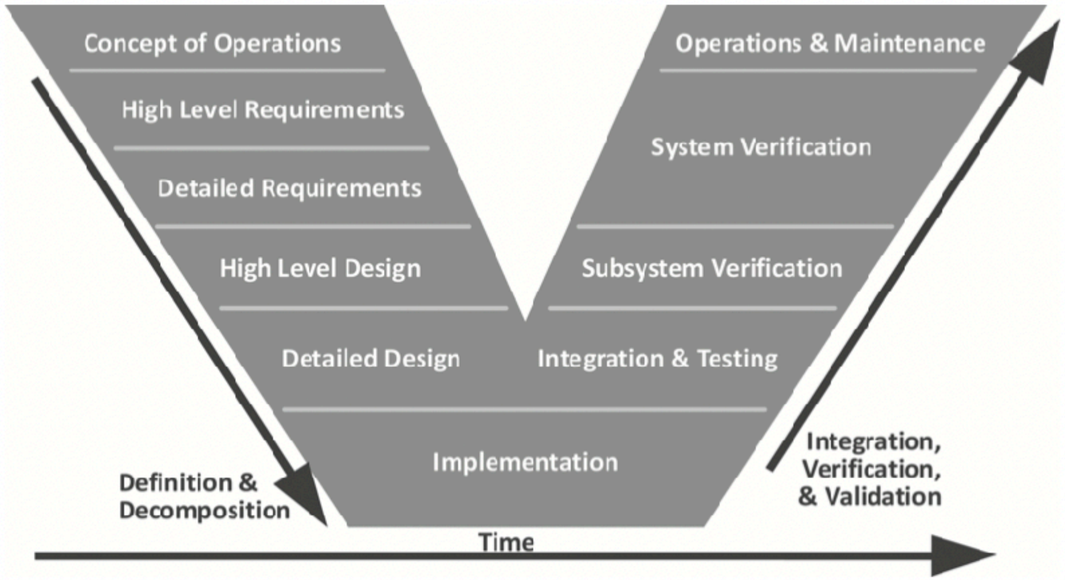


Figure 2. Overview of V-model of systems engineering.

DOI: 10.1518/155534308X377108

The cost of rework
After Test
Is 9.3X more per defect
Than on the left side of the V

Shift Left

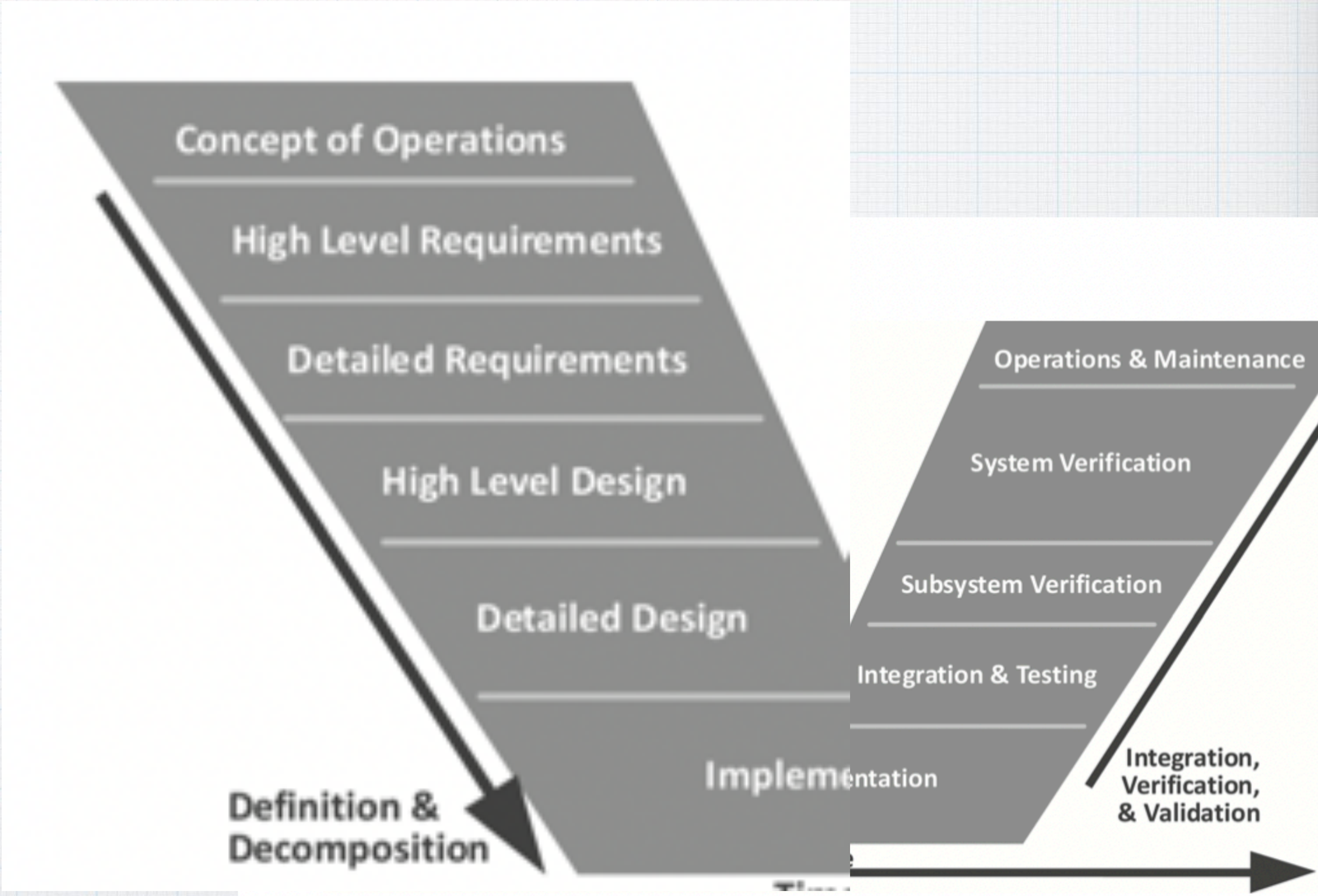


Figure 2. Overview of V-model of systems engineering.

DOI: 10.1518/155534308X377108

Reduce total effort and time by shifting left

* Invest more in QA initially
* + Reduce Testing, and Corrections
* = Reduce Total Costs

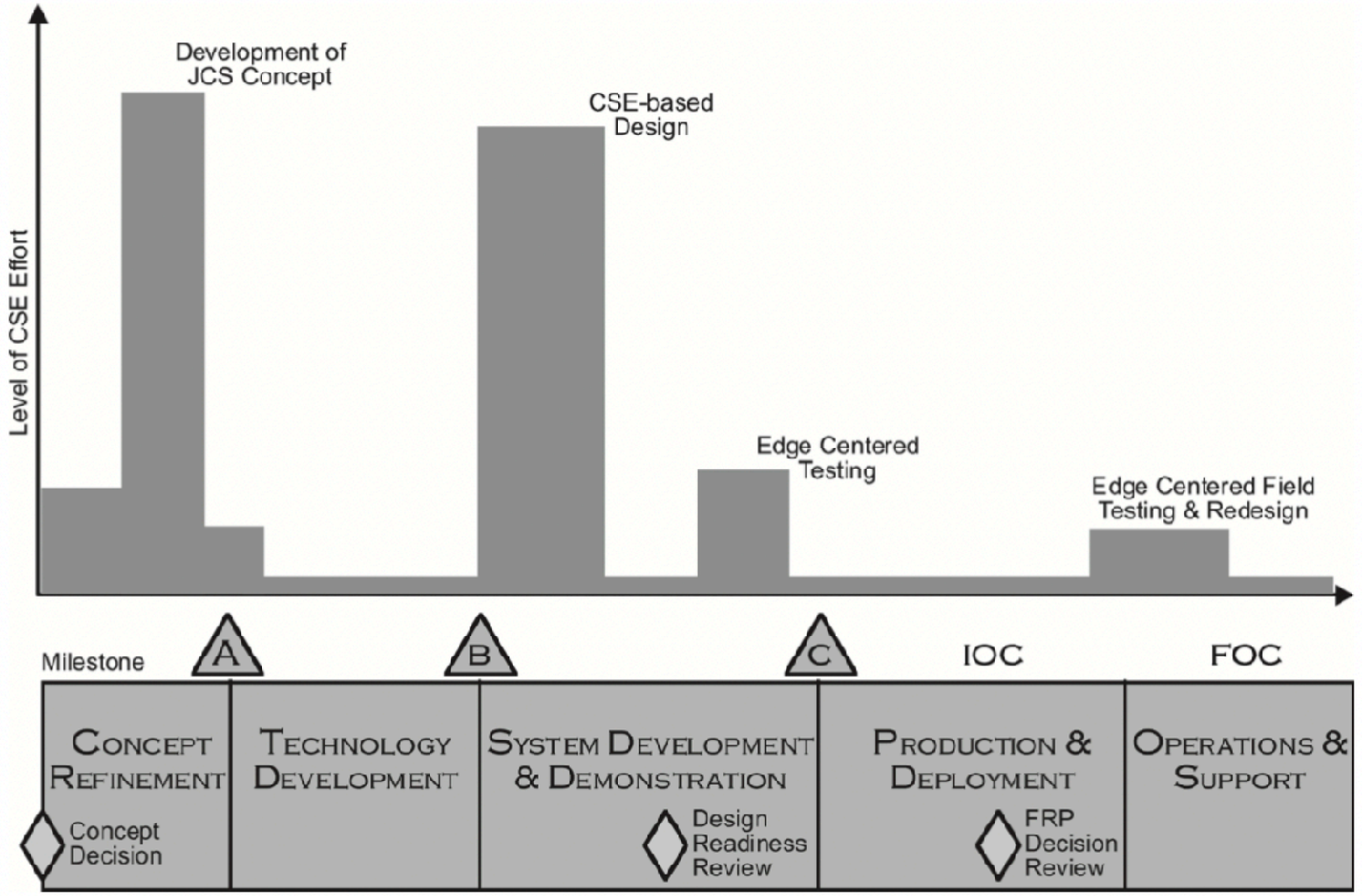


Figure 7. Relative level of cognitive systems engineering (CSE) effort in the Department of Defense system acquisition process required for effective joint cognitive system (JCS) development. IOC = initial operational capability, FOC = final operating capability, FRP = full rate production.

DOI: 10.1518/155534308X377108

Result – Obsolescence of Existing Weapons and Doctrine

Today's Legacy Systems	Where We Need To Be
Survivable assets/Stealth	** Attritable swarm of assets
Large platforms	Distributed consumable assets
Billion dollar platforms	Very low cost assets
Best in-class features	Aggregation of mass good-enough features
Fixed features	Easily Reconfigurable features
Updates to systems in months/years	Updates to systems in minutes/hours/days
Trusted platforms	No trust in anything
Human-to-human connection	Machine-to-machine
Human in the loop (human speed)	Machine-to-machine
Hub and Spoke Network	Mesh Networks
Old Primes	New primes
Buy and then operate our own assets	Buy services and outcomes
Contractors bend sheet metal and software is an add-on	Software is the platform and hardware is the add-on
DOD Labs and Primes own all the advanced tech	Advanced tech also driven by commercial companies
Centralized Manufacturing	Manufacturing close to the FOB
Waterfall Engineering	Agile – Iterative & incremental
Fixed Requirements	Pivot - as you learn
Fixed Contracts	Agile contracting - as you learn
Innovation disconnected or lower echelon activity	Continuous Innovation integral to the mission at HQ level

https://www.slideshare.net/sblank/technology-innovation-and-modern-war-lecture-2-238614359?next_slideshow=1

** ATTRITABLE: wear down (an opponent or enemy) by sustained action: *his defense was designed to attrit us*.

Governeering: Government Systems Engineering Planning.
https://tinyurl.com/Governeering.

Steve Blank

CMMI V2.0 Requirements Development and Management:
Your Toolkit for Engineering Agile Systems

RDM 1.1	Record requirements.
RDM 2.1	Elicit stakeholder needs, expectations, constraints, and interfaces or connections.
RDM 2.2	Transform stakeholder needs, expectations, constraints, and interfaces or connections into prioritized customer requirements.
RDM 2.3	Develop an understanding with the requirements providers on the meaning of the requirements.
RDM 2.4	Obtain commitment from project participants that they can implement the requirements.
RDM 2.5	Develop, record, and maintain bidirectional traceability among requirements and activities or work products.
RDM 2.6	Ensure that plans and activities or work products remain consistent with requirements.
RDM 3.1	Develop and keep requirements updated for the solution and its components.
RDM 3.2	Develop operational concepts and scenarios.
RDM 3.3	Allocate the requirements to be implemented.
RDM 3.4	Identify, develop, and keep updated interface or connection requirements.
RDM 3.5	Ensure that requirements are necessary and sufficient.
RDM 3.6	Balance stakeholder needs and constraints.
RDM 3.7	Validate requirements to ensure the resulting solution will perform as intended in the target environment.

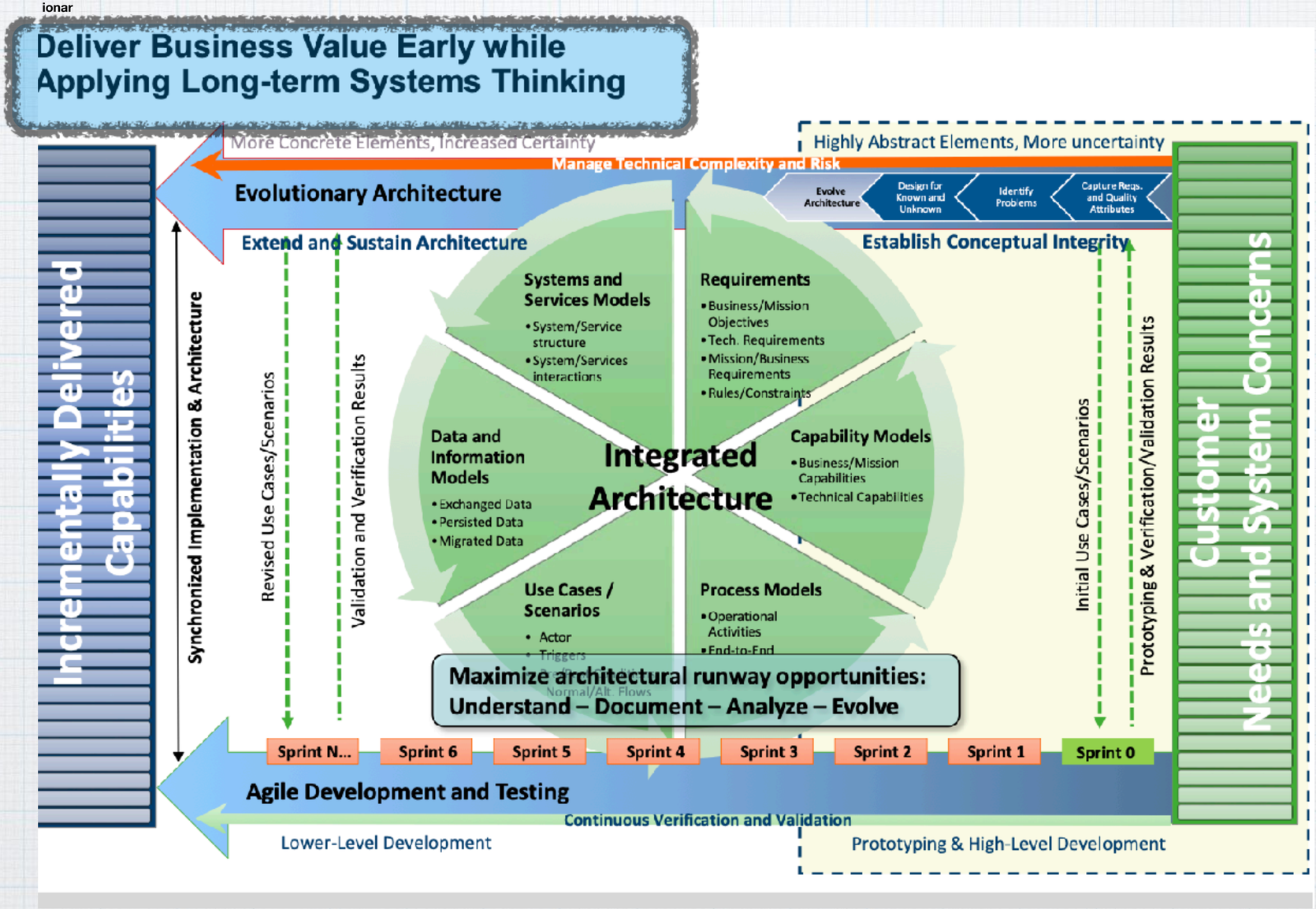
Planguage does this.

Validate the Design, using IET or Evo

https://tinyurl.com/StakeholderBook

A Reverse Chronology of Evolutionary Architecture and Agile Development

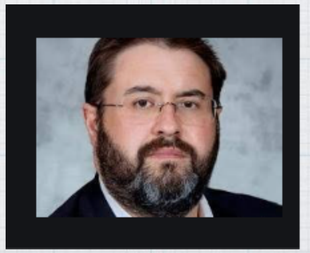
ionar



A Reverse Chronology of Evolutionary Architecture and Agile Development

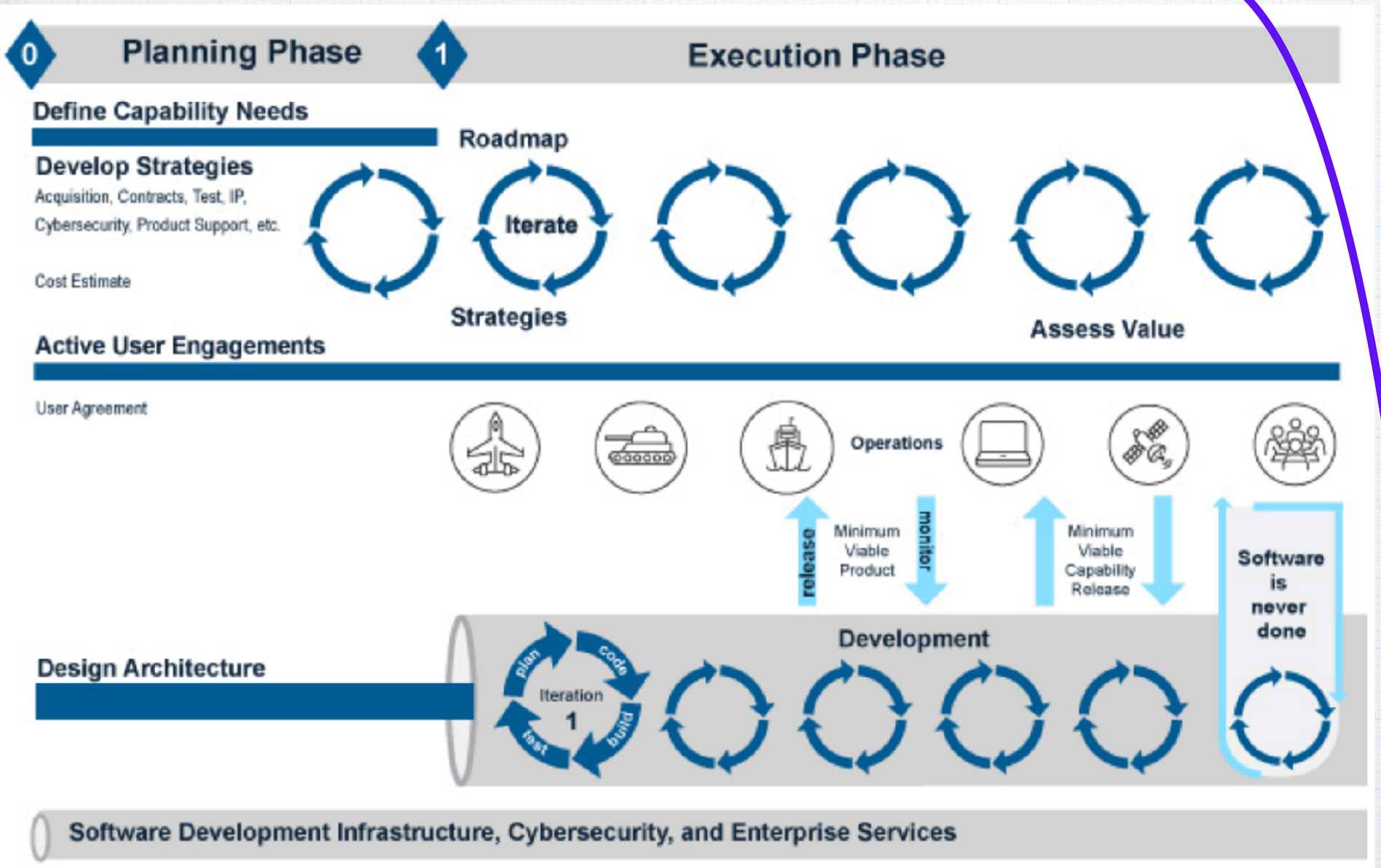
Gallagher Brian Hanif Mostafa Mielke Thomas

https://resources.sei.cmu.edu/asset_files/Presentation/2017_017_001_496091.pdf



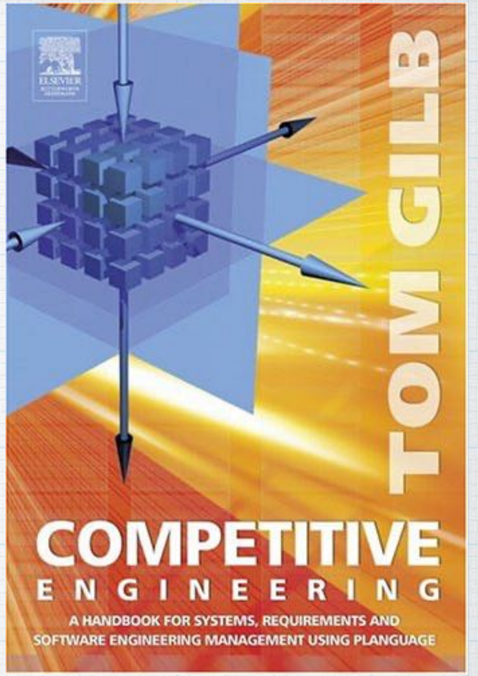
Brian Gallagher

US DoD Software Agility with Very clear 'Capabilities' (Qualities and costs)



"A rapid, iterative approach to software development reduces costs, technological obsolescence, and acquisition risk. To allocate resources to the most relevant capability needs, DoD or DoD component leadership will make software acquisition and development investment decisions within a framework that addresses tradeoffs between capabilities, affordability, risk tolerance, and other considerations."
(DoD 50000.87)

DOD INSTRUCTION 5000.87
OPERATION OF THE SOFTWARE ACQUISITION PATHWAY
https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF?ver=virAfQj4v_LgN1JxpB_dpA%3D%3D
Oct 2 2020



Some of My Methods

Requirements, Design, Spec QC

The Secret of Success

This book is going to go into some technical detail, on how we can be *fairly* sure of delivering successful stakeholder values, on time and under budget. Think of *'moving Success **up**, from less than 50%, to more than 99%'*. (Search 'failure rates' for IT and organizational projects, keep in mind that '*not* full Success' is *partial*, if not *full*, failure).

1. **A *clear agreed* definition** of the state of success (value and constraint levels). Requirements Engineering. [1, VR]. *'Decide to agree'*. [38, Generative Change]

2. **Systems Engineering Architecture** [3, SEA]: conscious quantitative design, towards the clear objectives and constraints. *"Design for Success!'*.

3. **Decomposition** [3, Decomposition Chapter] of *big* architectures, into much smaller, implementable sub-architectures. *'Prepare small increments'*.

4. **Risk-Consciousness Engineering**. [3, Risk Chapter] For every architecture idea, we take the time and trouble, to analyze and specify, the **'success potential'** (using 'Impact Estimation' [2]) of our architectures (aka strategies) in terms of *measurable* knowledge, and knowledge *sources*, for all concurrent value-objectives and resource constraints. A common *Science* tactic.

5. **Priority Engineering**: we use our analysis data, to compute the 'best overall value sub-architecture for the *remaining* budgeted resources', our *priority increment*, to deliver next, to the system, in the forthcoming incremental value delivery.

Steps 1-5 usually take 2 to 5 days of planning work. Startup Planning [7, SW]

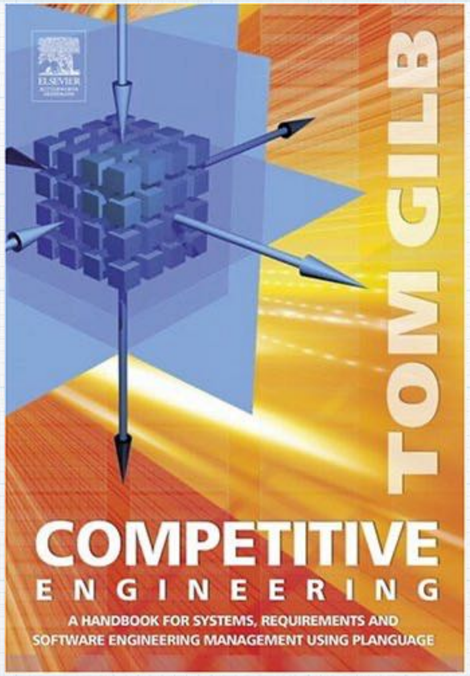
6. **Scientific Inquiry:** We measure and learn, from all 'incremented to date', delivered values, and incurred costs, and stakeholder reactions, at each value increment. Get good feedback early and frequently.

7. **Engineering Agility**: If the Values, costs, or stakeholder reactions are not good enough, we immediately *re-engage the architect*, or strategic planner, to analyze the root cause, and to design better (Dynamic Design to Value and Cost) and to try out new ideas, until they succeed, or we until we decide give up. 'Give up' on perhaps a *limited* requirement, or ambition, not necessarily everything.

Tom Gilb, SUCCESS : Super Secrets & Strategies for Efficient Value Delivery in Projects, Programs, and Plans, Book Folder, ***tinyurl.com/SUCCESSGilb.*** *October 2021. See Notes for slides and video: "Success Engineering Leadership"*

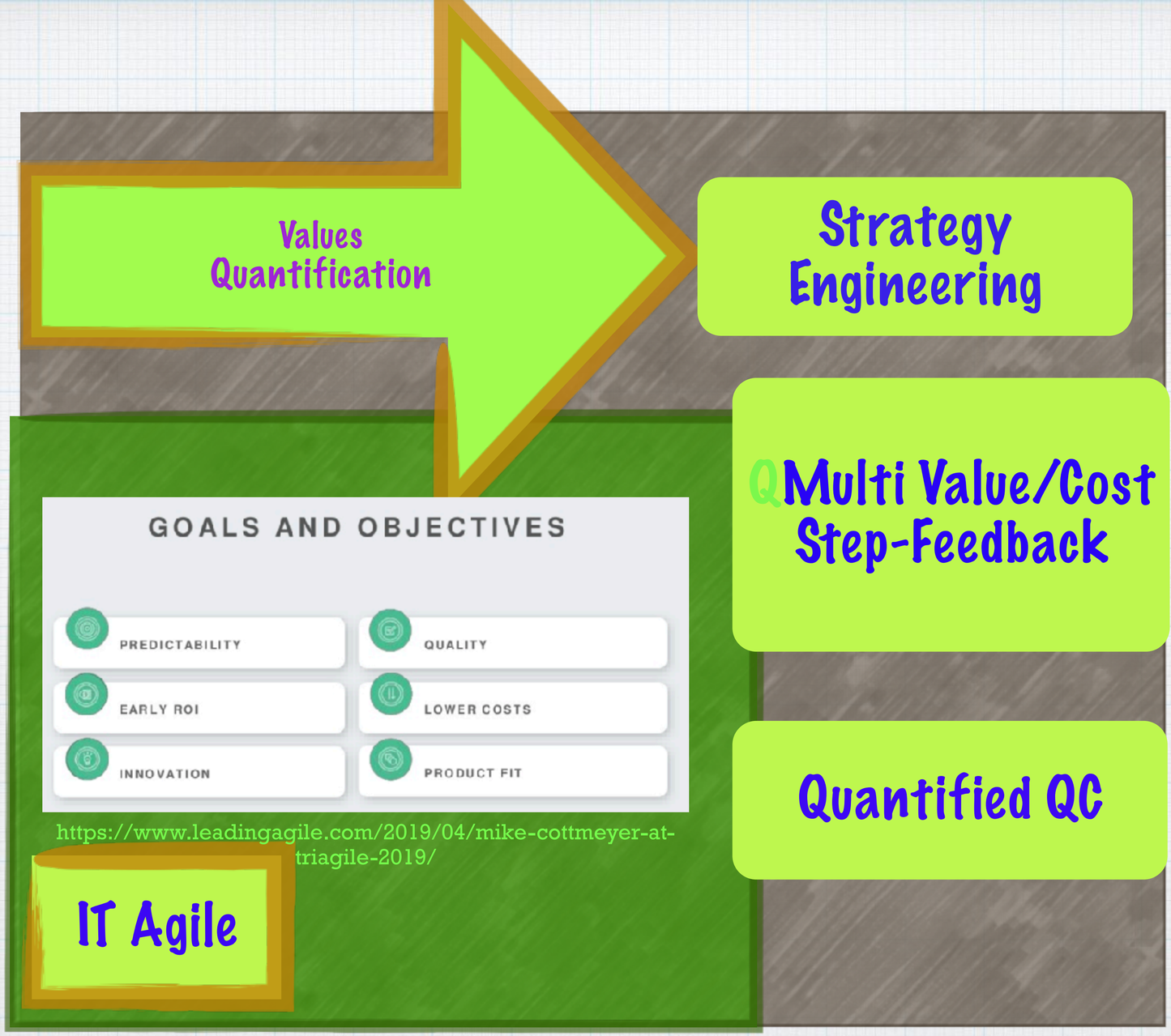
The Managility Hypothesis about better agile methods.

- 'Managility: Agile Methods
  - Quantification
    - Allowing engineering, science and increments
  - Decomposition
    - For early results, in a value stream
  - Dynamic Prioritization
    - Best value/costs stream
  - Corporate Learning
    - Methods Efficiency
    - Strategy Efficiency
    - Templates (Stakeholders, Scales of Measure, Evaluation Questions)
    - Digitization of Planning:
      - Use of digitization of planning, development and operational data
      - Moving towards maximum AI Help.
  - Preventative Quality Assurance
    - Defect Prevention rather than late detection and cure.
  - Systems Enterprise Architecture
    - Architecture to enable long-term rapid change and adaptability
    - Architecture Engineering to meet multiple critical values and constraints
  - Adoption of Strong: Concept Definitions, Policies, Principles, and Processes.
  - Methods Modularization: methods can be adopted and tried out gradually, and replaced whenever better methods come along, or are useful in a particular situation. Nothing is 'holy' except good results.



https://www.gilb.com/p/competitive-engineering
(free pdf)

Planguage



'Managility' is incremental improvement
in the direction of cost-effective quantified methods

Source 2020. tiny.cc/ManagilitySlides

Managility Principle 10. DIGITAL AGILITY:

Digital planning instruments will normally be needed to keep track of all these planning and decision-making elements, and helping us to sense the need for changes, sand to react quickly and accurately.

- Advantages of Digital Planning Apps for well structured planning languages based on metrics. (Like 'Planguage)
- Can handle very large scale of plans, many levels
- Good at remembering relationships (Stakeholders, Values, responsibilities)
- Can recompute priorities based on changes
- Can produce graphical selected views
- Can simplify when needed, but keep all detail
- Can deal with multiple dimensions at same time
- Can spot defects, inconsistencies and problems
- Can enforce governance of concepts (like strategy impact, evidence)
- Can provide libraries, corporate learning, for concepts, metrics.
- Cannot 'forget'. Easier to access from distributed sites.

ValPlan Info
gilb.com/valplan

Energy Domain Overview Table

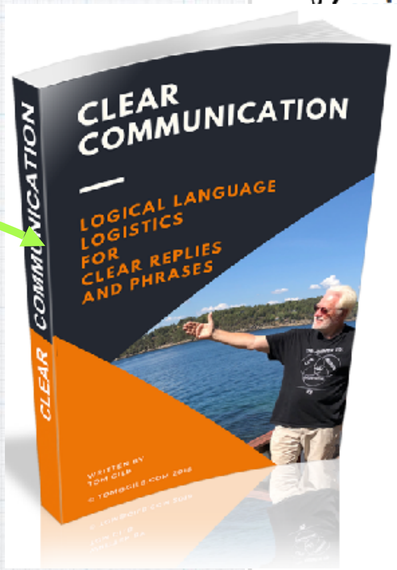
Settings... | + Add | Sort | Duplicate... | Δ: INCREMENTAL | Help | Show Sidebar

Requirements		New Legislation	Comms Plan	Ne
Consumer Costs Status: 0 → Wish: 100 % R... % relative level of [Energy Costs] ... [Energy Costs = { Consumer Monthly Ou...] 20th September 2017	Δ: Δ%:	70 70 % 70%	50 50 % 50%	???? 0 %
Reputational Damage Status: 0 → Wish: 0 *No qualifiers* ?	Δ: Δ%:	???? 0 % ????	???? 0 % ????	???? 0 %
Safe Transition Status: 0 → Wish: 0 *No qualifiers* ?	Δ: Δ%:	???? 0 % ????	???? 0 % ????	???? 0 %
Consumer Costs Status: 0 → Tolerable: 120 % R... % relative level of [Energy Costs] ... [Energy Costs = { Consumer Monthly Ou...] 20th September 2017	Δ: Δ%:	???? 0 % ????	130 108 % 108%	???? 0 %
Sum Of Values:	Σ%:	70 %	158 %	0 %
UP FRONT ONE OFF PAYMENTS Status: 0 → Budget: 10 £ B... £ Billion[Payment Type] to [Recipien.. [Payment Type = { <All> }, 20th September 2017	Δ%:	10 % 10%	3 30 % 30%	???? 0 %
Sum Of Development Resources:	Σ%:	10 %	30 %	0 %
Value To Cost:		7.00	5.30	0.00

Source 2020. tiny.cc/ManagilitySlides

Summary
'Quantify the Critical Causes you Care About'

- For any current management style, even if not formally 'agile':
- Much better QUANTIFICATION, and other CLARIFICATION of your most-critical value objectives.
- Multi-dimensional QUANTIFICATION of your strategies, for meeting goals.
- SYSTEMS ENGINEERING for large complex managed 'Systems'
  - (for organizations, products, services, human-value improvements)
  - = management agility. ('Managility')



"Clear Communication"
€14. https://www.gilb.com/store/oJCCxtsM

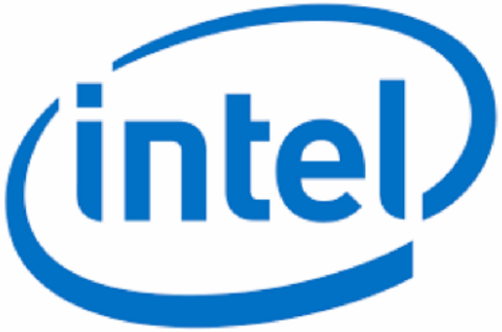
Source 2020. tiny.cc/ManagilitySlides

Energy Domain Overview Table

Settings... | + Add | Sort | Duplicate... | Δ: INCREMENTAL | Help | Show Sidebar

Requirements		New Legislation	Comms Plan	Ne
Consumer Costs Status: 0 → Wish: 100 % R... % relative level of [Energy Costs] ... [Energy Costs = { Consumer Monthly Ou...] 20th September 2017	Δ: Δ%:	70 70 % 70%	50 50 % 50%	???? 0 %
Reputational Damage → Wish: 0 ers	Δ: Δ%:	???? 0 % ????	???? 0 % ????	????
Transition → Wish: 0 ers	Δ: Δ%:	???? 0 % ????	???? 0 % ????	????
sumer Costs → Tolerable: 120 % R... ive level of [Energy Costs] ... = { Consumer Monthly Ou...] mber 2017	Δ: Δ%:	???? 0 % ????	130 108 % 108%	????
ues:	Σ%:	70 %	158 %	0 %
UP FRONT ONE OFF PAYMENTS Status: 0 → Budget: 10 £ B... £ Billion[Payment Type] to [Recipien.. [Payment Type = { <All> }, 20th September 2017	Δ%:	10 10 % 10%	3 30 % 30%	???? 0 %
Sum Of Development Resources:	Σ%:	10 %	30 %	0 %
Value To Cost:		7.00	5.30	0.00

Intel Measures of Gilb Methods 2013



The Impact of Requirements on Software Quality across Three Product Generations

John Terzakis

Intel Corporation, USA
john.terzakis@intel.com

***Abstract*—In a previous case study, we presented data demonstrating the impact that a well-written and well-reviewed set of requirements had on software defects and other quality indicators between two generations of an Intel product. The first generation was coded from an unorganized collection of requirements that were reviewed infrequently and informally. In contrast, the second was developed based on a set of requirements stored in a Requirements Management database and formally reviewed at each revision. Quality indicators for the second software product all improved dramatically even with the increased complexity of the newer product. This paper will recap that study and then present data from a subsequent Intel case study revealing that quality enhancements continued on the third generation of the product. The third generation software was designed and coded using the final set of requirements from the second version as a starting point. Key product differentiators included changes to operate with a new Intel processor, the introduction of new hardware platforms and the addition of approximately fifty new features. Software development methodologies were nearly identical, with only the change to a continuous build process for source code check-in added. Despite the enhanced functionality and complexity in the third generation software, requirements defects, software defects, software sightings, feature commit vs. delivery (feature variance),**

ays from project
he second to the

, requirements
quality, multi-

ort paper [1] that

II. Product Backgrounds

The requirements for Gen 1 that existed were scattered across a variety of documents, spreadsheets, emails and web sites and lacked a consistent syntax. They were under lax revision and change control, which made determining the most current set of requirements challenging. There was no overall requirements specification; hence reviews were sporadic and unstructured. Many of the legacy features were not documented. As a result, testing had many gaps due to missing and incorrect information.

The Gen 1 product was targeted to run on both desktop and laptop platforms running on an Intel processor (CPU). Code was developed across multiple sites in the United States and other countries. Integration of the code bases and testing occurred in the U.S. The Software Development Lifecycle (SDLC) was approximately two years.

After analyzing the software defect data from the Gen 1 release, the Gen 2 team identified requirements as a key improvement area. A requirements Subject Matter Expert (SME) was assigned to assist the team in the elicitation, analysis, writing, review and management of the requirements for the second generation product. The SME developed a plan to address three critical requirements areas: a central repository, training, and reviews. A commercial Requirements Management Tool (RMT) was used to store all product requirements in a database. The data model for the requirements was based on the Planguage keywords created by Tom Gilb [2]. The RMT was configured to generate a formatted Product Requirements Document (PRD) under revision control. Architecture specifications, design documents and test cases were developed from this PRD. The SME provided training on best practices for writing requirements, including a standardized syntax, attributes of well written requirements and Planguage to the primary authors (who were

TABLE I: GEN 2 REQUIREMENTS DEFECT DENSITY

PRD Revision	# of Defects	# of Pages	Defects/ Page (DPP)	% Change in DPP
0.3	312	31	10.06	-
0.5	209	44	4.75	-53%
0.6	247	60	4.12	-13%
0.7	114	33	3.45	-16%
0.8	45	38	1.18	-66%
1.0	10	45	0.22	-81%
Overall % change in DPP revision 0.3 to 1.0:				***-98%***

Intel Case Study

J. Terzakis, "The impact of requirements on software quality across three product generations," 2013 21st IEEE International Requirements Engineering Conference (RE), Rio de Janeiro, 2013, pp. 284-289.
https://www.thinkmind.org/download.php?articleid=iccgi_2013_3_10_10012

Productivity was 233% higher

50X better planning quality Before release

21,000+ Engineers 2 decades

Notice the team 'learning process' for tech standards

Positive Motivation

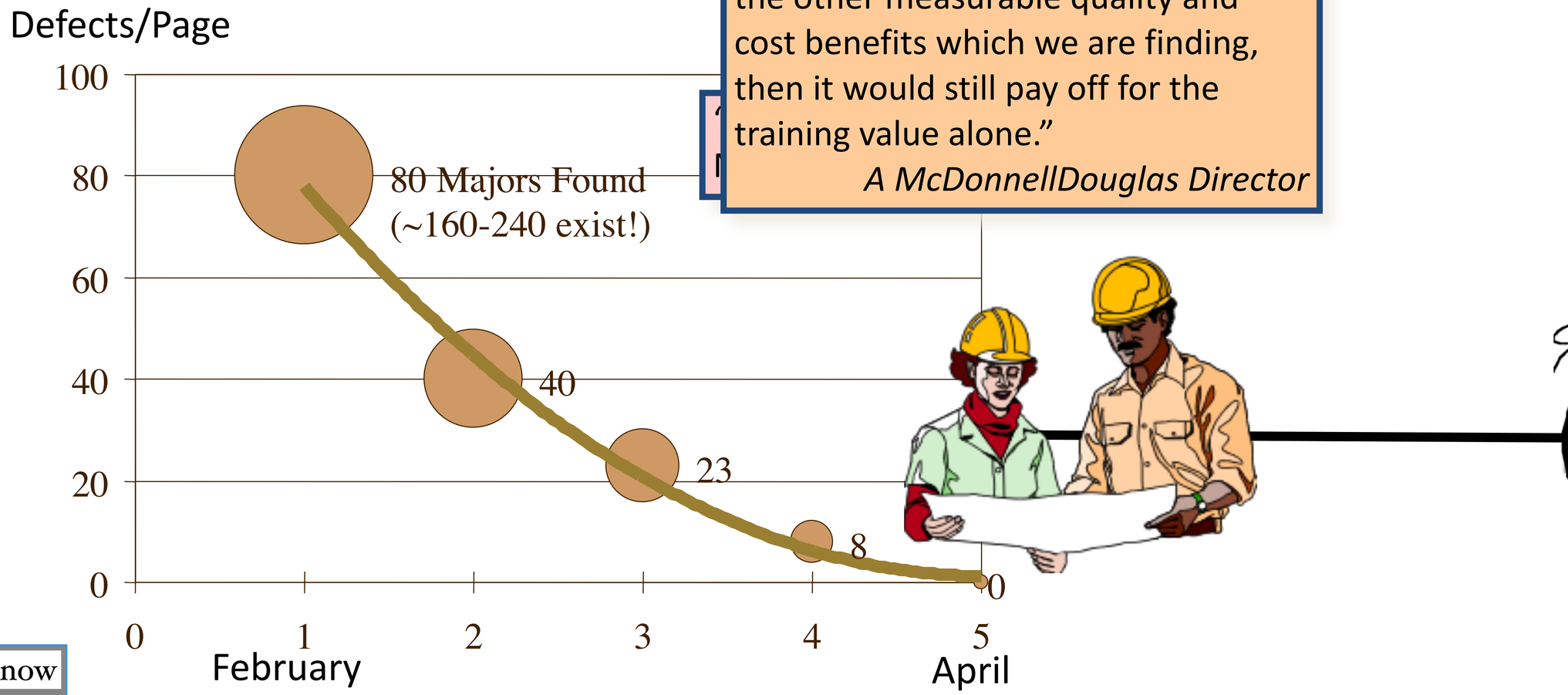
Personal Improvement

"We find an hour of doing Inspection is worth ten hours of company classroom training."

A McDonnell-Douglas line manager

"Even if Inspection did not have all the other measurable quality and cost benefits which we are finding, then it would still pay off for the training value alone."

A McDonnellDouglas Director



Inspections of Gary's Designs

Note: McDonnell Douglas is now Boeing
And I got similar results later
At Boeing
Renton, Washington

Why are these Planguage methods suggested?

- They are the best ones I know about.
- They are measured, studied, corporately adopted, publicly reported.
- They have decades of stable success, and close to zero failure rate.
- They are modular, and synergistic with each other.
- They can and have been digitized to use AI for complex thinking
- They handle multiple dimensions of concern simultaneously
- They apply at management and technical levels equally well
- They are scale free: they work on very large problems (but we need to invent something even better for managing the world better). UN Goals :)
- They are based on engineering and science practices

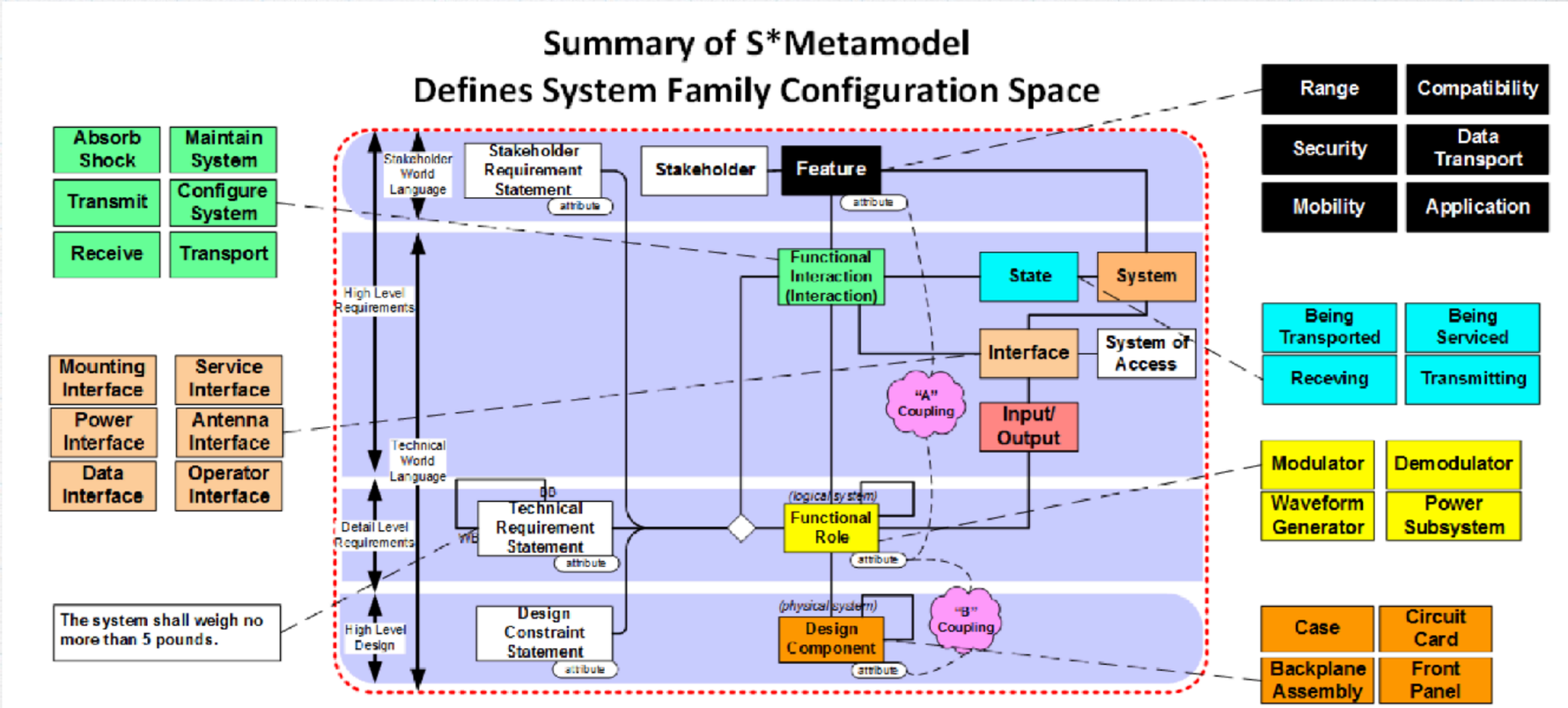


Figure 6: Product line family issues ultimately include the minimal system model issues.

http://www.parshift.com/s/ASELCM-02RC.pdf

RICK DOVE

"SCALE-FREE:
Practical Scaling Methods
for Industrial Systems Engineering"
lecture slides
http://concepts.gilb.com/dl892

Who or what controls or permits use of these Planguage methods?

- You do
- The methods are Creative Common Freeware.
- People can learn and practice from free internet link materials.
- No permission is needed to use the methods
- No approval or certification is necessary or desired: the methods are self-measuring;
- Real organizational results, in the short term and long term, are the judge of good methods.
- People, consultants, teachers, app developers, institutions, universities can offer their services to help people (for free or for payment) to learn and use these methods to best effect, quickly. But this is an option, not a pre-requisite.
- Judgement of the methods will be by detailed quantified case studies, and scientific research; as well as local measures and experience, in a given organization, and in sub-components of organizations, such as programmes, and projects.

QA Cycles on the left side of the V Model

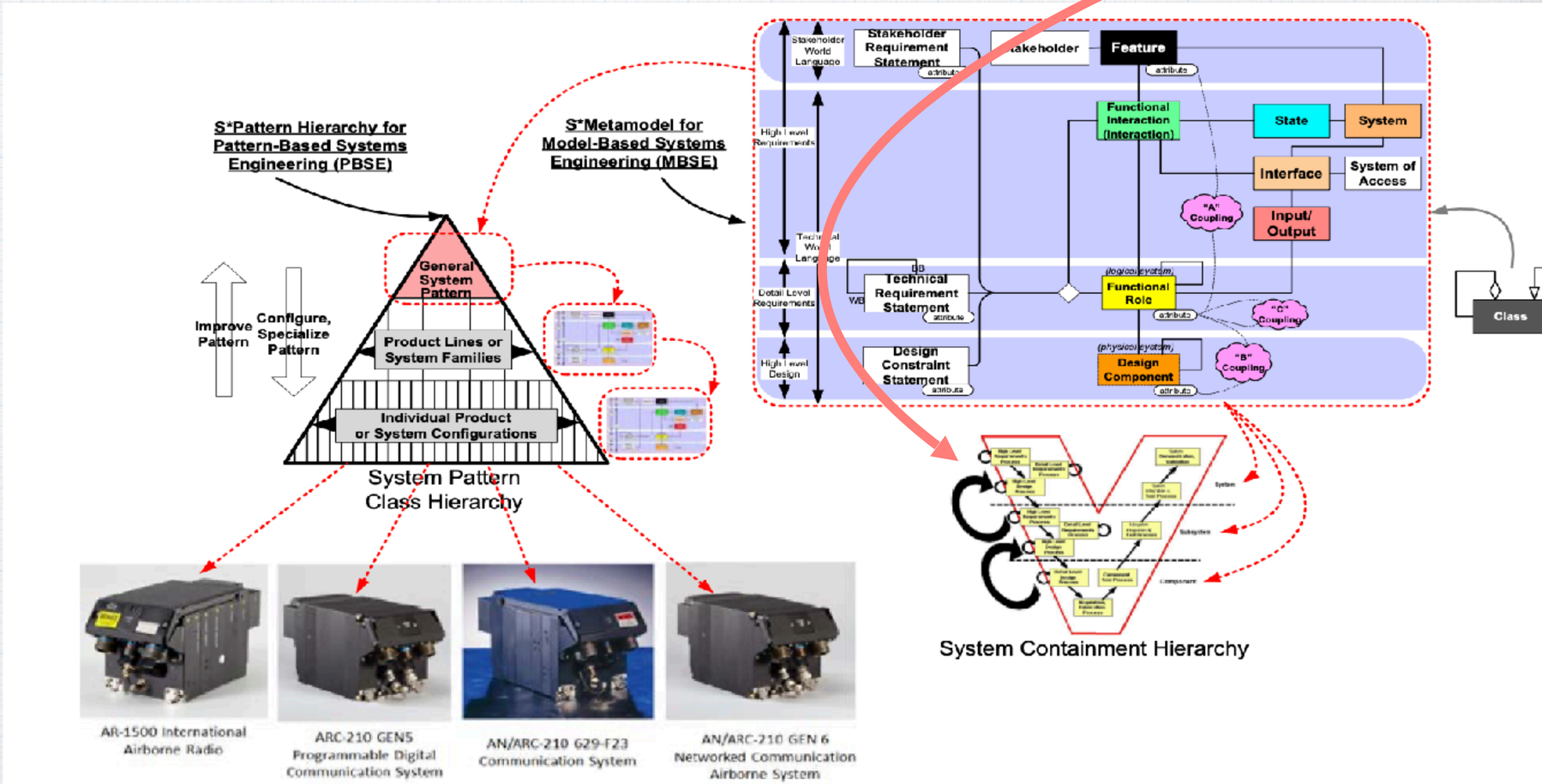


Fig. 7. Product lines configure varying products.

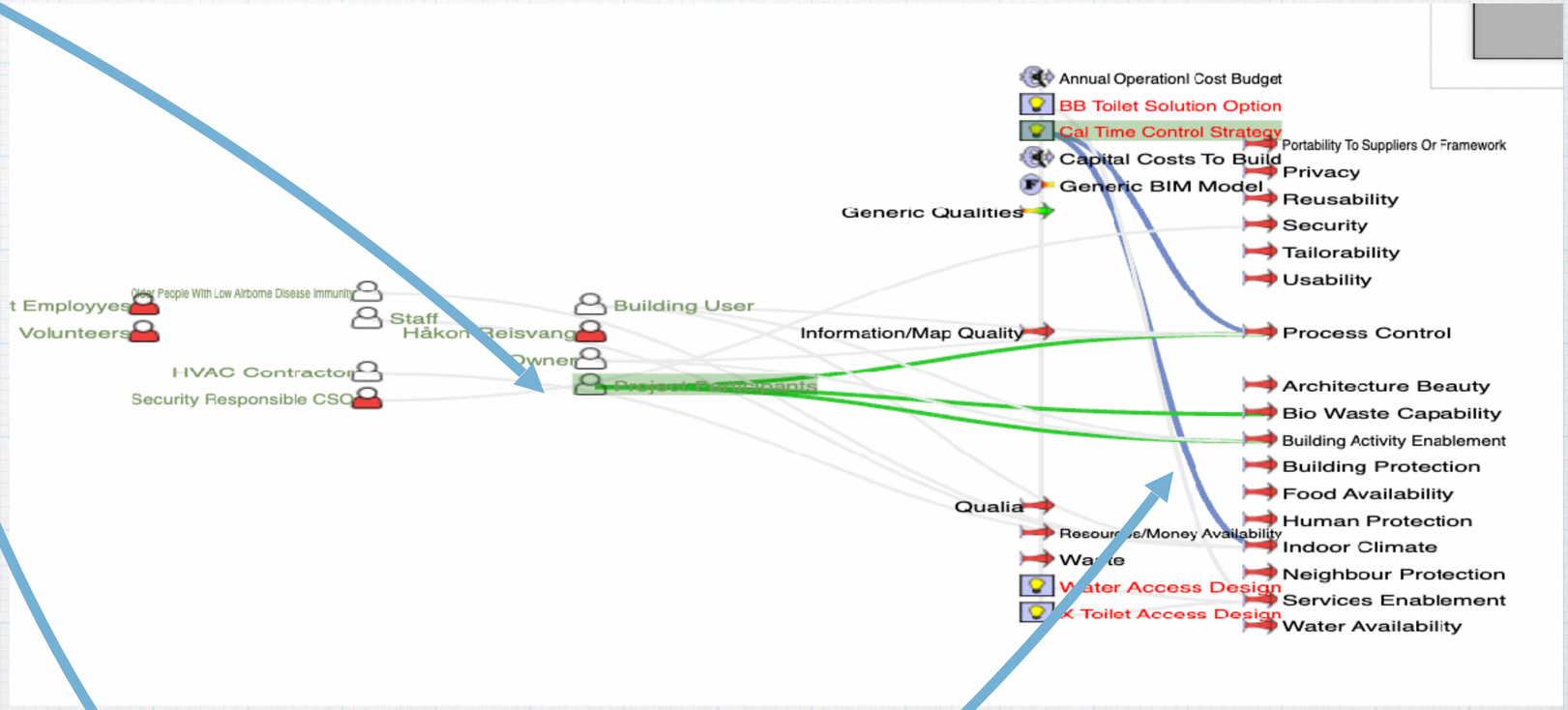
http://www.parshift.com/s/ASELCM-02RC.pdf

RICK DOVE

QA On Left of V

- Stakeholder Analysis
  - Stakeholder templates and models
  - Corrrelation analysis stakeholders vs. Values
- Requirements Specification
  - Planguage adoption, measurable Values and costs
  - Spec QC on requirements
  - ValPlan Tool
  - Impact Estimation Tool to test reality of requirements
  - Defect Prevention Process
  - Evo feedback cycle and learning
- Architecture
  - Detailed Design spec (ValPlan)
  - Requirement/Cost driven architecture Impact Estimation Tables
  - Specification Quality Control
- Module Decomposition
  - Planguage decomposition, ValPlan IET hierarchies
  - Independent Modules, Prioritized by Value/Cost
  - Deliver and measure Early
- Coding
  - Spec QC, Defect Prevention, QC as written, TDD
- Test Planning: Spec QC, Requirements Meters,

Planning Specification Diagram at Spec Objects Level (ValPlan)



A ValPlan Diagram linking known digital relations between stakeholders and requirements

Notice some stakeholders have no relations at all, and some Requirements (arrows) have no stakeholders. This allows us to do simple completeness checks when planning complex systems.

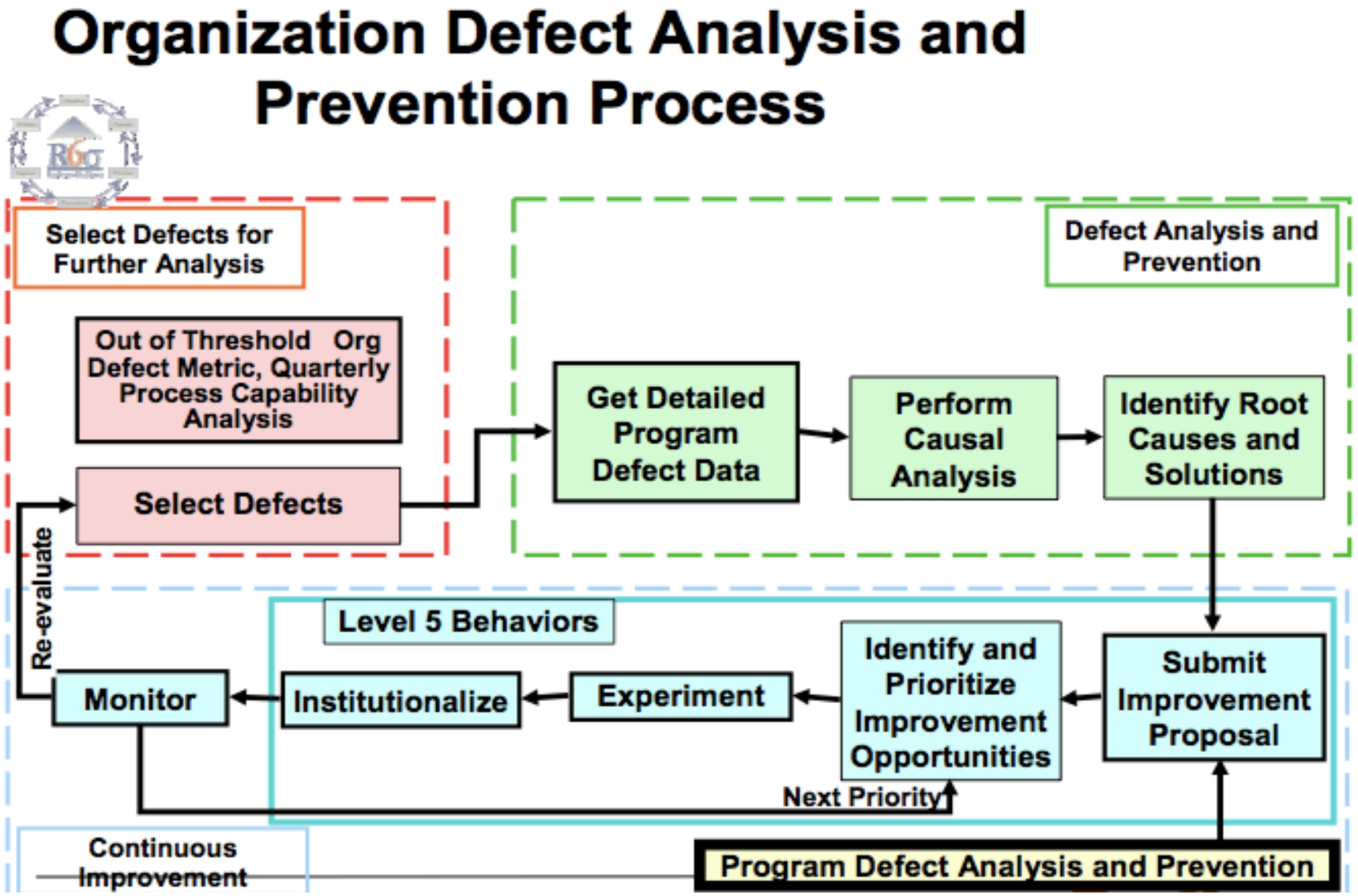
One design (lightbulb) impacts 2 Requirements (Arrows)

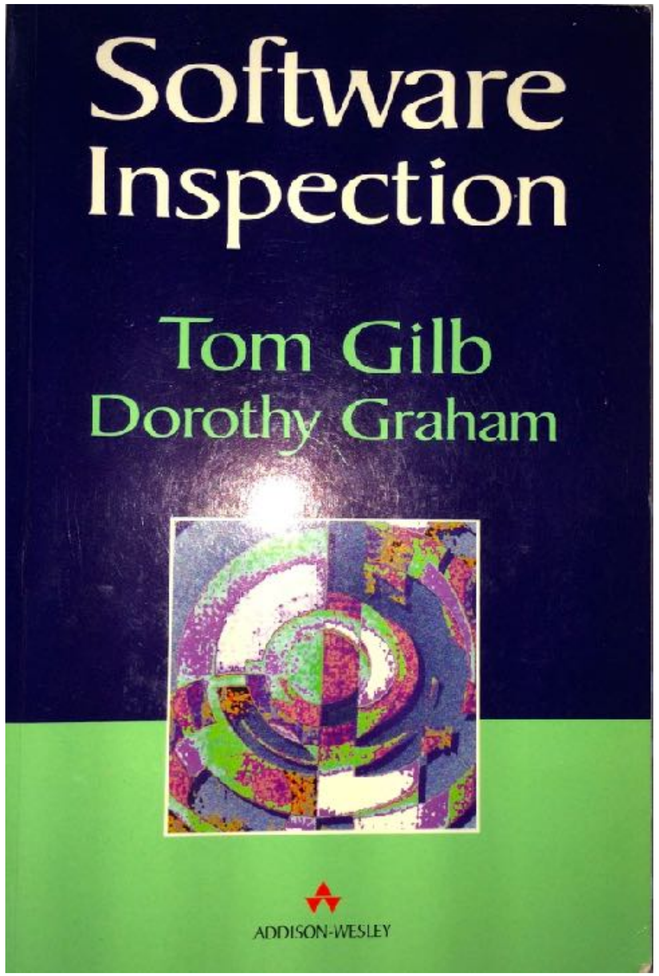
DEFECT PREVENTION PROCESS. DPP

Note: Elon Musk does his own variation of this basic idea with a vengeance.
Each team has an everyday responsibility to detect systemic problems and attempt To correct them 'same day'

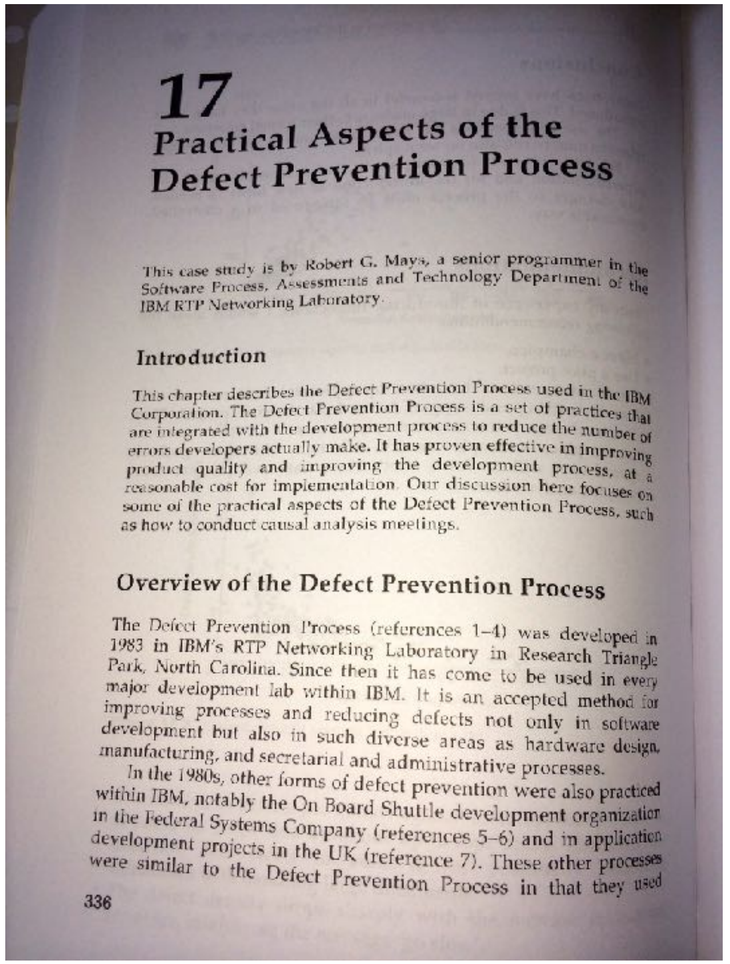
Free Book Link to Musk's Methods. 2022
https://tinyurl.com/MusksMethods

The DPP Process





Chapter 17 Robert Mays, IBM



17
Practical Aspects of the Defect Prevention Process

This case study is by Robert G. Mays, a senior programmer in the Software Process, Assessments and Technology Department of the IBM RTP Networking Laboratory.

Introduction

This chapter describes the Defect Prevention Process used in the IBM Corporation. The Defect Prevention Process is a set of practices that are integrated with the development process to reduce the number of errors developers actually make. It has proven effective in improving product quality and improving the development process, at a reasonable cost for implementation. Our discussion here focuses on some of the practical aspects of the Defect Prevention Process, such as how to conduct causal analysis meetings.

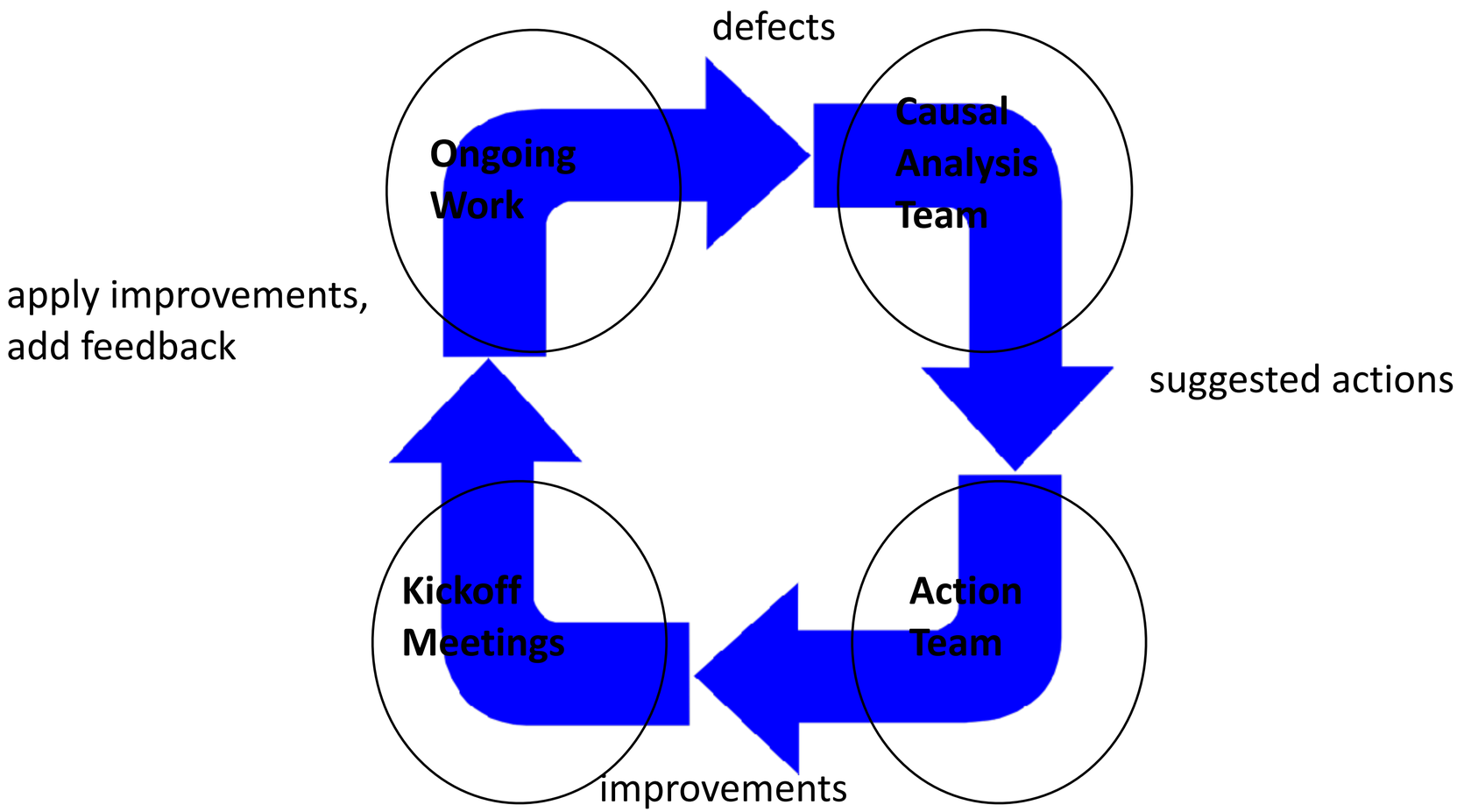
Overview of the Defect Prevention Process

The Defect Prevention Process (references 1–4) was developed in 1983 in IBM's RTP Networking Laboratory in Research Triangle Park, North Carolina. Since then it has come to be used in every major development lab within IBM. It is an accepted method for improving processes and reducing defects not only in software development but also in such diverse areas as hardware design, manufacturing, and secretarial and administrative processes.

In the 1980s, other forms of defect prevention were also practiced within IBM, notably the On Board Shuttle development organization in the Federal Systems Company (references 5–6) and in application development projects in the UK (reference 7). These other processes were similar to the Defect Prevention Process in that they used

336

How DPP Works



Centralized data collection and storage tools

Cost of Quality over Time: Raytheon

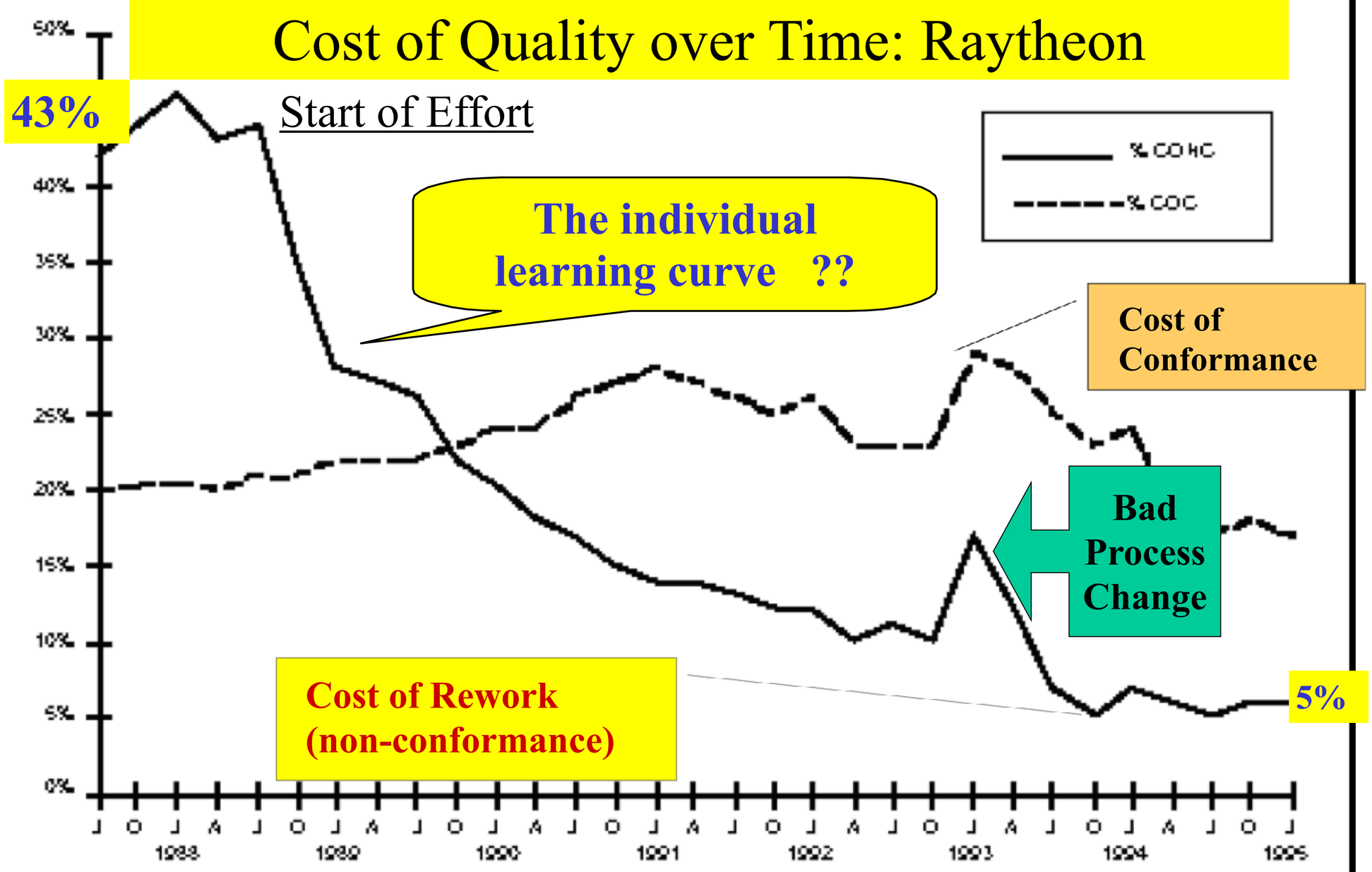
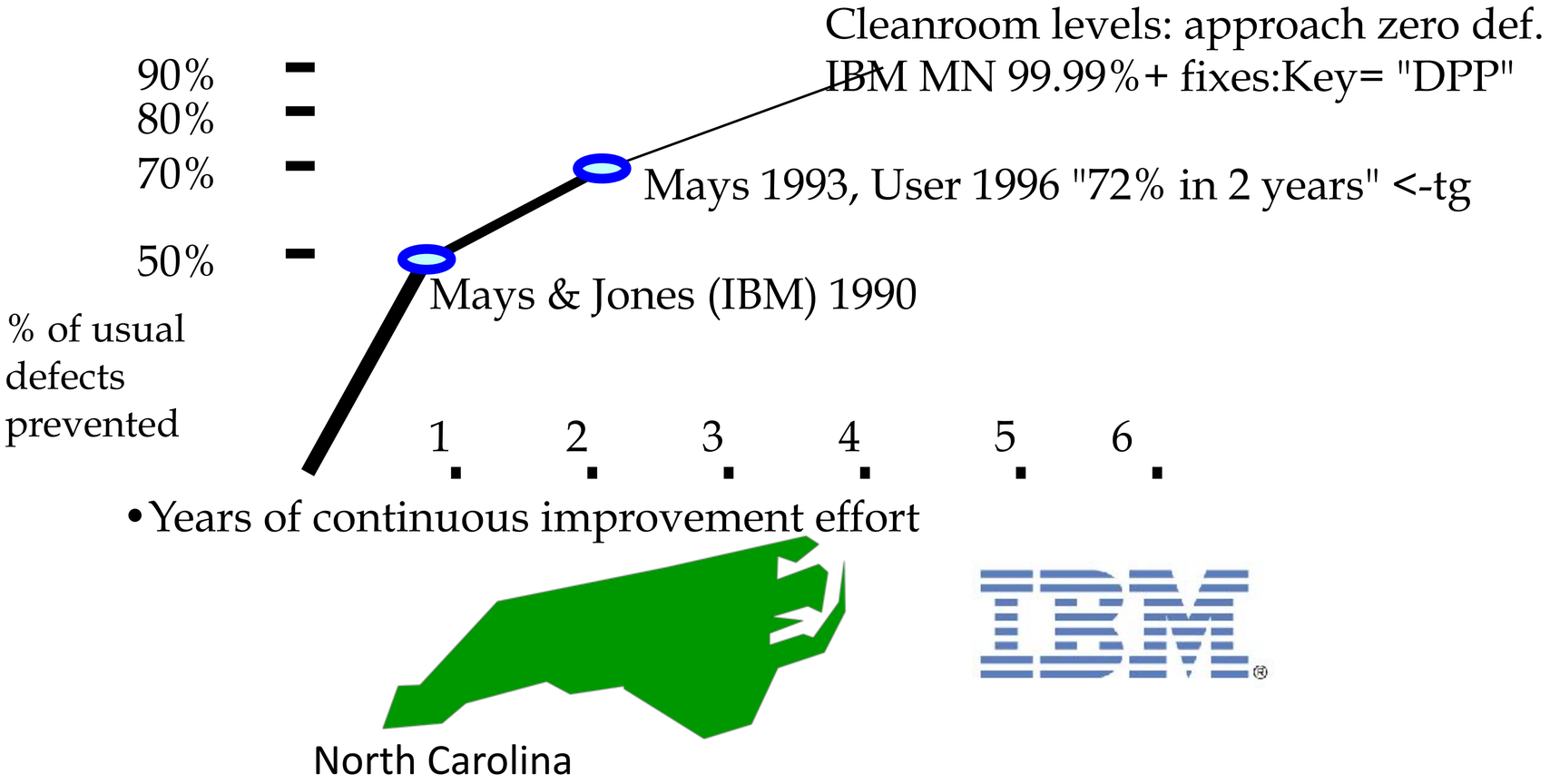


Figure 8: Cost of Quality Versus Time

Defect Prevention Experiences:
Most defects can be prevented from getting in there *at all*



IBM Research Triangle Park Networking Laboratory

Software Defect Prevention
Capers Jones:
'SOFTWARE DEFECT ORIGINS AND REMOVAL METHODS' www.Namcook.com

1. **Joint Application Design (JAD)**
2. **Quality function deployment (QFD) for software**
3. **Using the SEMAT approach (Software Engineering Methods and Theory)**
4. **Certified reusable requirements, architecture, and design segments**
5. **Certified reusable code**
6. **Certified reusable test plans and test cases (regression tests)**
7. **Kanban for software (mainly in Japan but spreading globally)**
8. **Kaizen for software (mainly in Japan but spreading globally)**
9. **Poka-yoke for software (mainly in Japan but spreading globally)**
10. **Quality circles for software (mainly in Japan but spreading globally)**
11. **Six Sigma for Software**
12. **Achieving CMMI levels => 3 for critical projects**
13. **Using quality-strong methodologies such as RUP and TSP**
14. **Embedded users for small projects < 500 function points in agile projects**
15. **Formal estimates of defect potentials and defect removal before starting projects**
16. **Formal estimates of cost of quality (COQ) and technical debt (TD) before starting**
17. **Quality targets such as > 97% defect removal efficiency (DRE) in all outsource contracts**
18. **Function points for normalizing quality data**
19. **Analysis of user-group requests or customer suggestions for improvement**
20. **Formal inspections (leads to long-term defect reductions)**
21. **Pair programming (leads to long-term defect reductions)**
22. **Hardware architecture changes to increase security**
23. **Software architecture changes to increase security**

Defect Containment: Get them while they are still young

- *Defect Containment focus*
  - Finding defects in the stage they were introduced and as early in the lifecycle as possible
  - Eliminating escaping defects
- *Defect Prevention focus*
  - Preventing the occurrence of an individual defect or group of defects

Software Defect Containment Matrix

Stage Detected	Stage Originated							
	Requirements	Design	Code and Unit Test	SW Integration	SW Quality Test	System Integration and test	SW Maintenance	Total
Requirements	1,515							1,515
Design	1,181	1,555						2,736
Code and Unit Test	402	912	2,421					3,735
SW' Integration	200	420	1,525	37				2,182
SW Quality Test	191	223	370	7	1			792
System Integration and Test	89	114	114	5	0	10		332
SW Maintenance	0	0	0	0	0	0	0	0
Total	3,578	3,224	4,430	49	1	10	0	11,292

- Overall defect containment which equal the total number of defects caught in-stage/total defects:
  - (1,515+1,555+2,421+37+1+10+0)/11,292 = 49 %

Defect Containment

AND

Defect Prevention

https://slideplayer.com/slide/15829099/

Musk's Methods

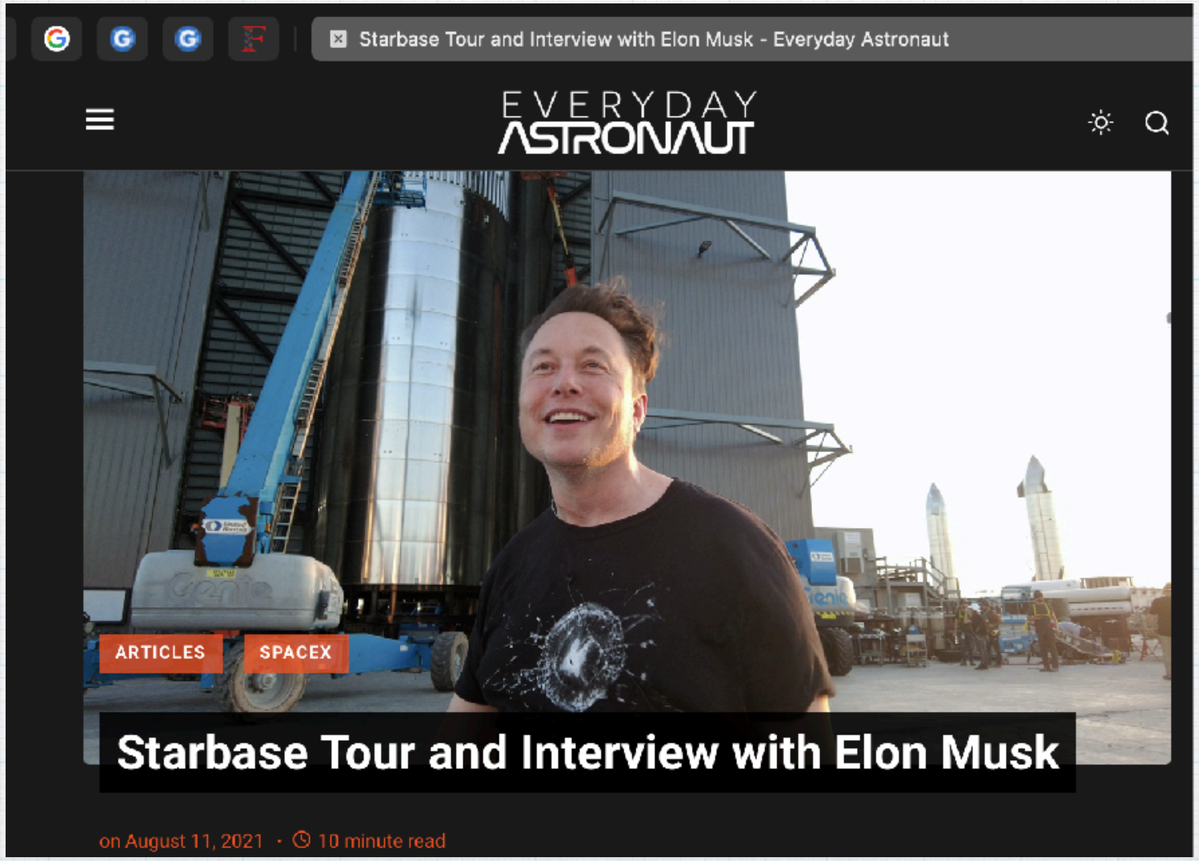


Figure 1.2.2 https://www.youtube.com/watch?v=t705r8ICkRw&t=804s

Watch Videos 1
and maybe as I did 2 and 3
***Good advice*, if you ask me.**

Musk's Engineering Philosophy:

Musk overviewed his five step engineering process, which *must be completed in order:*

1. **Make the requirements less dumb.** The requirements are definitely dumb; it does not matter who gave them to you. He notes that it's particularly dangerous if an intelligent person gives you the requirements, as you may not question the requirements enough. "Everyone's wrong. No matter who you are, everyone is wrong some of the time." He further notes that "all designs are wrong, it's just a matter of how wrong."

2. **Try very hard to delete the part or process.** If parts are not being added back into the design at least 10% of the time, not enough parts are being deleted. Musk noted that the bias tends to be very strongly toward "let's add this part or process step in case we need it." Additionally, each required part and process must come from a name, not a department, as a department cannot be asked why a requirement exists, but a person can.

3. **Simplify and optimize the design.** This is step three as the most common error of a smart engineer is to optimize something that should not exist.

4. **Accelerate cycle time**. Musk states "you're moving too slowly, go faster! But don't go faster until you've worked on the other three things first."

5. **Automate**. An important part of this is to remove in-process testing after the problems have been diagnosed; if a product is reaching the end of a production line with a high acceptance rate, there is no need for in-process testing.

Additionally, Musk restated that he believes ***everyone should be a chief engineer***. Engineers need to understand the **system at a high level** to understand when they are making a bad optimization. As an example, Musk noted that an order of magnitude more time has been spent reducing engine mass than reducing residual propellant, despite both being equally as important.

Source: https://everydayastronaut.com/starbase-tour-and-interview-with-elon-musk/

Let me attempt a **reformulation**,

hoping to increase **clarity** of Musk's intent.

1. **DYNAMIC. CRITICAL REQUIREMENTS: State really critical requirements only, and be prepared to learn quickly about even smarter requirements.**

2. **DYNAMIC DESIGN PRIORITIZATION: Resist keeping designs, that are not *clearly justified* and *prioritized* by means of *cost-effectiveness*, in terms of their impacts on *our* requirements.**

3. **DYNAMIC DESIGN OPTIMIZATION: keep continuous motivation to challenge existing designs, and to improve *design cost-effectiveness*, using *new technology*, using *feedback from iterations*, and using *revised requirements*.**

4. **ACCELERATE: Speed up repetitive processes, by *re-design*, by *paralleling*, *automation*, *feedback*, and relaxing *unnecessary requirements*.**

5. **AUTOMATE: Automate processes with high automation payoff, and avoid bad automation.**

6. **SYSTEMS: Do Systems-Level Engineering Trade-offs (prioritization)**

https://www.youtube.com/watch?v=hTSVvQJHboE

Free Book Link to Musk's Methods.
https://tinyurl.com/MusksMethods

Car Safety

Leading by consistent long-term value vision: Safety

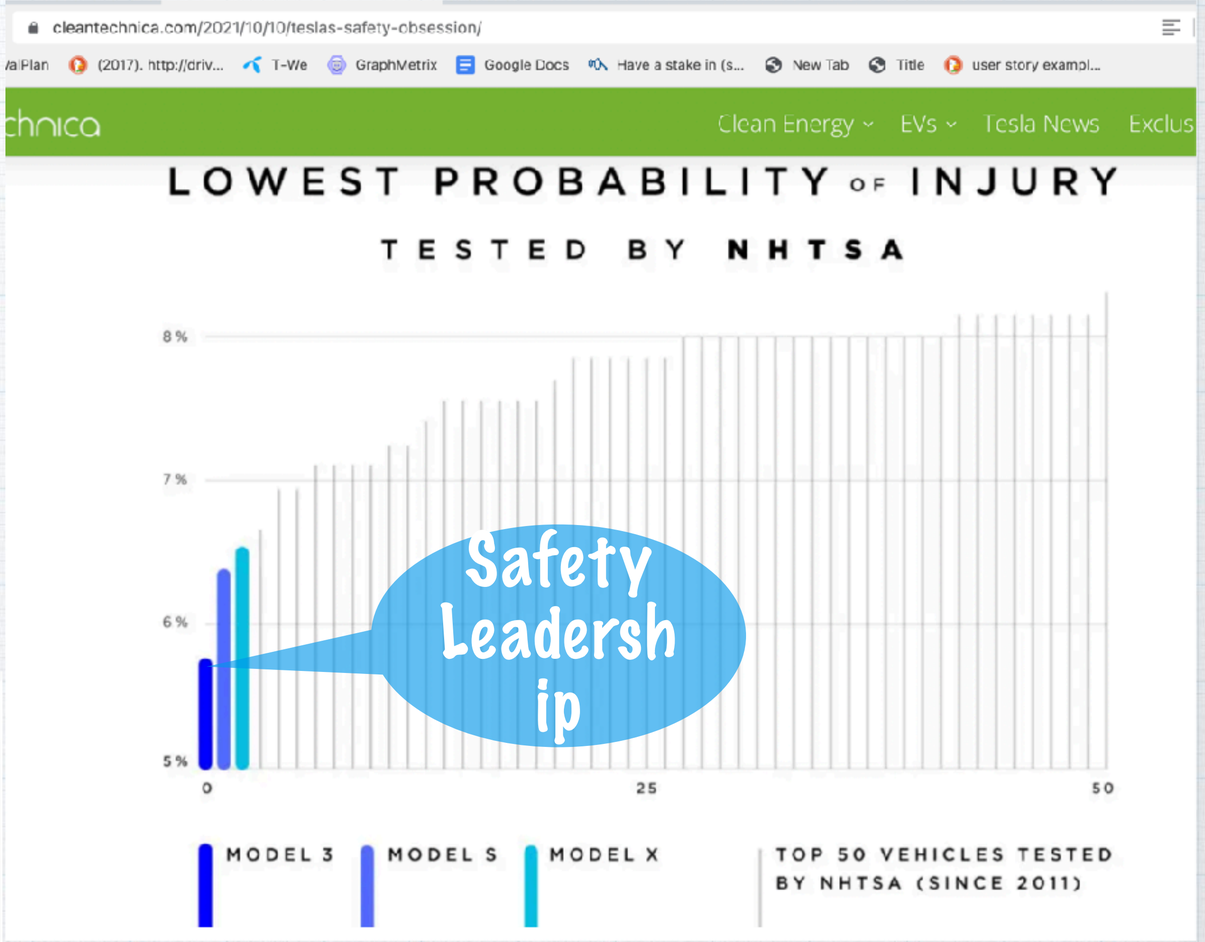
Relentless improvement

YOU CANNOT TEST QUALITY IN, YOU NEED TO DESIGN IT IN

Safety is *consciously designed* into the Tesla Cars. First by basic *design*, then by the *27 weekly increments* production line changes, to software and hardware.

"Elon mentioned that they are trying to design and develop the safest ATV (4-wheeler). ATVs are not very safe, but Elon is looking for ways to truly improve on their safety."

Photo courtesy of Tesla.



https://cleantechnica.com/2021/10/10/teslas-safety-obsession/

Use First Principles to get on a better path early

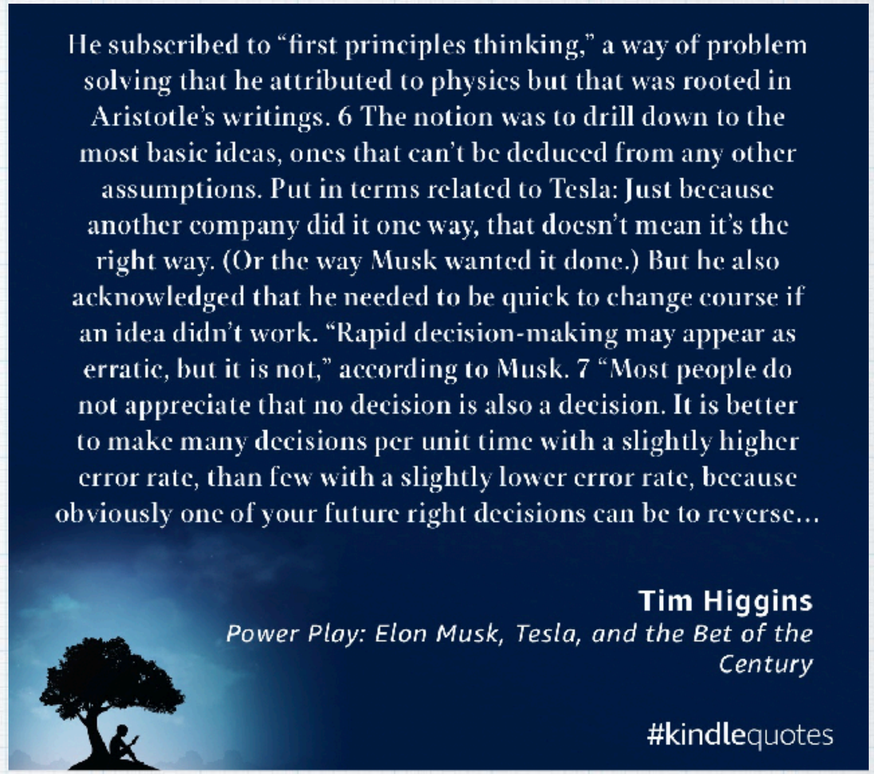
First principles thinking does not remove the need for continuous improvement, but it does alter the direction of improvement. Without reasoning by first principles, you spend your time making small improvements to a bicycle rather than a snowmobile. First principles thinking sets you on a ***different trajectory.*** *What are you trying to accomplish?*

Source: https://jamesclear.com/first-principles

My own 'Planguage' method interpretation is that we need to get out of the bad habit of specifying our objectives in terms of specific known designs or strategies. This is what Musk calls an 'analogy'. We need to **define the improvement dimensions (values, qualities, costs) we want**. Then ***design to meet them.*** https://leanpub.com/SysEntArchBook

What are called 'First Principles' here, are viewed by my Planguage methods as '*incremental design components*'. *What Musk is saying is 'do not specify your design ideas in terms of analogy to known complicated solutions'.*

Instead, build your design solutions, one basic component at a time. And, identify the most cost-effective, new technology, components, as you go. Don't get locked into old thinking, but think fresh to be competitive.

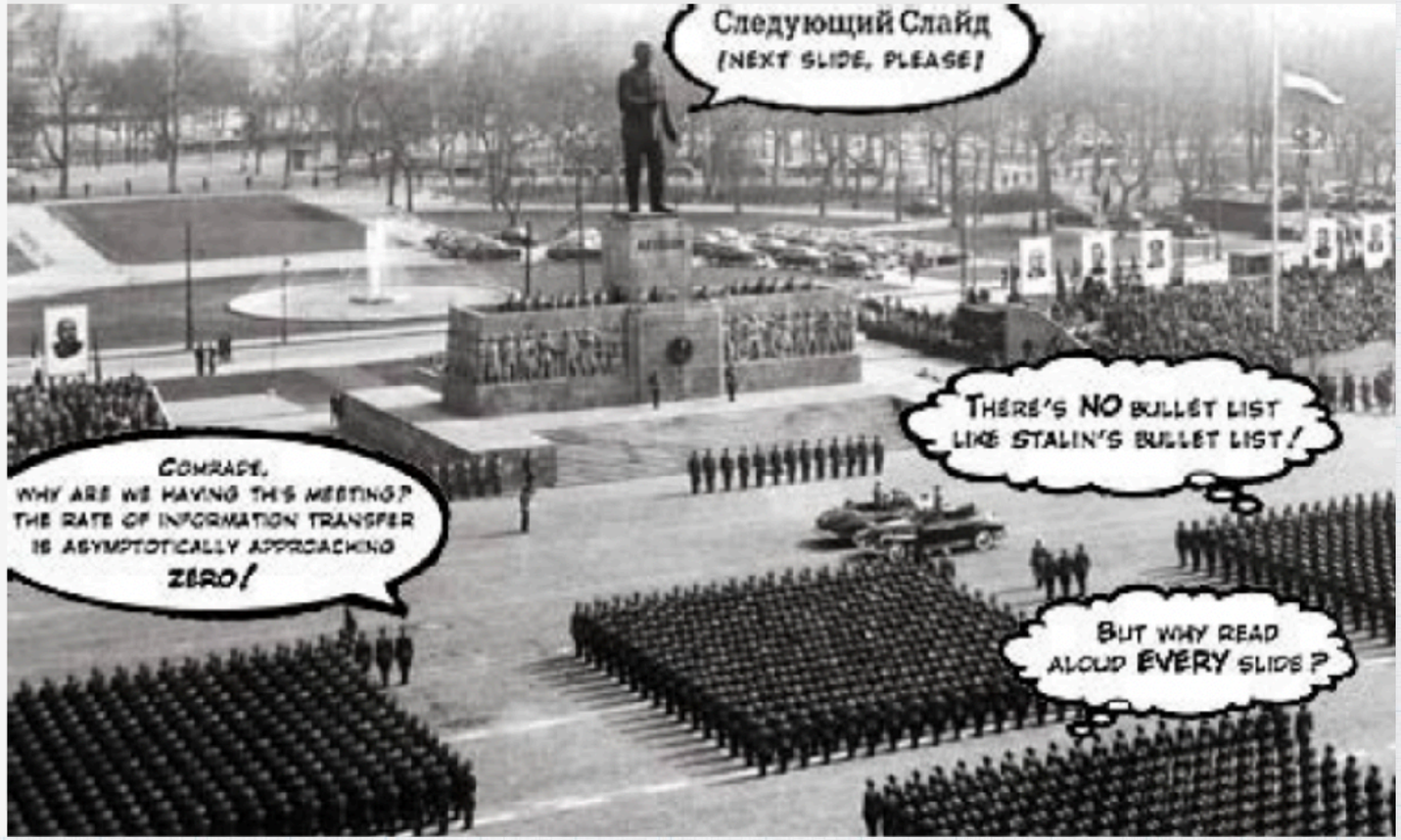


https://medium.com/the-mission/first-principles-and-the-art-of-thinking-like-elon-musk-98658bb36569

Bozos, Amazon, Narrative
No Powerpoint Presentations

* Source of practice Edward Tufte
* 6 Pages maximum text
* 20 minutes, all read in meet
* Sentence analysis, Bezos
  * Compare to Musk #1 Assume requirements and design are wrong.
* Gilb; Planalysis

Gilb, PLanalysis, Analyzing Garbage Specs. I 2020, book, 62 pages.
https://tinyurl.com/PLanalysisFree, PDF and EPub copies
A booklet with advice on how to analyze plans, and make them better.



https://www.slideshare.net/plowenthal/using-powerpoint-differently-when-teaching-online

Last Slide

Of the lecture, but detailed references are after this slide)

- tom@Gilb.com
- www.Gilb.com
- twitter.com/ImTomGilb. *
- www.linkedin.com/in/tomgilb *
- +47 920 66 705
- Honorary Fellow of BCS

* on twitter and Linkedin, I proudly announce free links to hopefully-useful deep-knowledge, slides, papers, videos, broadcasts, and books. I do not directly commercially-promote or sell. You can go to www.Gilb.com for sales pitches. I do not want your money, I want your mind, for your benefit.

Pictures from Spring edition of the Masterclass

7 / 41

Literature references
For
Further study

B Gilb Books. Page 1A

Main Current Books Written by Tom Gilb. Supports this book with detail.

B1.CE: **Competitive Engineering** (paper or digital 2005).

The **definition of the Planguage. A Handbook and a Planguage standard.**

https://www.gilb.com/p/competitive-engineering (free pdf)

and paper via Amazon (Kindle and paper)

https://www.amazon.com/dp/0750665076/ref=rdr_ext_sb_ti_sims_2

B2:VP: **Value Planning**

"Value Planning. **Practical Tools for Clearer Management Communication"**

Digital Only Book. 2016-2019, 893 pages, €10

https://www.gilb.com/store/2W2zCX6z

This book is aimed at management planning. It is based on the Planguage standards in 'Competitive Engineering' (2005). It contains detailed practical case studies and examples, as well as over 100 basic planning principles.

The 'Technoscopes' book (2018) is a condenses version of this with the 100 principles and some examples o quotes related to the principles.

The 'Vision Engineering' book [B3, VE] is. Short (60 pages) top manager oriented overview of the ideas in Value Planning, and it is the **front end (the real book)** of the Value Planning book.

B2.**Decomposition**: https://tinyurl.com/VPDecomposition

B2: **Prioritization**: https://www.dropbox.com/sh/34llx1a7ckyagxl/AAA0pDzSxN5WmoP9lOKR0Mpca?dl=0

P18: **Risk Management Chapter 7, (70 pages) VP Value Planning** Book. https://tinyurl.com/RiskMgtVP

B2: Chapter 3 **Levels of interest,** https://www.dropbox.com/sh/xbzn5s8imf9vla0/AAB8h-OFvQmJ_w3wNhrDxa9_a?dl=0

B2: Chapter 8 **Delegation Outsourcing Contracting,** https://www.dropbox.com/sh/eub01zkvybl2q8k/AAD6cUUIOqc00aTPK2OZx6gua?dl=0

B3:VE.**Vision Engineering**.

"Value Planning: **Top Level Vision Engineering"**

How to communicate critical visions and values quantitatively. Using The Planning Language.

http://concepts.gilb.com/dl926

A 64 Page pdf book. Aimed at demonstrating with examples how top management can communicate their 'visions' far more clearly.

The many case studies of
Use of my methods
Are in
the detailed References
<—— in these books,

The most detailed is
the 'Value Planning' book [VP]

B Gilb Books. Page 1B

Main Current Books Written by Tom Gilb.

B4. FREE LINK TO **5 NEW DIGITAL BOOKS (B5 to B9)** WRITTEN SUMMER 2019, see also Videos and Slides, same title

https://www.dropbox.com/sh/adcrki52xo5zb36/AABMD_2GOX4rT6c-HRCmT-Qua?dl=0

B5:VR. **Value Requirements** book

https://www.dropbox.com/s/hxg1rx9rzesw2id/Value%20RequirementsPDF%20BEST%20%2070MBQ%20011019%202245%202.pdf?dl=0

B6:VD. **Value Design,** Book. July 2019

https://www.dropbox.com/s/ldrofca89sfwzur/Value%20Design%20MASTER%20B2607%20V1408.pdf?dl=0

B7:VM. **Value Management,** book August 2019

https://www.dropbox.com/s/7utbgxzcmahfj0c/Value%20Management%20MASTER%20B070819%20V160819.2252.pdf?dl=0

B8:VA. **Value Agile ,** tinyurl.com/ValueAgile, 2019

B9SP. **Sustainability Planning,** https://tinyurl.com/UNGoalsGilb, 2019

BOOK, See slides **[P2.2]** http://concepts.gilb.com/dl977

B10. **Books written Summer 2020.** https://www.dropbox.com/sh/tj1p6a3omlg9hx3/AABXuj_YnUmAFeRWOpGVvQtla?dl=0,

B11 **KEN: Knowledge Edu-Neering booklet**

https://tinyurl.com/KENGilb, 2020, **FREE CC.**

B12. **Governeering: Government Systems Engineering Planning**: https://tinyurl.com/Governeering, 2020

B13. **PLanalysis: A booklet with advice on how to analyze plans, and make them better,** 2020, https://tinyurl.com/PLanalysisFree

B14. **QUANTeer: The Art of quantifying your value ideas** https://tinyurl.com/Quanteer2020,

B15: 12?: **12 Tough Questions for Better Management**, https://tinyurl.com/12TOUGH

B16:SEA: **Systems Enterprise Architecture** (SEA) BOOK, Free Download

https://tinyurl.com/SysEntArchBook (2020)

https://www.dropbox.com/sh/o2g7ib3z2g2uzfw/AABtxPsj3rMYfc1ldGm80qDsa/Latest%20Master%20edit%20of%20Value%20Agile?dl=0&subfolder_nav_tracking=1

The 2018 5 Books, & Older Gilb Books

G10. **Technoscopes**, 2018
Technoscopes:
Tools for understanding complex projects https://www.gilb.com/store/Pd4tqL8s
Price €14B12. **Clear Communication**, 2018

G13. **Innovative Creativity**, 2018
'INNOVATIVE CREATIVITY' 124 pages €14 https://www.gilb.com/store/QMMQhn2g

G14. **100 Practical Planning Principles**, 2018
Based on the same 100 **Value Planning** sub-sections and principles.
100 Practical Planning Principles. Booklet €14 https://www.gilb.com/store/4vRbzX6X

G15. **PoSEM** 1988, **Principles of Software Engineering Management**, 1988, Pearson.
Chapter 15 Deeper Perspectives on Evolutionary Delivery, www.gilb.com/dl561,
Whole Book (Paper) https://www.amazon.com/Principles-Software-Engineering-Management-Gilb/dp/0201192462

G16. **Software Inspection**, 1993, https://www.amazon.com/Software-Inspection-Tom-Gilb/dp/0201631814

G17. **CLEAR COMMUNICATION Booklet**
"Principles of Clear Communication"
By Tom Gilb
DIGITAL BOOKLET €14
Published 31 August 2018
https://www.gilb.com/store/oJCCxtsM

G18. **Software Metrics**, 1976-7,
Gilb Tom. Software metrics. *Studentlitteratur AB Sweden*, 1976., Tom Gilb. *Software metrics*. Winthrop, 1977. (USA Hardcover). The term **Software Metrics** was coined by me here.

G19. **Life Design**, 2018
LIFE DESIGN Booklet €14
https://www.gilb.com/store/kCBGcG6L

Early (1976, 1979, 1988) Gilb Agile Quotes and References

HERE ARE SOME REFERENCES TO MY EARLY ‘AGILE’ IDEAS FROM MY 1976 BOOK AND PUBLISHED ARTICLES

Source: T. Gilb, **Software Metrics, October 1976-7**, (Studentlitteratur, Winthrop).
'Evolution is a designed characteristic of a system development which involves gradual stepwise change.' (p. 214)

On step results measurement and retreat possibility

'A complex system will be most successful if it is implemented in small steps and if each step has a clear measure of successful achievement as well as a "retreat" possibility to a previous successful step upon failure.' (p. 214)

On minimizing failure risk, using feedback, correcting design errors

'The advantage is that you cannot have large failures.

You have the opportunity of receiving some feedback from the real world before throwing in all resources intended for a system, and you can correct possible design errors before they become costly live systems.' (p. 214)

On total project time
'The disadvantage is that you may sometimes have to wait longer before the whole system is functioning. This is offset by the fact that some results are produced much earlier than they would be if you had to wait for total system completion.

It is also important to distinguish between a date for total system operation and a date for total "successful" system operation.' (p. 215)

On the general applicability

'Many people claim that their system cannot be put into operation gradually. It is all or nothing. This may conceivably be true in a few cases . . . I think we shall find that virtually all systems can be fruitfully put-in in more than one step, even though some must inevitably take larger steps than others.' (p. 215)

A measure of degree of evolution
'A metric for evolution is degree of change to system "S" during any time interval "t".' (p. 214)

On risk and predicting requirements
'Risk estimates plus/minus worst case are key to selection of step size', and 'Saving of analysis of future real world'. (p. 217)

The first remark is recognition that step sizing is determined by the need to control risk of failure. It is not small steps in themselves which are important. A large step may be taken if the risk is under control; for example by using contract guarantees or known technology. The second remark is recognition that the evolutionary method avoids the need to predict requirements and environments in the future; it allows us to wait until the future has arrived, to see the current requirements and the current environment.

On the scientific experiment analogy

'The concept of stability (where evolution is a technique for achieving stability) at individual levels of a system has the same usefulness as the concept of keeping all-factors-except-one constant in a scientific experiment. It allows systematic and orderly change of systems where the cause and effect may be more accurately measured without interfering factors, which may cause doubt as to the reason for good or bad results.' (p. 217)

'Systems may be specifically designed to go through a revolution in several phases, where only one level of the system is changed significantly at a time.' (pp. 217-8)

Evolutionary modularity design: conflict and priority
On p. 187 I raised the issue of 'Modularity division criteria', and gave six examples of rules for dividing software modules. Rule six was 'By calendar schedule of need of module' and the explanation for this rule was: 'Early implementation; evolutionary project develop.'

'Each rule can conflict with other modularization rules and with other design criteria. Resolution of the conflict can be achieved by a clearly stated set of priorities'.

This is a forerunner to the present perception of step design and selection being based on those elements of the total system which will contribute the greatest value towards stated objectives at the least development resource cost.

Later writings on the subject
The evolutionary idea was developed in my articles in the trade press:

'Evolutionary Planning and Delivery: an Alternative', Computer Weekly, 2 August 1979;

'Evolutionary Planning can prevent Failures', Computer Data, Canada, April 1979, p. 13;

'Realistic Time/Cost Data', Computer Weekly, 16 August 1979;

'Eleven Guidelines for Evolutionary Design and Implementation', Computer Weekly, 12 March 1981;

and 'The Seventh Principle of Technology Projects: Small Steps will Result in Earlier Success', Computer Weekly, 30 July 1981.

In all there were about 122 Gilb's Mythodology Columns in Computer Weekly (UK), which developed many of the ideas in this book.

Should any researchers wish copies of any of this, they are in my personal archives in NOrway. They were before internet of course. Let me know if you want to host copies of these for historic purposes.

As referenced above, my 1988 PoSEM book was full of my Evo ideas. Here is 1 chapter where I surveyed the history, and you can [illegible] book if you want the full picture.

Gilb: Principles of Software Engineering Management, (1988)

Pearson.

Chapt 15 Deeper Perspectives on Evolutionary Delivery

www.gilb.com/dl561