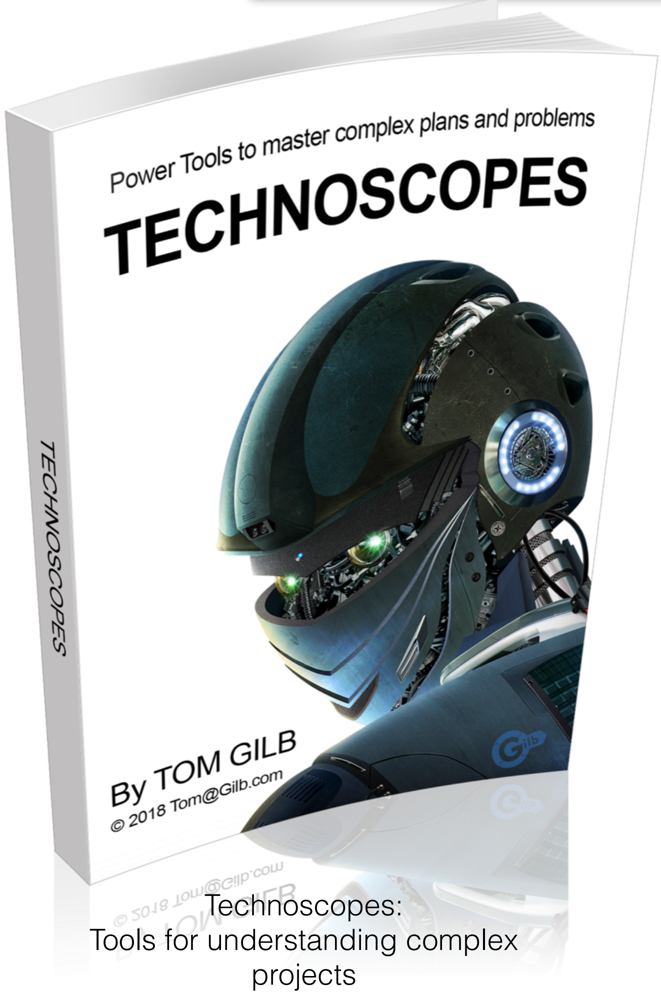
Technoscopes -
Meet the Challenge of Software Engineering Complexity



Technoscopes:
Tools for understanding complex projects
TECHNOSCOPES. https://www.gilb.com/offers/YYAMFQBH/ checkout (free).
For citation and purchase
Outside of this talk see
Sales LINK TO ALL BOOKLETS and Books
https://www.gilb.com/store?tag=books

By
Tom Gilb, Norway
tom@Gilb.com
www.Gilb.com
@ImTomGilb

April 1 2020, 18:30-20:00 approximately - presentation
BCS, London . 90 minutes presentation time.
http://concepts.gilb.com/dl968 (slides)

<—- Technoscopes.(free digital book about 100 tools to fight complexity)

Epub and pdf
Only offered to BCS SPA Talk participants
<————-(otherwise sold at gilb.com)

Slide version 010420.17:00

Wicked Problems Characteristics

(Some false assertions)

think: ‘Virus Planning’. Not ‘Chess’

1. There is no definitive formulation of a wicked problem.

2. Wicked problems have no stopping rule.

3. Solutions to wicked problems are not true-or-false, but good-or-bad.

4. There is no immediate and no ultimate test of a solution to a wicked problem.

5. Every solution to a wicked problem is a "one-shot operation"; because there is no opportunity to learn by trial-and-error, every attempt counts significantly.

6. Wicked problems do not have an enumerable (or an exhaustively describable) set of potential solutions, nor is there a well-described set of permissible operations that may be incorporated into the plan.

7. Every wicked problem is essentially unique.

8. Every wicked problem can be considered to be a symptom of another problem.

9. The existence of a discrepancy in representing a wicked problem can be explained in numerous ways. The choice of explanation determines the nature of the problem's resolution.

10. The planner (designer) has no right to be wrong.

“As We May Think”, In July 1945 formulated a vision
that inspired J.C.R. Licklider, Doug Engelbart and Ted Nelson
(Werner Kuntz and Horst Rittel, the designers of IBIS)
Dino Karabeg, OMS Group, Department of Informatics, University of Oslo

See slides after end of this talk here, after slide 67, with more detail, refuting each point here

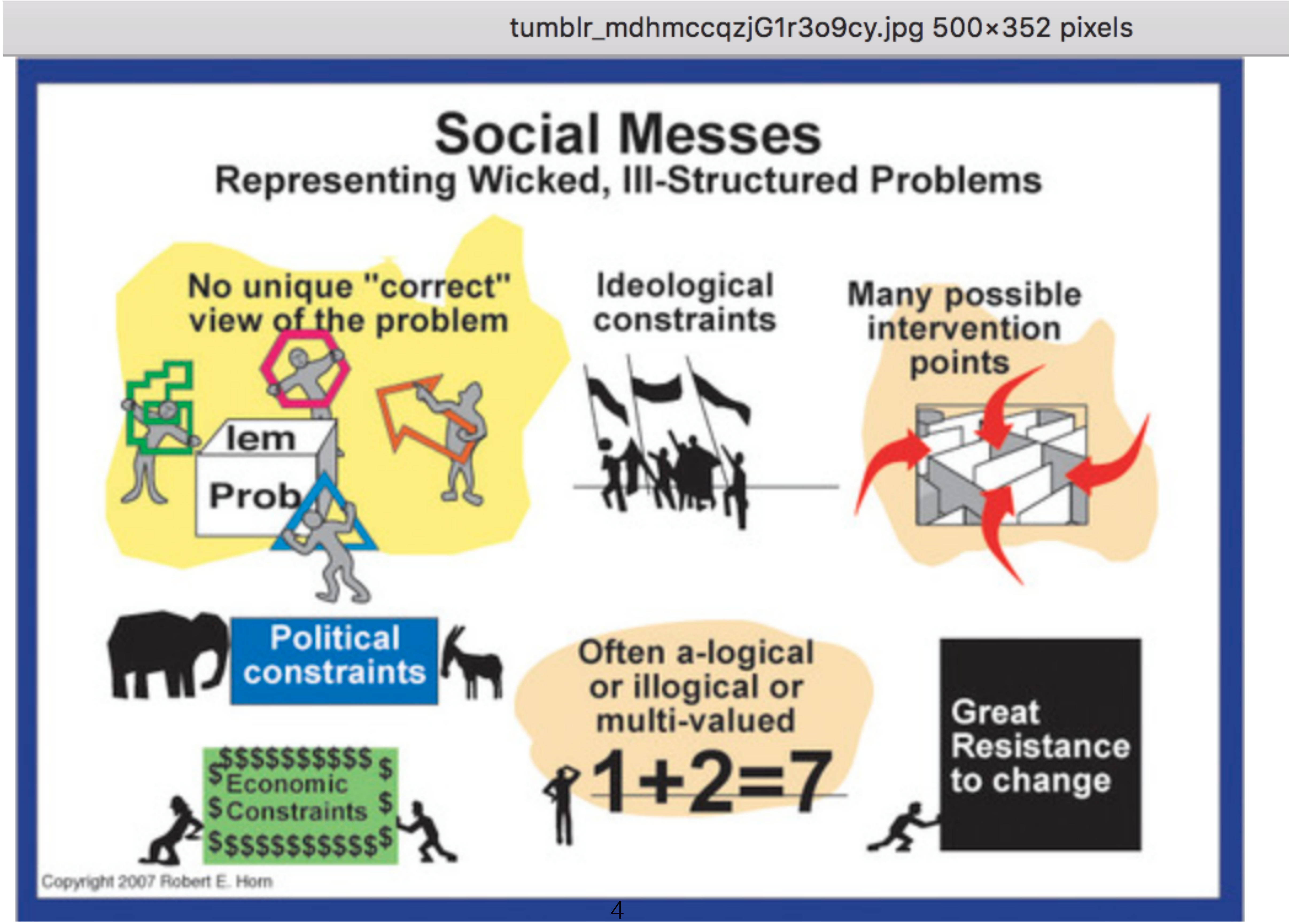
"A *tame* problem: is not so complex (think, 'stopping people smoking')

1. Has a ***well-defined and stable*** **problem statement;**
2. Has a ***definite stopping point*****, i.e., when the solution is reached;**
3. Has a ***solution that can be objectively evaluated*** **as right or wrong;**
4. Belongs to a ***similar class of problems*** **that are all solved in the same similar way;**
5. Has solutions that can be ***easily tried*** **and abandoned;**
6. Comes with a ***limited set of alternative solutions."***

Jeff Conklin

http://cognexusgroup.com/wp-content/uploads/2013/07/Using-Dialogue-Mapping-to-Address-Wicked-Problems-05-23-2013.pdf

Some characteristics of Complex Systems Like Covid-19 Pandemics



tumblr_mdhmccqzjG1r3o9cy.jpg 500×352 pixels

Social Messes

Representing Wicked, Ill-Structured Problems

No unique "correct" view of the problem

Ideological constraints

Many possible intervention points

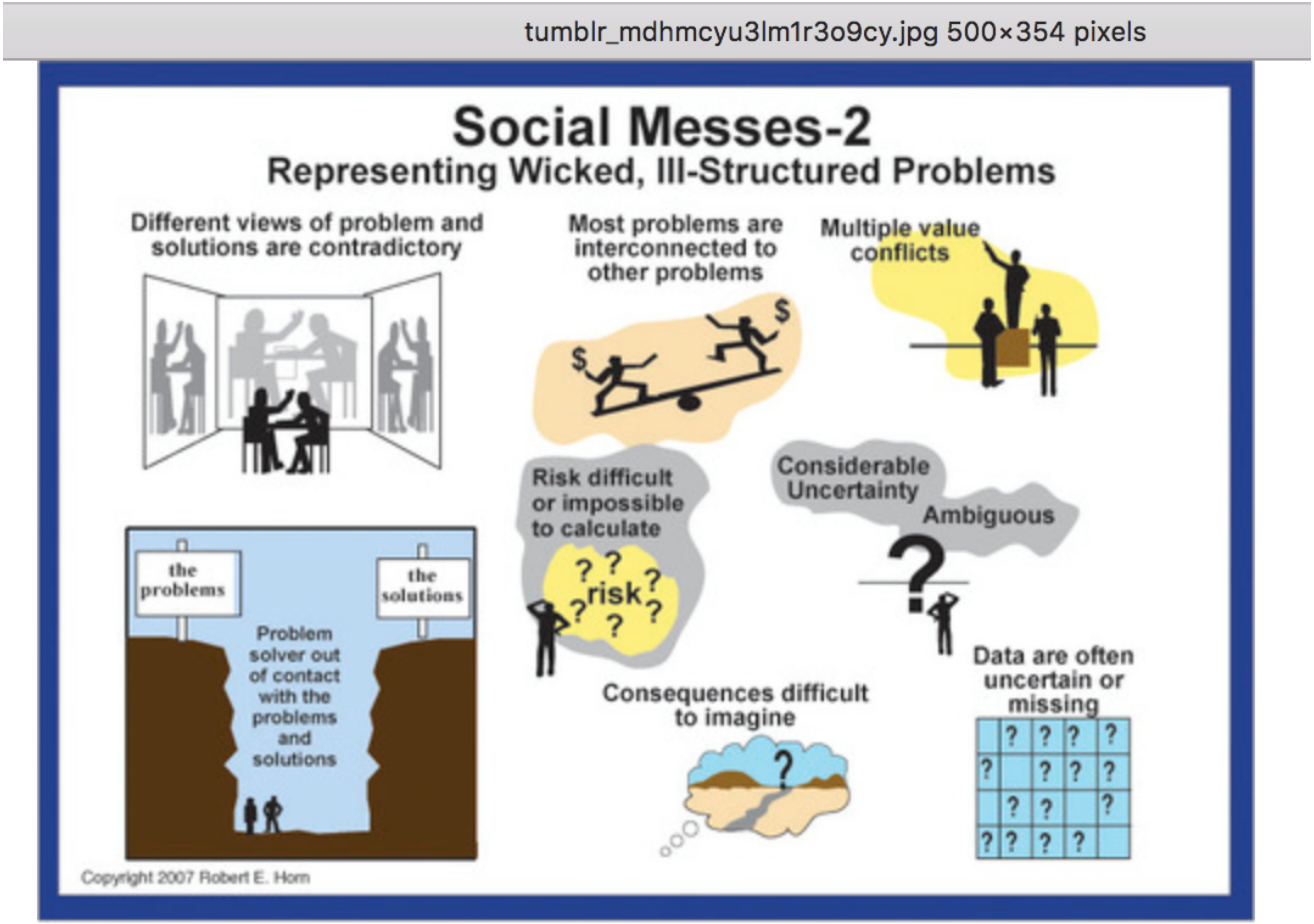
Political constraints

Often a-logical or illogical or multi-valued

1+2=7

Great Resistance to change

Some characteristics of Complex Systems Like Covid-19 Pandemics



tumblr_mdhmcyu3lm1r3o9cy.jpg 500×354 pixels

Social Messes-2

Representing Wicked, Ill-Structured Problems

Different views of problem and solutions are contradictory

Most problems are interconnected to other problems

Multiple value conflicts

Risk difficult or impossible to calculate

Considerable Uncertainty

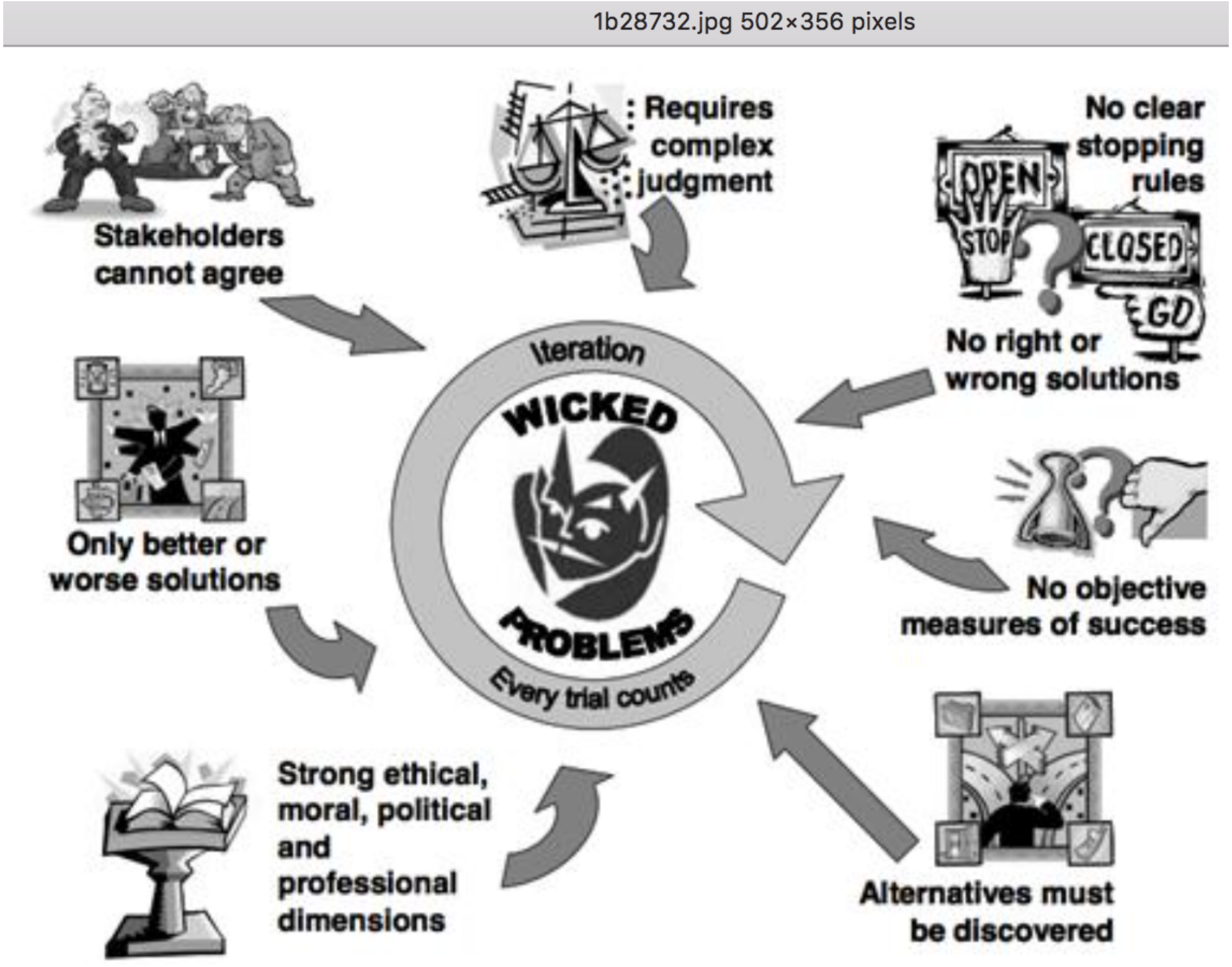
Ambiguous

Problem solver out of contact with the problems and solutions

Consequences difficult to imagine

Data are often uncertain or missing

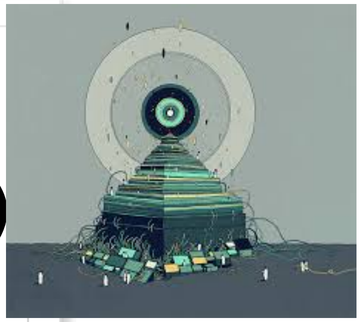
Some characteristics of Complex Systems Like Covid-19 Pandemics


1b28732.jpg 502×356 pixels
Stakeholders cannot agree
Requires complex judgment
No clear stopping rules
OPEN
STOP
CLOSED
GO
No right or wrong solutions
Iteration
WICKED PROBLEMS
Every trial counts
Only better or worse solutions
No objective measures of success
Strong ethical, moral, political and professional dimensions
Alternatives must be discovered

How 'Planguage**' Helps deal with 'Wicked' Software engineering Problems: Complexity

1. **by viewing the problem from a high 'stakeholder values' level**

   (avoiding all complex innards, be 'outside the black box'—>)



2. **by dealing with design and costs *incrementally***

   (so you do not get all complexity at once)

3. **by contracting for *results*,   not 'work'**

   (so complexity is transferred to expert contractors)

4. **by being lean: early, and preventive (like reduce late bugs)**

   (so complexity is reduced in total, *later*)

5. **by using scale-free methods: scale does not matter**

   (so scaling up *size*, does not drive complexity up)

ELSEVIER
TOM GILB
COMPETITIVE
ENGINEERING
A HANDBOOK FOR SYSTEMS, REQUIREMENTS AND SOFTWARE ENGINEERING MANAGEMENT USING PLANGUAGE

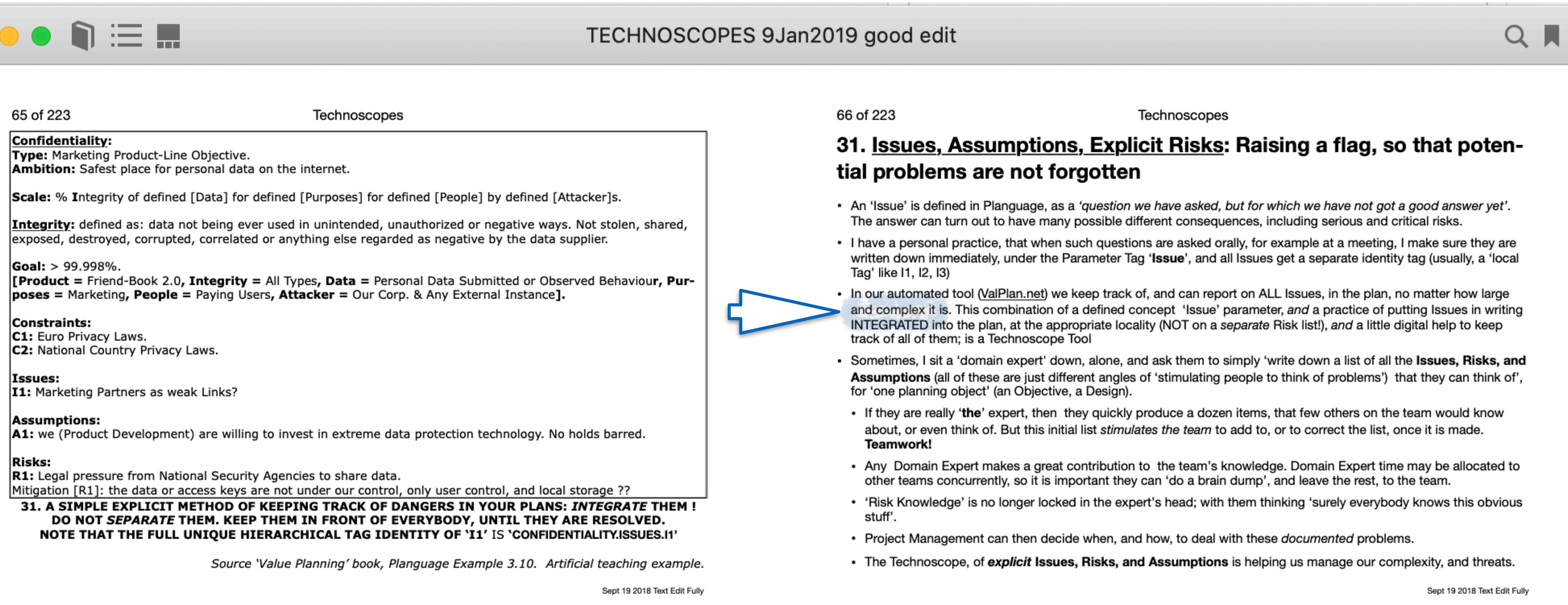
**Free Digital CE link Later slide

My talk is divided into these 5 main ideas

Technoscopes: book with 100 Complexity Tools

Example 31 of 100:

Keeping track of potential risks in complex systems



65 of 223 Technoscopes

Confidentiality:
Type: Marketing Product-Line Objective.
Ambition: Safest place for personal data on the internet.

Scale: % **I**ntegrity of defined [Data] for defined [Purposes] for defined [People] by defined [Attacker]s.

Integrity: defined as: data not being ever used in unintended, unauthorized or negative ways. Not stolen, shared, exposed, destroyed, corrupted, correlated or anything else regarded as negative by the data supplier.

Goal: > 99.998%.
[Product = Friend-Book 2.0**, Integrity =** All Types**, Data =** Personal Data Submitted or Observed Behaviou**r, Purposes =** Marketing**, People =** Paying Users**, Attacker =** Our Corp. & Any External Instance**].**

Constraints:
C1: Euro Privacy Laws.
C2: National Country Privacy Laws.

Issues:
I1: Marketing Partners as weak Links?

Assumptions:
A1: we (Product Development) are willing to invest in extreme data protection technology. No holds barred.

Risks:
R1: Legal pressure from National Security Agencies to share data.
Mitigation [R1]: the data or access keys are not under our control, only user control, and local storage ??

31. A SIMPLE EXPLICIT METHOD OF KEEPING TRACK OF DANGERS IN YOUR PLANS: *INTEGRATE* THEM ! DO NOT *SEPARATE* THEM. KEEP THEM IN FRONT OF EVERYBODY, UNTIL THEY ARE RESOLVED. NOTE THAT THE FULL UNIQUE HIERARCHICAL TAG IDENTITY OF 'I1' IS **'CONFIDENTIALITY.ISSUES.I1'**

Source 'Value Planning' book, Planguage Example 3.10. Artificial teaching example.

Sept 19 2018 Text Edit Fully

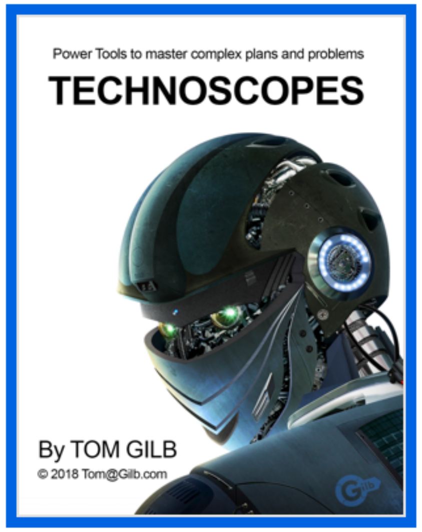
66 of 223 Technoscopes

31. Issues, Assumptions, Explicit Risks: Raising a flag, so that potential problems are not forgotten

- An 'Issue' is defined in Planguage, as a *'question we have asked, but for which we have not got a good answer yet'*. The answer can turn out to have many possible different consequences, including serious and critical risks.
- I have a personal practice, that when such questions are asked orally, for example at a meeting, I make sure they are written down immediately, under the Parameter Tag '**Issue**', and all Issues get a separate identity tag (usually, a 'local Tag' like I1, I2, I3)
- In our automated tool (ValPlan.net) we keep track of, and can report on ALL Issues, in the plan, no matter how large and complex it is. This combination of a defined concept 'Issue' parameter, *and* a practice of putting Issues in writing INTEGRATED into the plan, at the appropriate locality (NOT on a *separate* Risk list!), *and* a little digital help to keep track of all of them; is a Technoscope Tool
- Sometimes, I sit a 'domain expert' down, alone, and ask them to simply 'write down a list of all the **Issues, Risks, and Assumptions** (all of these are just different angles of 'stimulating people to think of problems') that they can think of', for 'one planning object' (an Objective, a Design).
  - If they are really '**the**' expert, then they quickly produce a dozen items, that few others on the team would know about, or even think of. But this initial list *stimulates the team* to add to, or to correct the list, once it is made. **Teamwork!**
  - Any Domain Expert makes a great contribution to the team's knowledge. Domain Expert time may be allocated to other teams concurrently, so it is important they can 'do a brain dump', and leave the rest, to the team.
  - 'Risk Knowledge' is no longer locked in the expert's head; with them thinking 'surely everybody knows this obvious stuff'.
  - Project Management can then decide when, and how, to deal with these *documented* problems.
  - The Technoscope, of ***explicit*** **Issues, Risks, and Assumptions** is helping us manage our complexity, and threats.

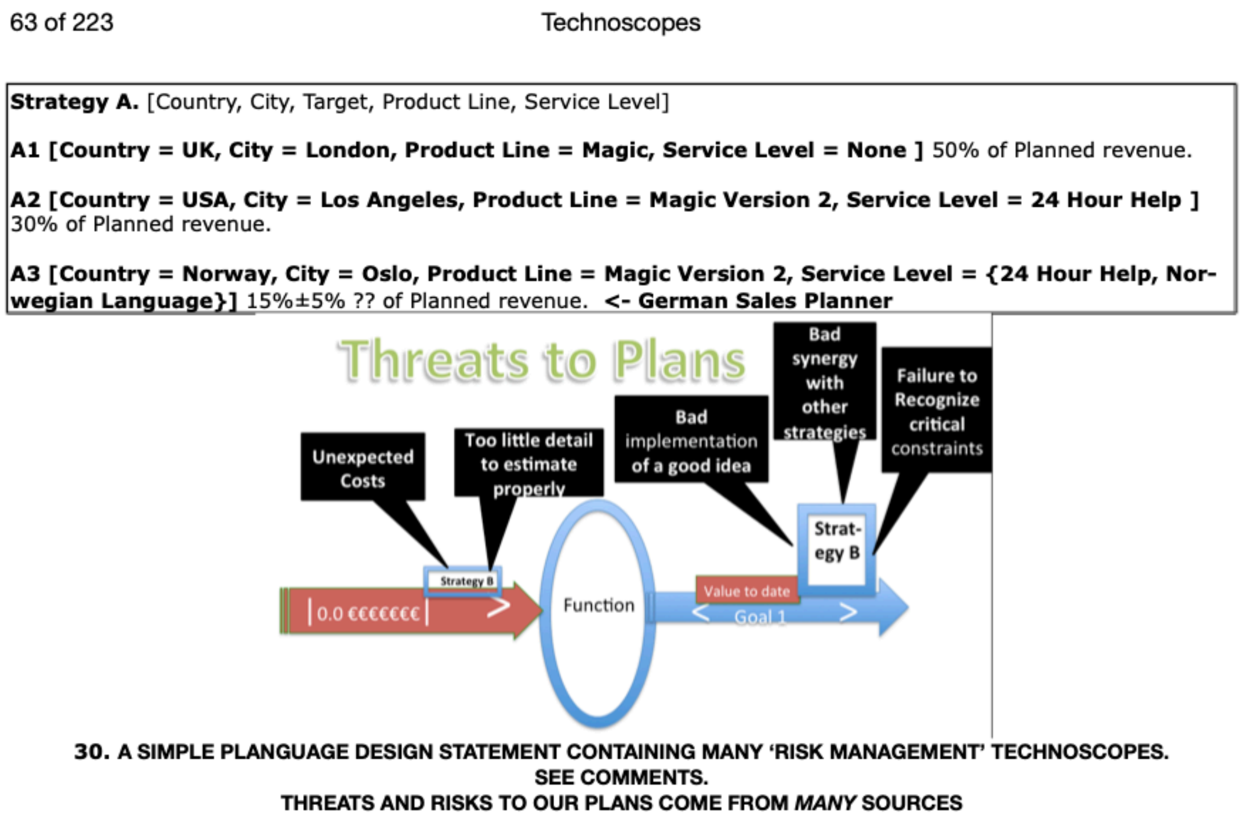
Sept 19 2018 Text Edit Fully

Technoscopes:
Tools for understanding complex projects
TECHNOSCOPES. https://www.gilb.com/offers/YYAMFQBH/ checkout (free).
For citation and purchase
Outside of this talk see
Sales LINK TO ALL BOOKLETS and Books
https://www.gilb.com/store?tag=books



Technoscopes: book with 100 Complexity Tools

Example 30 of 100: Keeping track of ‘risks complexity’



63 of 223 Technoscopes

Strategy A. [Country, City, Target, Product Line, Service Level]

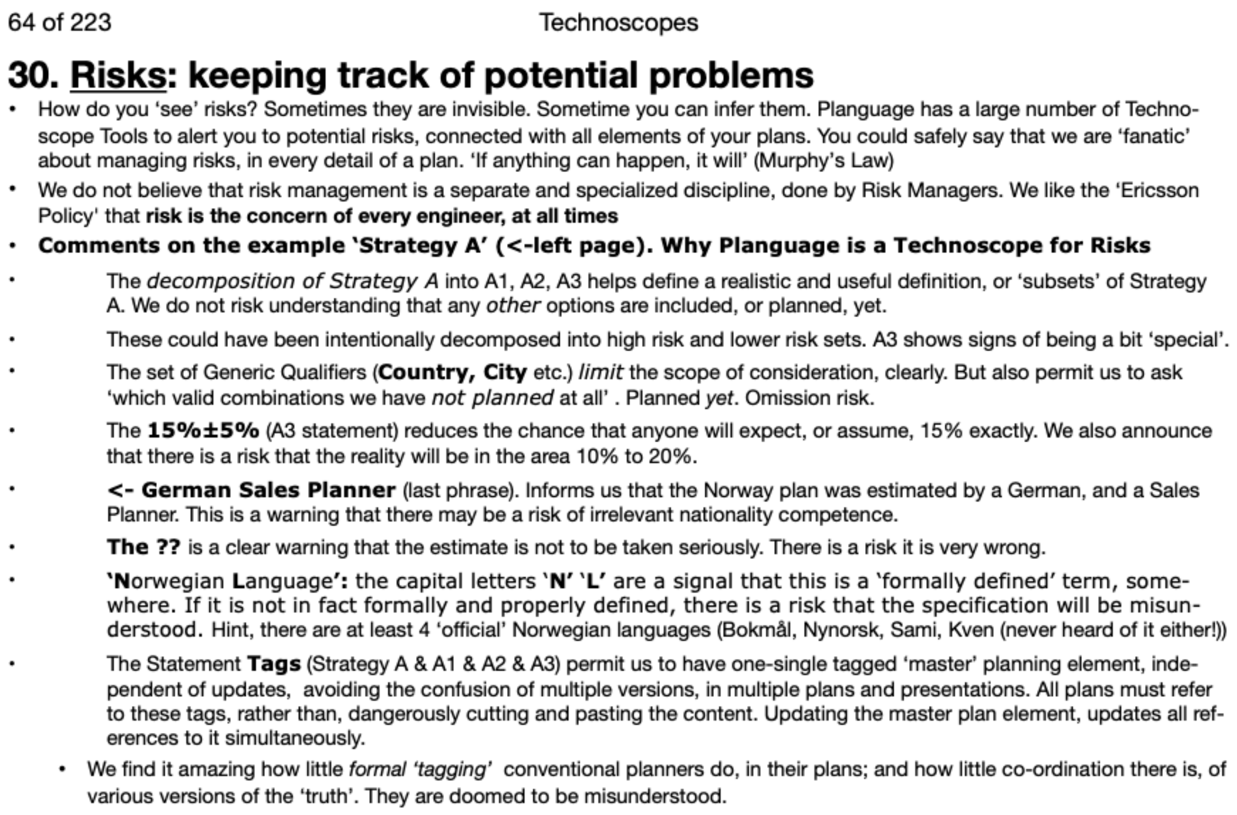
A1 [Country = UK, City = London, Product Line = Magic, Service Level = None] 50% of Planned revenue.

A2 [Country = USA, City = Los Angeles, Product Line = Magic Version 2, Service Level = 24 Hour Help] 30% of Planned revenue.

A3 [Country = Norway, City = Oslo, Product Line = Magic Version 2, Service Level = {24 Hour Help, Norwegian Language}] 15%±5% ?? of Planned revenue. **<- German Sales Planner**

Threats to Plans
Unexpected Costs
Too little detail to estimate properly
Bad implementation of a good idea
Bad synergy with other strategies
Failure to Recognize critical constraints
Strategy B
|0.0 €€€€€€€|
>
Function
Value to date
< Goal 1 >
Strat-egy B

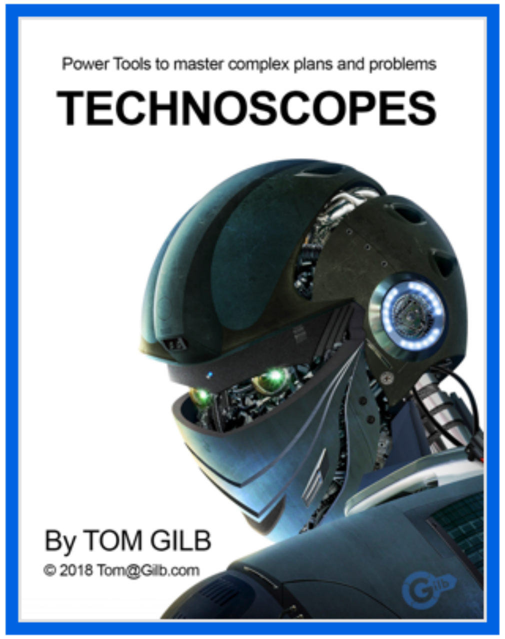
30. A SIMPLE PLANGUAGE DESIGN STATEMENT CONTAINING MANY ‘RISK MANAGEMENT’ TECHNOSCOPES.
SEE COMMENTS.
THREATS AND RISKS TO OUR PLANS COME FROM *MANY* SOURCES



64 of 223 Technoscopes

30. Risks: keeping track of potential problems

- How do you ‘see’ risks? Sometimes they are invisible. Sometime you can infer them. Planguage has a large number of Technoscope Tools to alert you to potential risks, connected with all elements of your plans. You could safely say that we are ‘fanatic’ about managing risks, in every detail of a plan. ‘If anything can happen, it will’ (Murphy’s Law)
- We do not believe that risk management is a separate and specialized discipline, done by Risk Managers. We like the ‘Ericsson Policy' that **risk is the concern of every engineer, at all times**
- **Comments on the example ‘Strategy A’ (<-left page). Why Planguage is a Technoscope for Risks**
- The *decomposition of Strategy A* into A1, A2, A3 helps define a realistic and useful definition, or ‘subsets’ of Strategy A. We do not risk understanding that any *other* options are included, or planned, yet.
- These could have been intentionally decomposed into high risk and lower risk sets. A3 shows signs of being a bit ‘special’.
- The set of Generic Qualifiers (**Country, City** etc.) *limit* the scope of consideration, clearly. But also permit us to ask ‘which valid combinations we have *not planned* at all’ . Planned *yet*. Omission risk.
- The **15%±5%** (A3 statement) reduces the chance that anyone will expect, or assume, 15% exactly. We also announce that there is a risk that the reality will be in the area 10% to 20%.
- **<- German Sales Planner** (last phrase). Informs us that the Norway plan was estimated by a German, and a Sales Planner. This is a warning that there may be a risk of irrelevant nationality competence.
- **The ??** is a clear warning that the estimate is not to be taken seriously. There is a risk it is very wrong.
- **‘N**orwegian **L**anguage**’:** the capital letters **‘N’ ‘L’** are a signal that this is a ‘formally defined’ term, somewhere. If it is not in fact formally and properly defined, there is a risk that the specification will be misunderstood. Hint, there are at least 4 ‘official’ Norwegian languages (Bokmål, Nynorsk, Sami, Kven (never heard of it either!))
- The Statement **Tags** (Strategy A & A1 & A2 & A3) permit us to have one-single tagged ‘master’ planning element, independent of updates, avoiding the confusion of multiple versions, in multiple plans and presentations. All plans must refer to these tags, rather than, dangerously cutting and pasting the content. Updating the master plan element, updates all references to it simultaneously.
- We find it amazing how little *formal ‘tagging’* conventional planners do, in their plans; and how little co-ordination there is, of various versions of the ‘truth’. They are doomed to be misunderstood.



Power Tools to master complex plans and problems

TECHNOSCOPES

By TOM GILB

© 2018 Tom@Gilb.com

Technoscopes:
Tools for understanding complex projects
TECHNOSCOPES. https://www.gilb.com/offers/YYAMFQBH/
checkout   (free).
For citation and purchase
Outside of this talk see
Sales LINK TO ALL BOOKLETS and Books
https://www.gilb.com/store?tag=books

Technoscopes Tool Area 1 of 5.
by viewing the problem from a high 'stakeholder values' level
(avoiding all complex innards)
Black box analysis

- How does this, high level view, help us deal with complexity?

1. The stakeholder and their values (needs, requirements) are the essential focus of all projects

2. The underlying complex details (the design, the actual system, the code) do not really matter, as long as the stakeholders are getting their needs met.

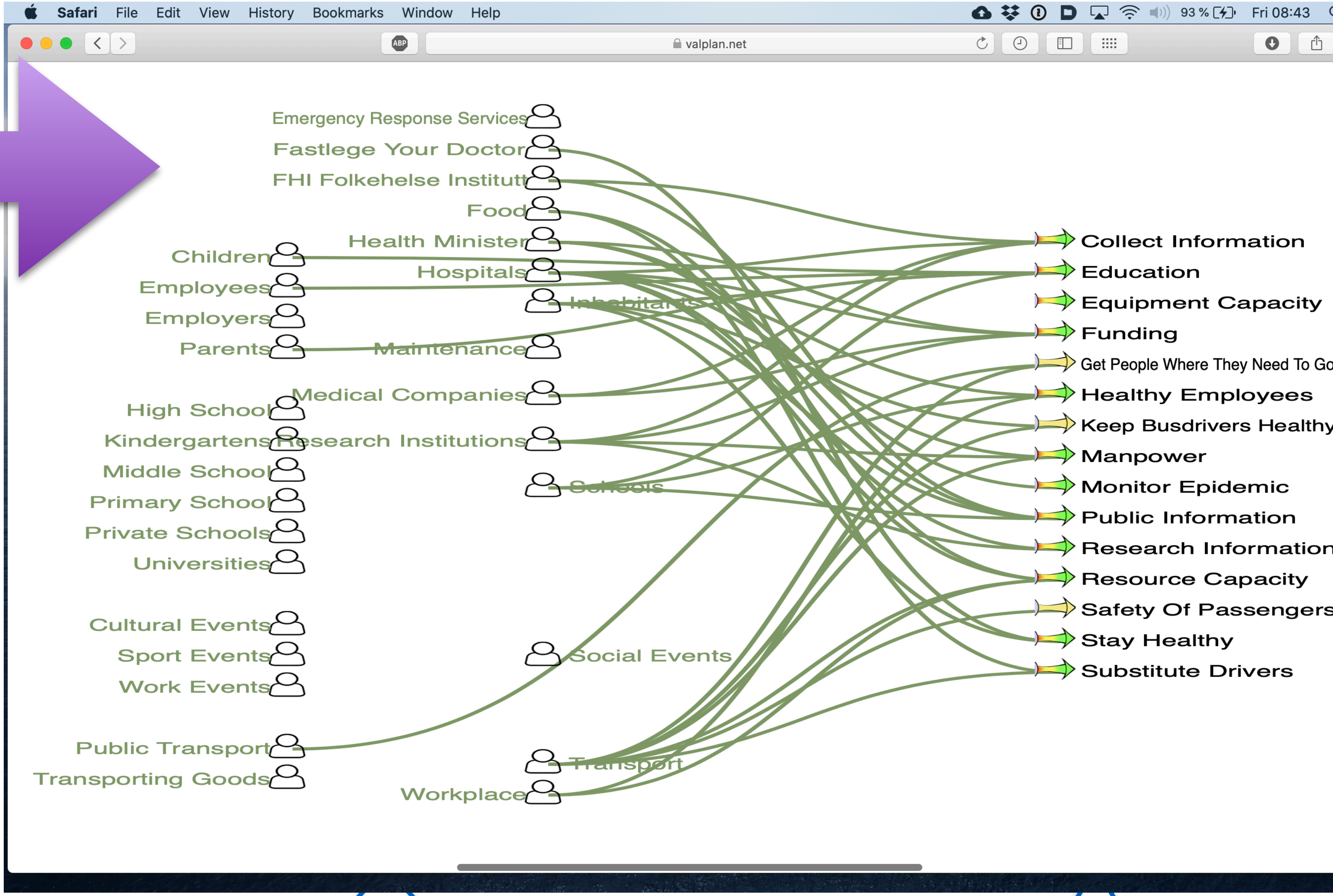
And meeting needs can be measured directly.

You do not need to go into the 'black box'.

The 'inside of the black box is extremely difficult to analyze *directly*:

It is better to just <u>measure the results, values</u>, qualities,

as delivered to the stakeholder.

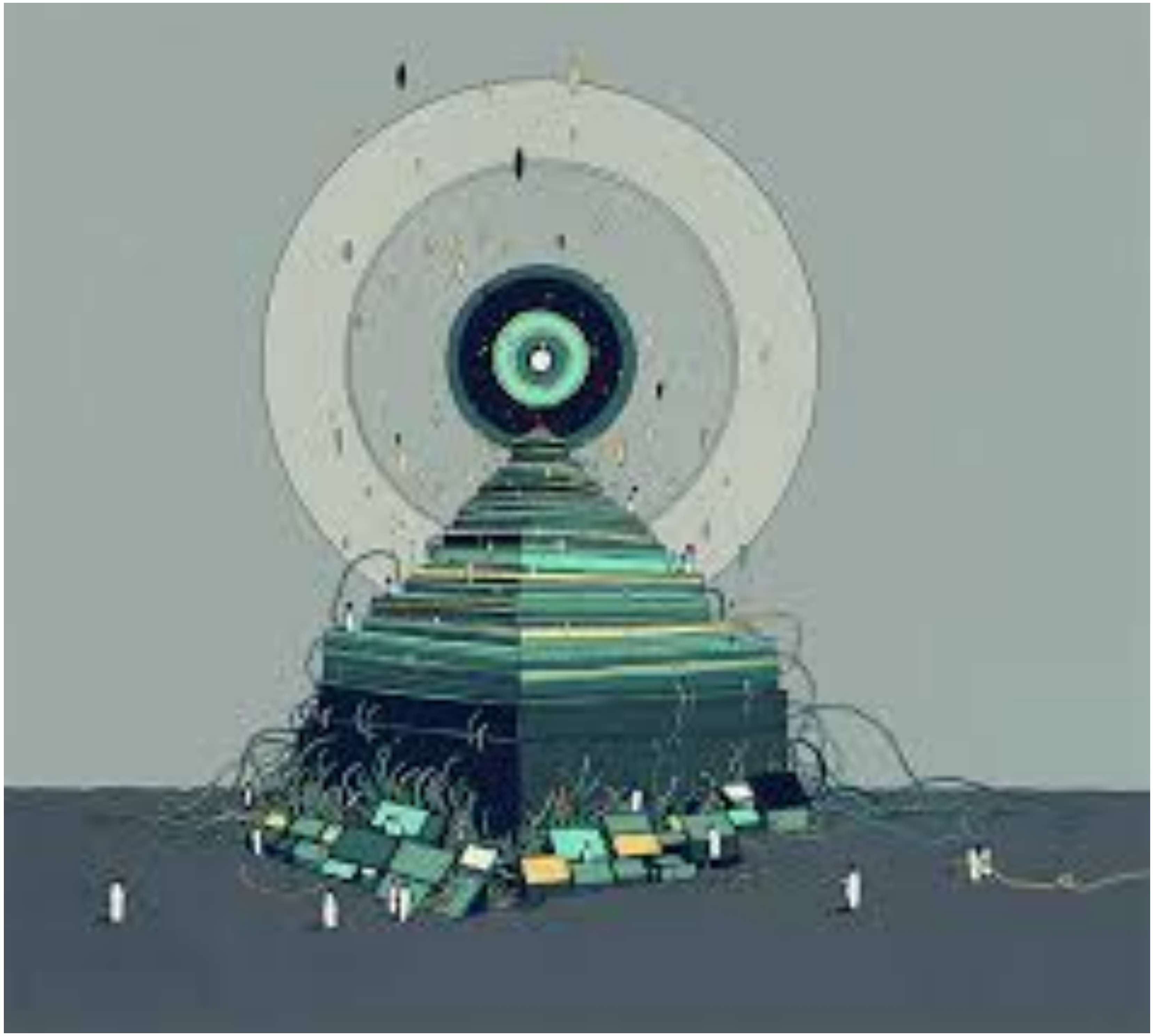


Example from OSWA Oslo,
Architecture Workshop
March 2020

Stakeholders

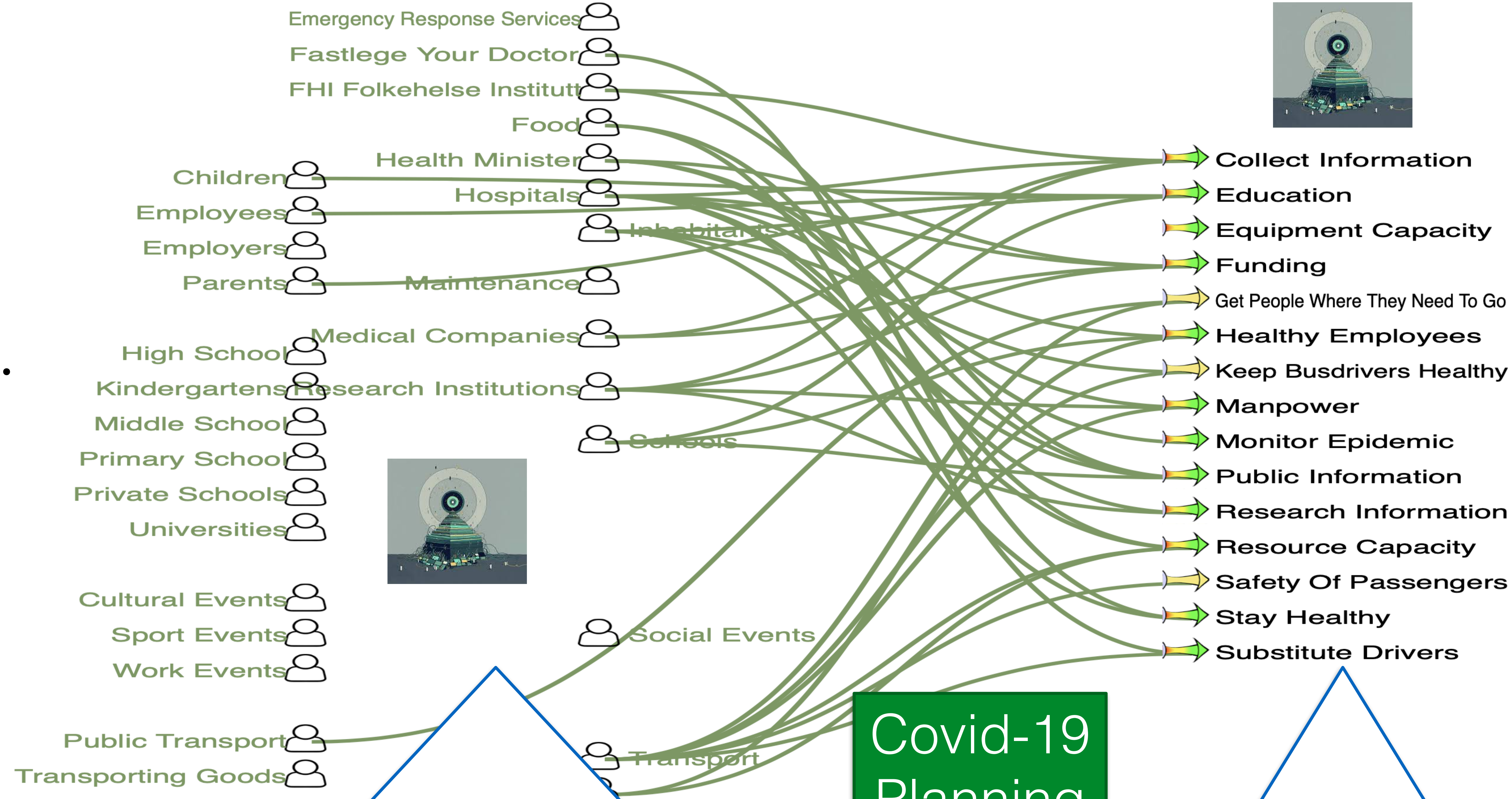
Values

Black Box analysis of Complex (AI) systems



https://www.nature.com/news/can-we-open-the-black-box-of-ai-1.20731

Technoscopes Tool Area 1 of 5.
Notice the beginning of complexity even here:
Any stakeholder can have many values.
Any value can have many stakeholders
Any identified stakeholder or value can have yet-unknown connection to other values or stakeholders.
IN THIS CASE WE HAVE A DIGITAL TOOL TO KEEP CONTINUOUS TRACK OF ALL KNOWN RELATIONSHIPS
So you can ask questions like which values does the University stakeholder have, which we have forgotten right now?



Stakeholders

Covid-19 Planning Exercise

Values

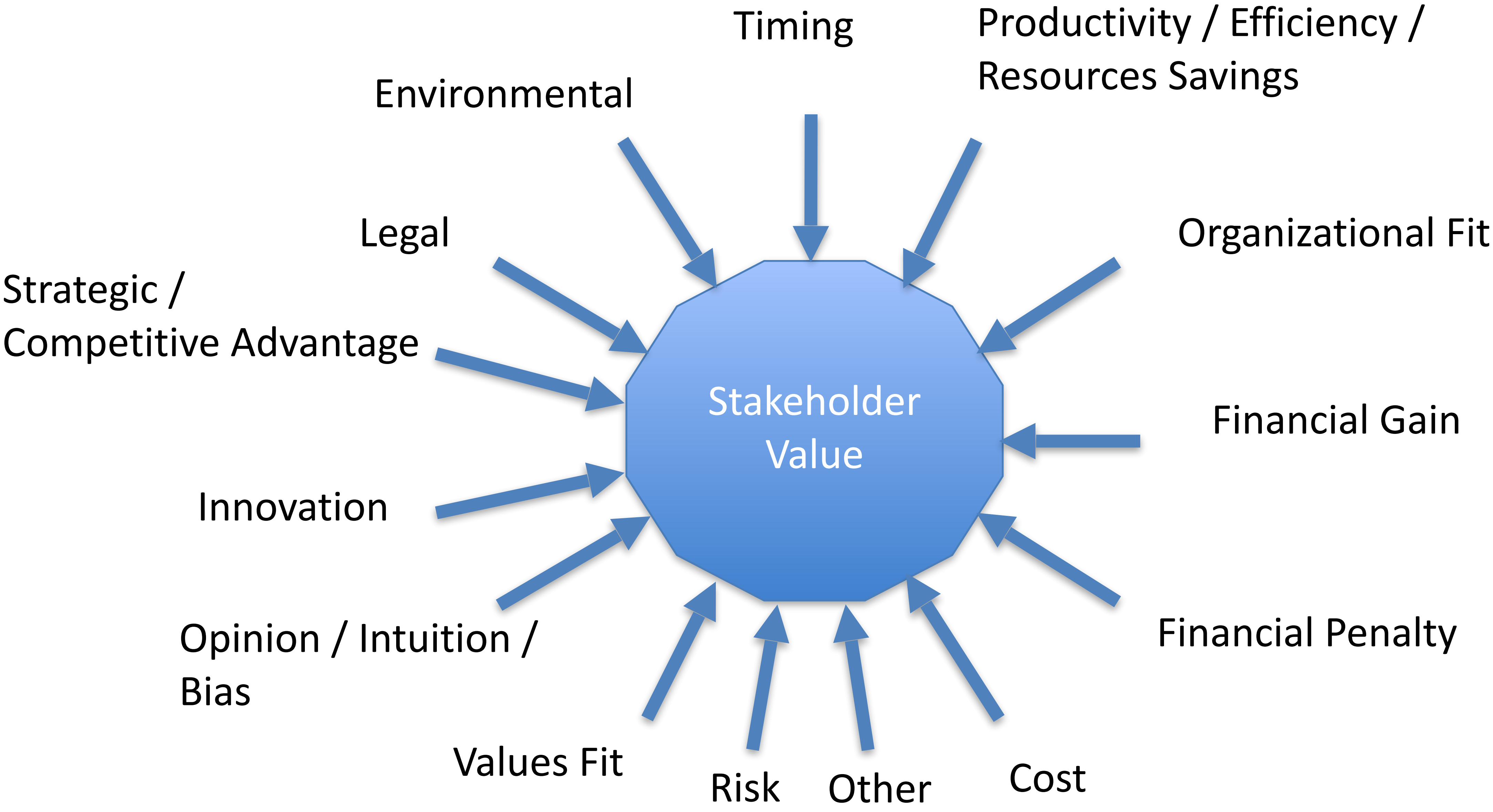
Dimensions of Stakeholder Value

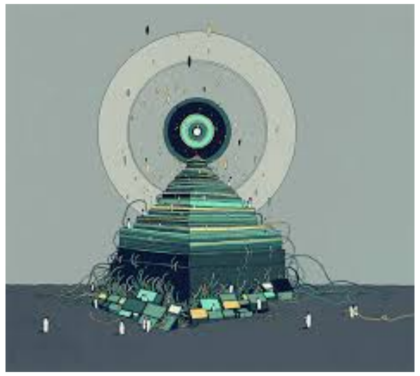
One single stakeholder type, has many critical values, and *variable* values,

And individual instances of the stakeholder type (MIT, UCLA, LSE), have different sets of values and different levels needed

THIS IS VERY COMPLEX, BUT WE CAN KEEP TRACK OF THE HIGHEST PRIORITY CASES BY USING A DIGITAL TOOL

And by using quantified definitions of each critical value of a stakeholder





Any technical solution or strategy can be evaluated against several of these dimensions of stakeholder value at once.

This is a systematic view of complexity, and needs to be tracked in a digital model.

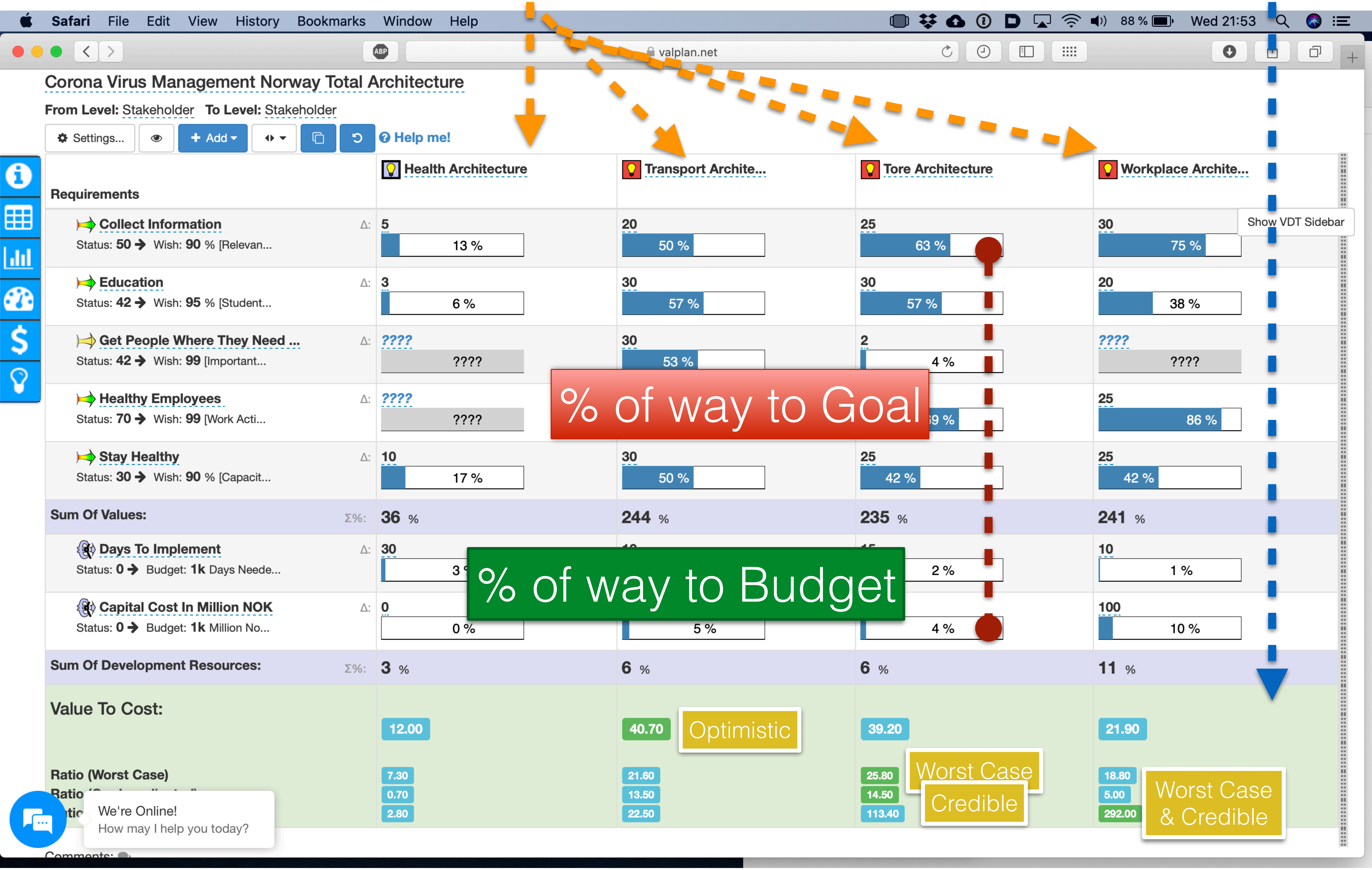
Complexity = X Objectives x Y Resources x Z Solutions = Benefit/Cost Ratios = (more, risks, evidence, …)

Basic Model of an Impact Estimation Table

	Solution 1	Solution 2	Solution n	Total Impacts
Objective	Impact on Objective	Impact on Objective	Impact on Objective	Total Impact on Objective
Resources	Impact on Budget	Impact on Budget	Impact on Budget	Total Impact on Budget
Benefits-to-Cost Ratio	Ratio	Ratio	Ratio	

Courtesy Rolf Goetz

Corona Virus Planning: a 4-team class 2020,
The IE Table shows 4 Solutions x 6 Attributes + 4 Val/£



Corona Virus Management Norway Total Architecture

From Level: Stakeholder **To Level:** Stakeholder

Requirements	Health Architecture	Transport Archite...	Tore Architecture	Workplace Archite...
Collect Information Status: 50 → Wish: 90 % [Relevan... Δ:	5 13 %	20 50 %	25 63 %	30 75 %
Education Status: 42 → Wish: 95 % [Student... Δ:	3 6 %	30 57 %	30 57 %	20 38 %
Get People Where They Need ... Status: 42 → Wish: 99 [Important... Δ:	???? ????	30 53 %	2 4 %	???? ????
Healthy Employees Status: 70 → Wish: 99 [Work Acti... Δ:	???? ????	[illegible]	[illegible] 9 %	25 86 %
Stay Healthy Status: 30 → Wish: 90 % [Capacit... Δ:	10 17 %	30 50 %	25 42 %	25 42 %
Sum Of Values: Σ%:	36 %	244 %	235 %	241 %
Days To Implement Status: 0 → Budget: 1k Days Neede... Δ:	30 3 %	[illegible]	[illegible] 2 %	10 1 %
Capital Cost In Million NOK Status: 0 → Budget: 1k Million No... Δ:	0 0 %	[illegible] 5 %	[illegible] 4 %	100 10 %
Sum Of Development Resources: Σ%:	3 %	6 %	6 %	11 %
Value To Cost:	12.00	40.70	39.20	21.90
Ratio (Worst Case)	7.30	21.60	25.80	18.80
Ratio [illegible]	0.70	13.50	14.50	5.00
[illegible]	2.80	22.50	113.40	292.00

% of way to Goal

% of way to Budget

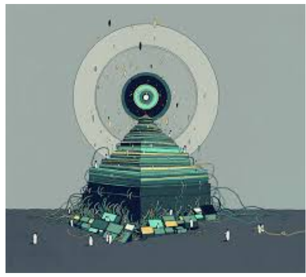
Optimistic

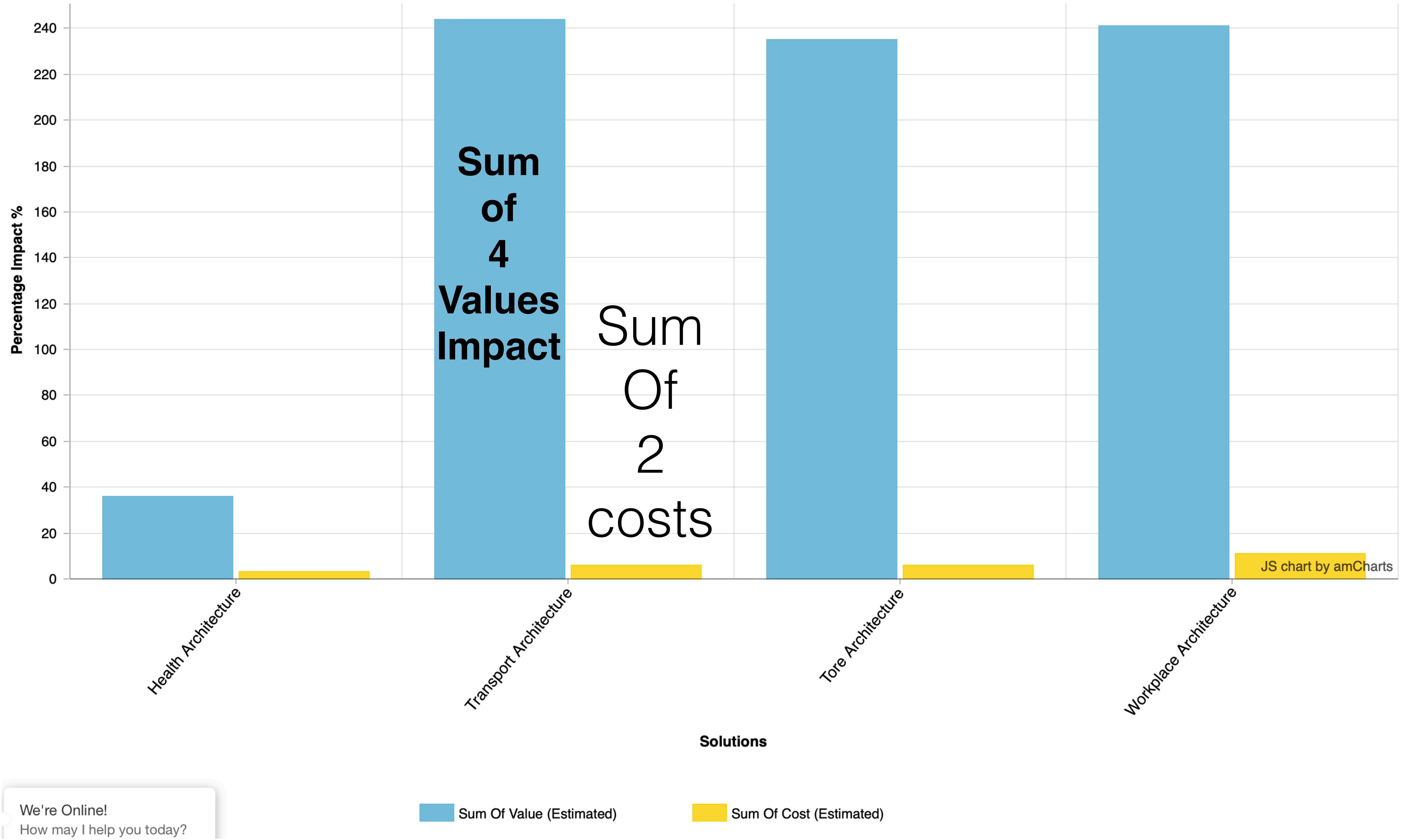
Worst Case

Credible

Worst Case & Credible

The ‘Complex’ Table, simplified
Which solution is best ‘values for costs’?





Sum of 4 Values Impact

Sum Of 2 costs

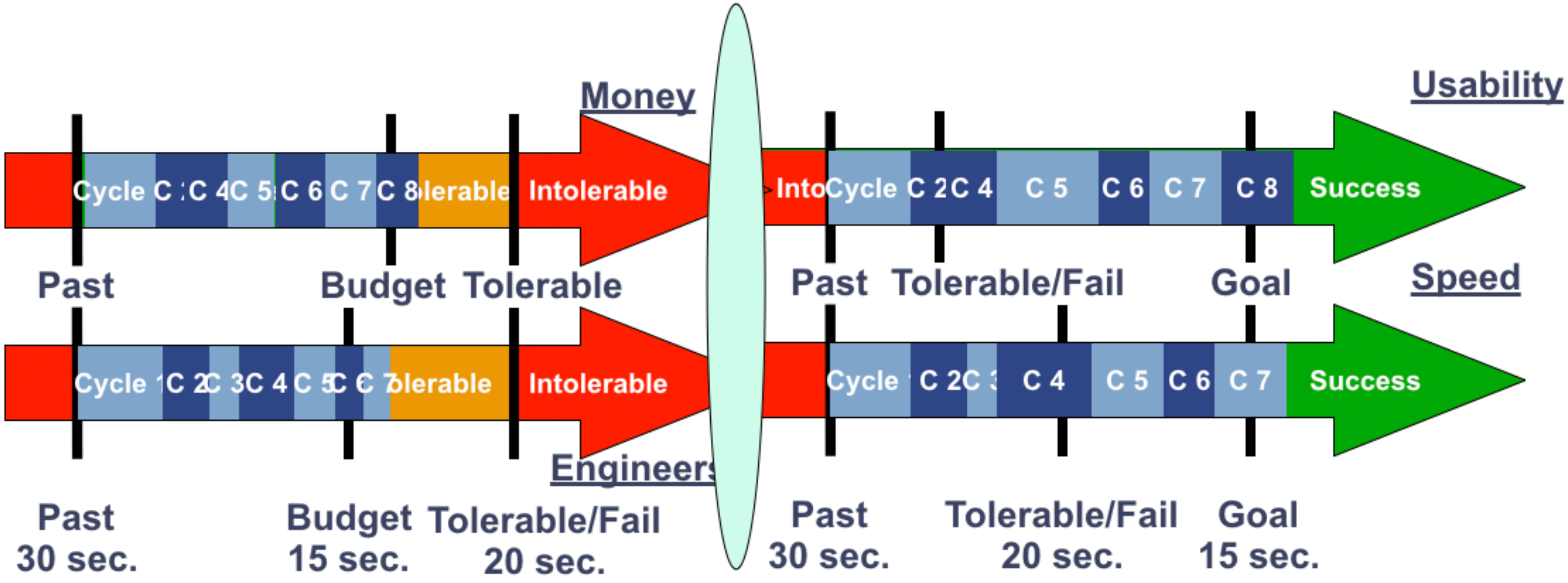
Complexity Technoscopes, Tool Area 2 of 5.

By dealing with design and costs incrementally

(= Agile, Evo)

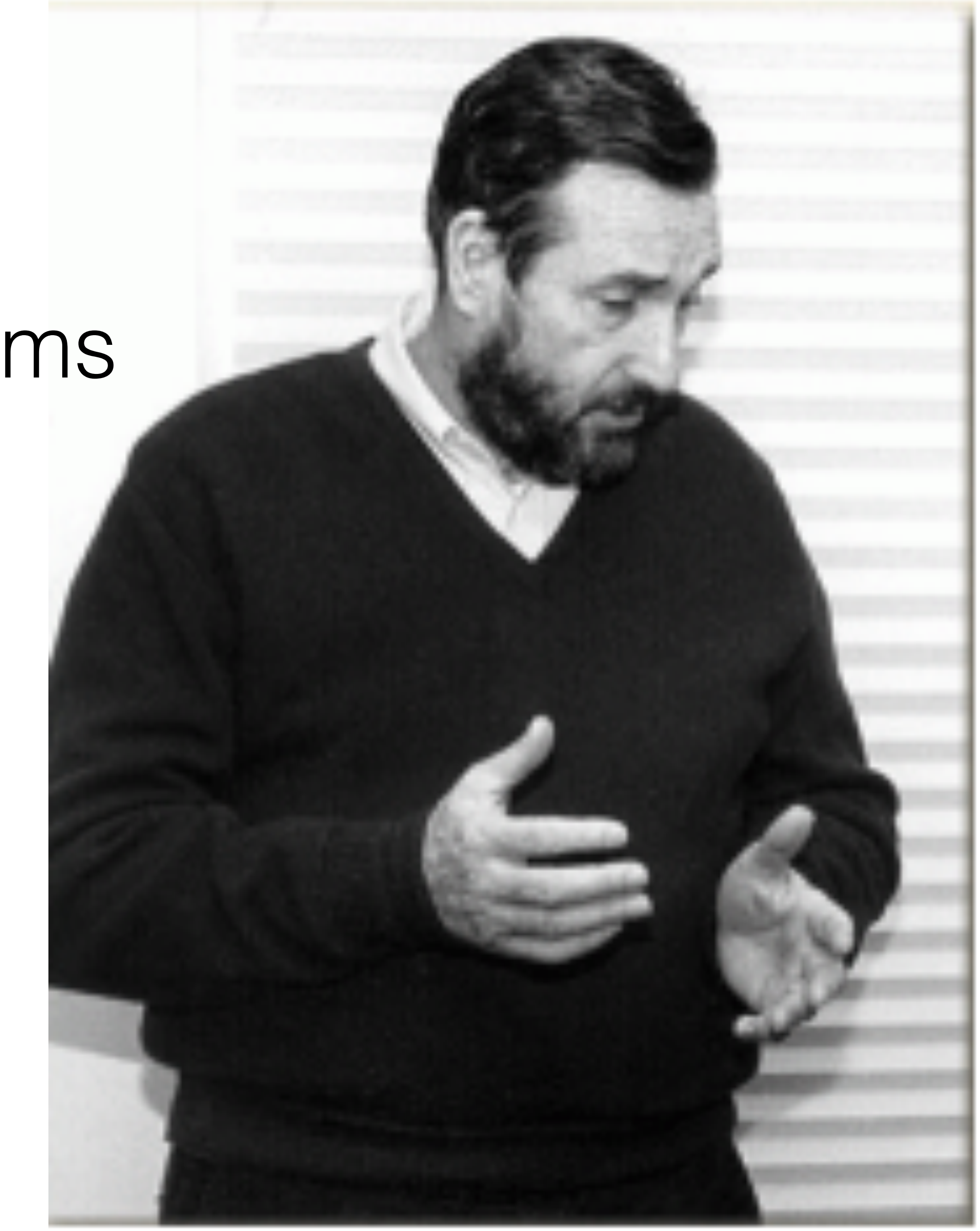
Clearer cause and effect.
Easier to correct early.

This helps deal with complexity,
because we only need to consider
one small increment of the system, at a a time.
Maybe 1/50 or 1/200 of it.


Money
Cycle C :C 4C 5:C 6 C 7 C 8lerable Intolerable
Past
Budget Tolerable
Cycle 1C 2C 3C 4 C 5C C 7lerable Intolerable
Engineers
Past
30 sec.
Budget
15 sec.
Tolerable/Fail
20 sec.
Usability
Into Cycle C 2C 4 C 5 C 6 C 7 C 8 Success
Past Tolerable/Fail Goal
Speed
Cycle C 2C 3 C 4 C 5 C 6 C 7 Success
Past
30 sec.
Tolerable/Fail
20 sec.
Goal
15 sec.

‘Cleanroom’
An advanced software development process
(‘perfect’ complex project management)

Dr. Harlan Mills,
IBM Federal Systems
Division



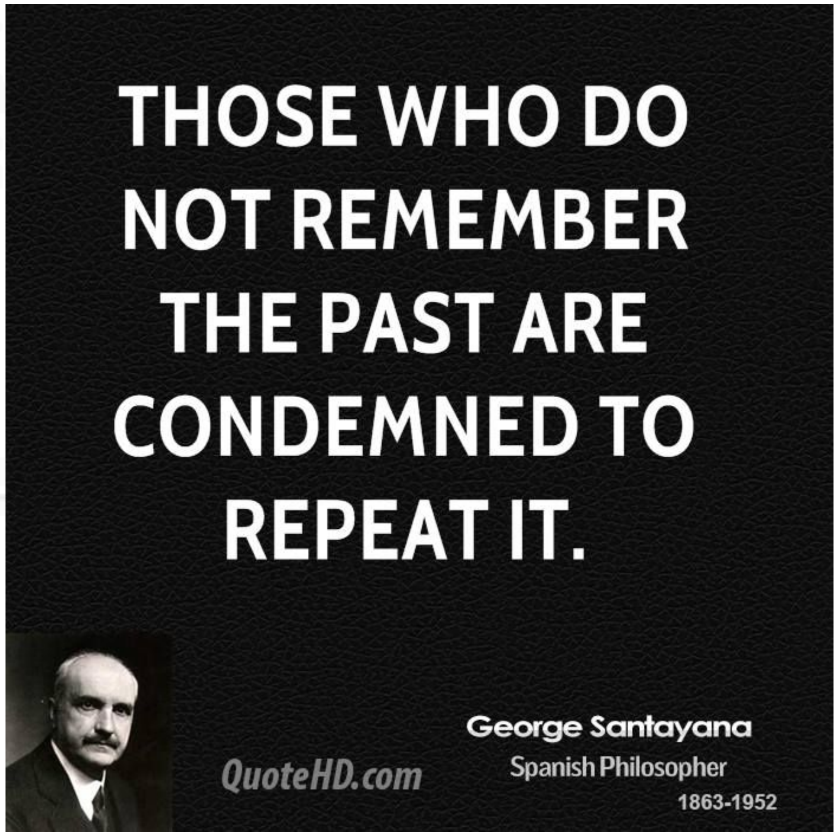
A real ‘agile’
software engineer
pioneer

https://trace.tennessee.edu/utk_harlan/?utm_source=trace.tennessee.edu/utk_harlan/5&utm_medium=PDF&utm_campaign=PDFCoverPages
Collected works of Harlan Mills

Reprinted from

IBM Systems Journal

Volume Ninteen | Number Four | 1980



THOSE WHO DO NOT REMEMBER THE PAST ARE CONDEMNED TO REPEAT IT.

QuoteHD.com

George Santayana
Spanish Philosopher
1863-1952

The management of software engineering Part I: Principles of software engineering

by H. D. Mills

http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan

"The first guarantee of quality":

Designing Qualities into a system
Is much *simpler*
than trying to get qualities, by *testing* them in**

Harlan Mills

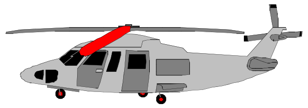
- **"The first guarantee of quality in design is in well-informed, well-educated, and well-motivated designers.**
- **Quality must be built into designs, and cannot be inspected in or tested in.**
- **Nevertheless, any prudent development process verifies quality through inspection and testing.**
- **Inspection by peers in design, by users or surrogates, by other financial specialists concerned with cost, reliability, or maintainability not only increases confidence in the design at hand, but also provides designers with valuable lessons and insights to be applied to future designs.**
- **The very fact that designs face inspections motivates even the most conscientious designers to greater care, deeper simplicities, and more precision in their work."**

http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan

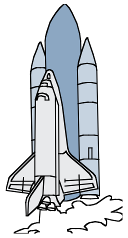
** testing qualities into a system, is impossibly **complex**, and takes infinite time.

In the 'Cleanroom Method', developed by IBM's Harlan Mills (IBM SJ No. 4/1980) they reported:

- **"Software Engineering began to emerge in FSD" (IBM Federal Systems Division, from 1996 a part of Lockheed Martin Marietta) "some ten years ago [Ed. about 1970] in a continuing evolution that is still underway:**
- **Ten years ago general management expected the worst from software projects – cost overruns, late deliveries, unreliable and incomplete software**
- **Today [Ed. 1980!], management has learned to expect on-time, within budget, deliveries of high-quality software. A Navy helicopter ship system, called LAMPS, provides a recent example. LAMPS software was a four-year project of over 200 person-years of effort, developing over three million, and integrating over seven million words of program and data for eight different processors distributed between a helicopter and a ship in 45 incremental deliveries [Ed. Note 2%!]s. Every one of those deliveries was on time and under budget**



- **A more extended example can be found in the NASA space program,**



- **- Where in the past ten years, FSD has managed some 7,000 person-years of software development, developing and integrating over a hundred million bytes of program and data for ground and space processors in over a dozen projects.**
- **- There were few late or overrun deliveries in that decade, and none at all in the past four years."**

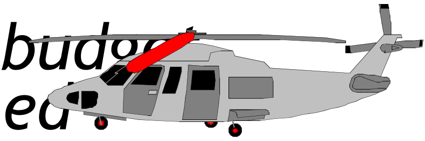
In the Cleanroom Method,
developed by IBM’s Harlan Mills (1980) they reported:

- *“Software Engineering began to emerge in FSD” (IBM Federal Systems Division, f out 1*
- *T ects - cost overruns, late deliveries, unreliable and incomplete software*

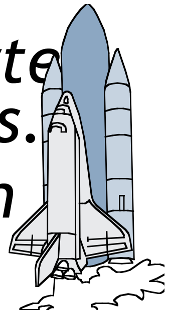
in 45 incremental deliveries

Agile!

- *Today [Ed. 1980!], management has learned to expect on-time, within budg deliveries of high-quality software. A Navy helicopter ship system, called LAMPS, provides a recent example. LAMPS software was a four-year project of over 200 person-years of effort, developing over three million, and integrating over s processors distrib veries [Ed. Note 2 dget*



- *A mor ,*
- *- Whe -years of softw million byte of pro en projects.*
- *- Ther ne at all in the pa*



were few late or overrun deliveries in that decade, and none at all in the past four years

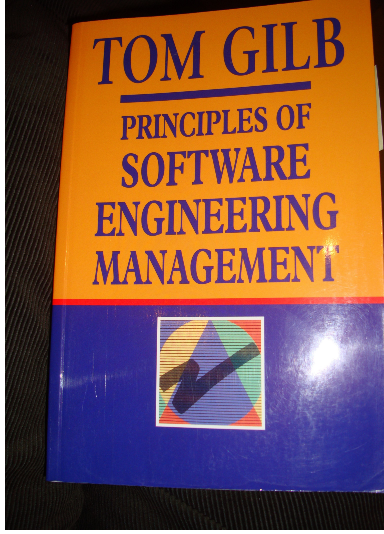
wow! *normal* agile fails 19% (Scrum) to 40%: Jeff Sutherland

Mills on Design-to-Cost (call it ‘agile’, incremental, design)

- “To meet cost/schedule commitments based on imperfect estimation techniques, a software engineering manager must adopt a **manage-and-design-to-cost/schedule process**.
- That process requires a **continuous and relentless rectification** of design objectives with the cost/schedule needed to achieve those objectives.”
- in IBM sj 4 80 p.420

See Quinnan’s flow chart “Design to Cost” below For process detail

Robert E. Quinnan (Cleanroom Architect): IBM FSD Cleanroom *Dynamic Design to Cost*



Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management. . . yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance. Design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method [illustrated in PoSEM book by Figure 7.10] consists of developing a design, estimating its cost, and ensuring that the design is cost-effective.' (p. 473)

He goes on to describe a design iteration process trying to meet cost targets by either redesign or by sacrificing 'planned capability.' When a satisfactory design at cost target is achieved for a single increment, the 'development of each increment can proceed concurrently with the program design of the others.'

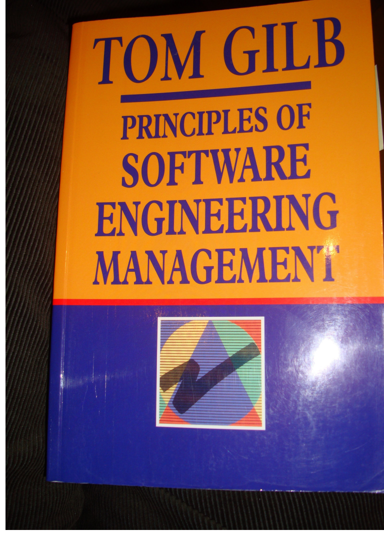
'Design is an iterative process in which each design level is a refinement of the previous level.' (p. 474)

It is clear from this that they avoid the big bang cost estimation approach. Not only do they iterate in seeking the appropriate balance between cost and design for a single increment, but they iterate through a series of increments, thus reducing the complexity of the task, and increasing the probability of learning from experience, won as each increment develops, and as the true cost of the increment becomes a fact.

'When the development and test of an increment are complete, an estimate to complete the remaining increments is computed.' (p. 474)

Source: Robert E. Quinnan, 'Software Engineering Management Practices', IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 466~77
This text is cut from Gilb: The Principles of Software Engineering Management, 1988

Initial design and cost estimates are incrementally reviewed and improved



Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management. . . yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance. Design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method [illustrated in this book by Figure 7.10] consists of developing a design, estimating its cost, and ensuring that the design is cost-effective.' (p. 473)

He goes on to des … or by sacrificing 'planned capability.' When a satisf … t of each increment can proceed concurrently wit

'Design is an iterative pro

It is clear from thi … e in seeking the appropriate balance betw … increments, thus reducing the complexity … each increment develops, and as the true

of developing a design, estimating its cost, and ensuring that the design is cost-effective

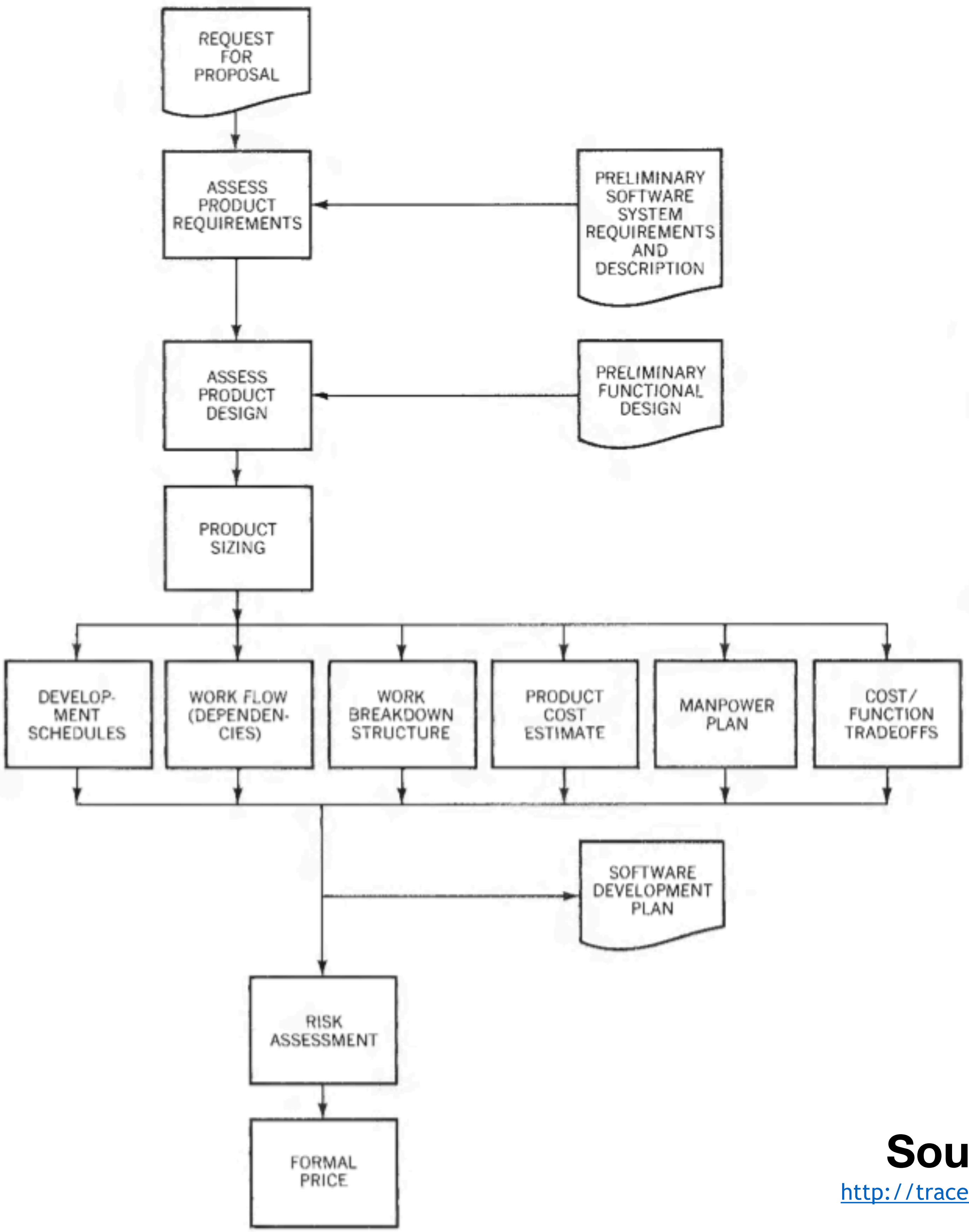
'When the development and test of an increment are complete, an estimate to complete the remaining increments is computed.' (p. 474)

Source: Robert E. Quinnan, 'Software Engineering Management Practices', IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 466~77

This text is cut from Gilb: The Principles of Software Engineering Management, 1988

Cleanroom Planning and Estimating
For making fixed price bid

Figure 2 Planning and estimating



“developing a design, estimating its cost”

<— making sure it is cost effective (static)

Source: Quinnan, IBM SJ, page 472

http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan

Cleanroom Cost Management Process

Figure 1 Cost management process

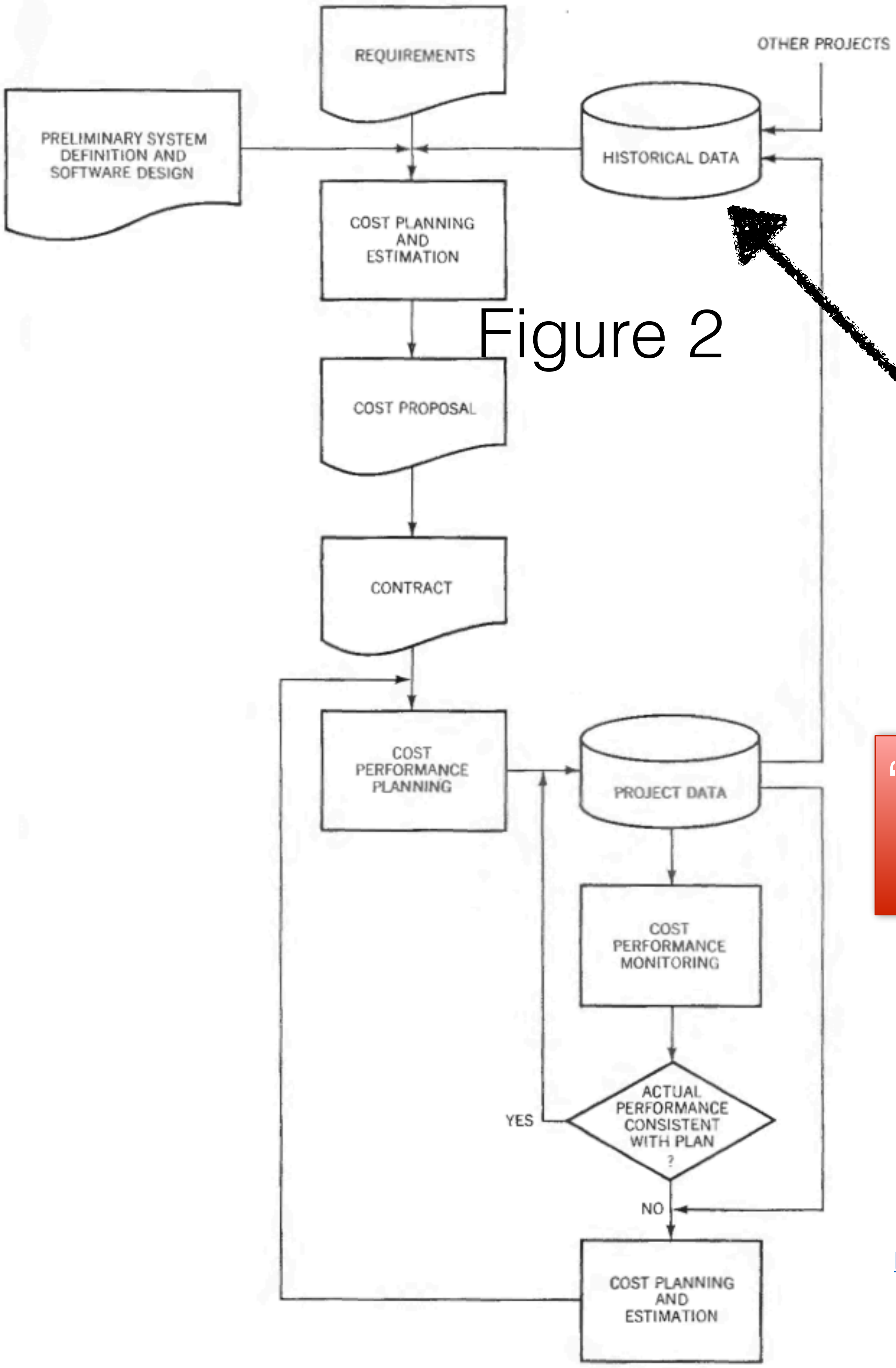
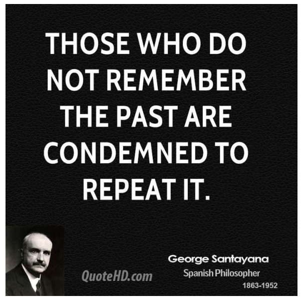

REQUIREMENTS
OTHER PROJECTS
PRELIMINARY SYSTEM DEFINITION AND SOFTWARE DESIGN
HISTORICAL DATA
COST PLANNING AND ESTIMATION
COST PROPOSAL
CONTRACT
COST PERFORMANCE PLANNING
PROJECT DATA
COST PERFORMANCE MONITORING
YES
ACTUAL PERFORMANCE CONSISTENT WITH PLAN ?
NO
COST PLANNING AND ESTIMATION

Figure 2

IBM FSD had a very advanced detailed collection of historical data from previous projects. Published in IBM SJ, Walston and Felix About 20 pages of data per project were collected



THOSE WHO DO NOT REMEMBER THE PAST ARE CONDEMNED TO REPEAT IT.

George Santayana
Spanish Philosopher
1863-1952

QuoteHD.com

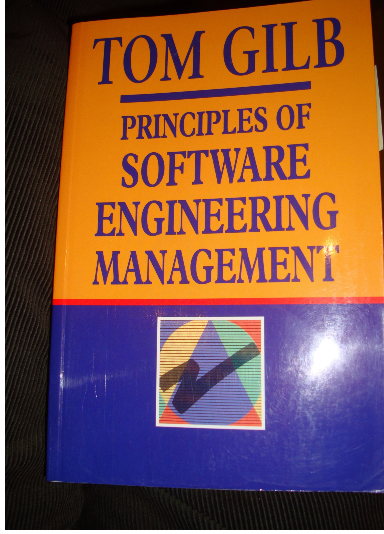
Think: Fighting Covid-19 Virus by data collection

“ensuring that the design is cost effective”

Source: Quinnan, IBM SJ, page 471

http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan

Design better, ‘as needed’.
See your need for tradeoffs, ‘as needed’



Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management. . . yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance. Design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method [illustrated in this book by Figure 7.10] consists of developing a design, estimating its cost, and ensuring that the design is cost-effective.' (p. 473)

He goes on to describe a design iteration process trying to meet cost targets by either redesign or by sacrificing 'planned capability.' When a satisfactory design at cost target is achieved for a single increment, the 'development of each increment can proceed concurrently with the program design of the others.'

'Design is an iterative pro

**It is clear from thi
appropriate balance betw
reducing the complexity
develops, and as the true**

**e in seeking the
increments, thus
each increment**

**'When the development a
computed.' (p. 474)**

crements is

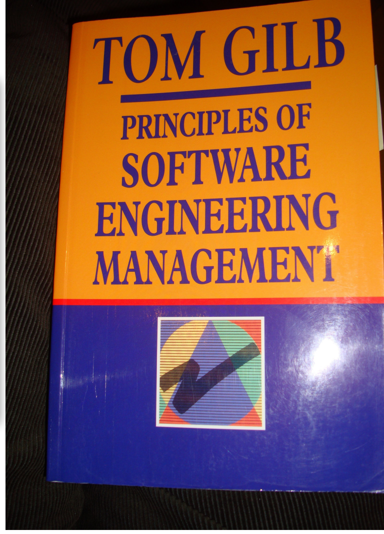
Source: Robert E. Quinnan
This text is cut from Gilb:

1980, pp. 466~77

iteration process trying to meet cost targets by either *redesign* or by *sacrificing* 'planned capability’

See diagram Figure 3 below

Do design based on incremental feedback and facts, *'one small step for mankind'*



Quinnan describes the process control loop used by IBM FSD to ensure that cost targets are met.

'Cost management. . . yields valid cost plans linked to technical performance. Our practice carries cost management farther by introducing design-to-cost guidance. Design, development, and managerial practices are applied in an integrated way to ensure that software technical management is consistent with cost management. The method [illustrated in this book by Figure 7.10] consists of developing a design, estimating its cost, and ensuring that the design is cost-effective.' (p. 473)

He goes on to describe a design iteration process trying to meet cost targets by either redesign or by sacrificing 'planned capability.' When a satisfactory design at cost target is achieved for a single increment, the 'development of each increment can proceed concurrently with the program design of the others.'

'Design is an iterative process in which each design level is a refinement of the previous level.' (p. 474)

seek … **ries**
of in
expe

'Wh … **nts**
is co

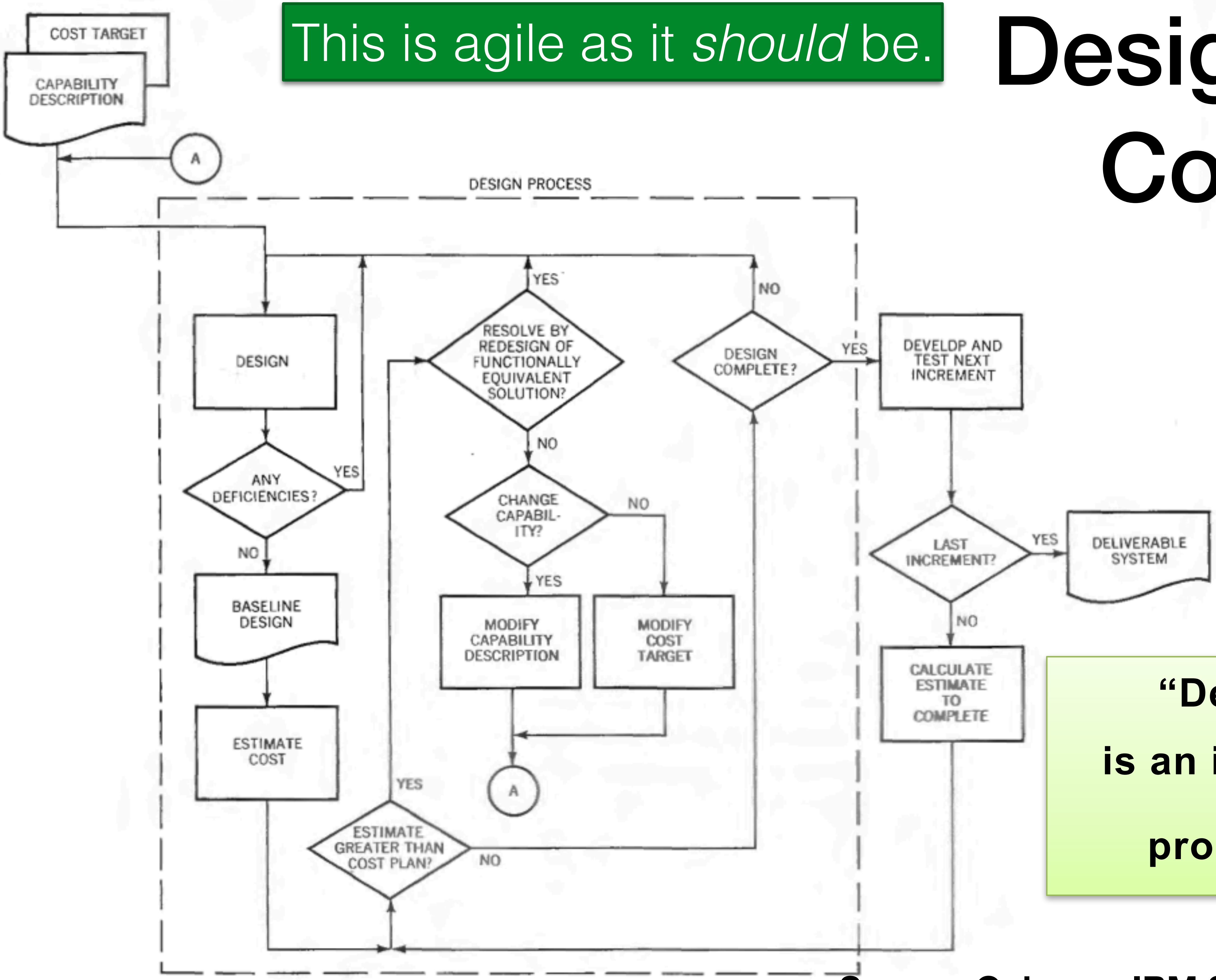
Sour … pp.
466~

Design is an iterative process

Design to Cost

This is agile as it *should* be.

Figure 3 Design-to-cost



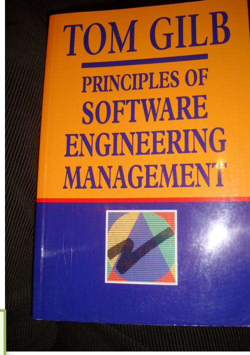
“Design is an iterative process”

Source: Quinnan, IBM SJ, page 473

http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan

And PoSEM page 105, Figure 7.10

‘Small increments’ reduce complexity



but they iterate through a series of increments,
thus *reducing the complexity of the task*,
and *increasing the probability of learning from experience*

are
nt.

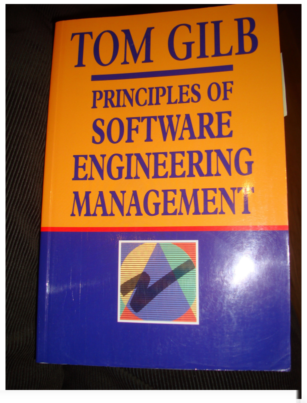
e

It is clear from this that they avoid the big bang cost estimation approach. Not only do they iterate in seeking the appropriate balance between cost and design for a single increment, but they iterate through a series of increments, thus reducing the complexity of the task, and increasing the probability of learning from experience, won as each increment develops, and as the true cost of the increment becomes a fact.

'When the development and test of an increment are complete, an estimate to complete the remaining increments is computed.' (p. 474)

Source: Robert E. Quinnan, 'Software Engineering Management Practices', IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 466~77

It is less complex to estimate future costs: *incrementally*, based on *experience*



Qu hat cost target

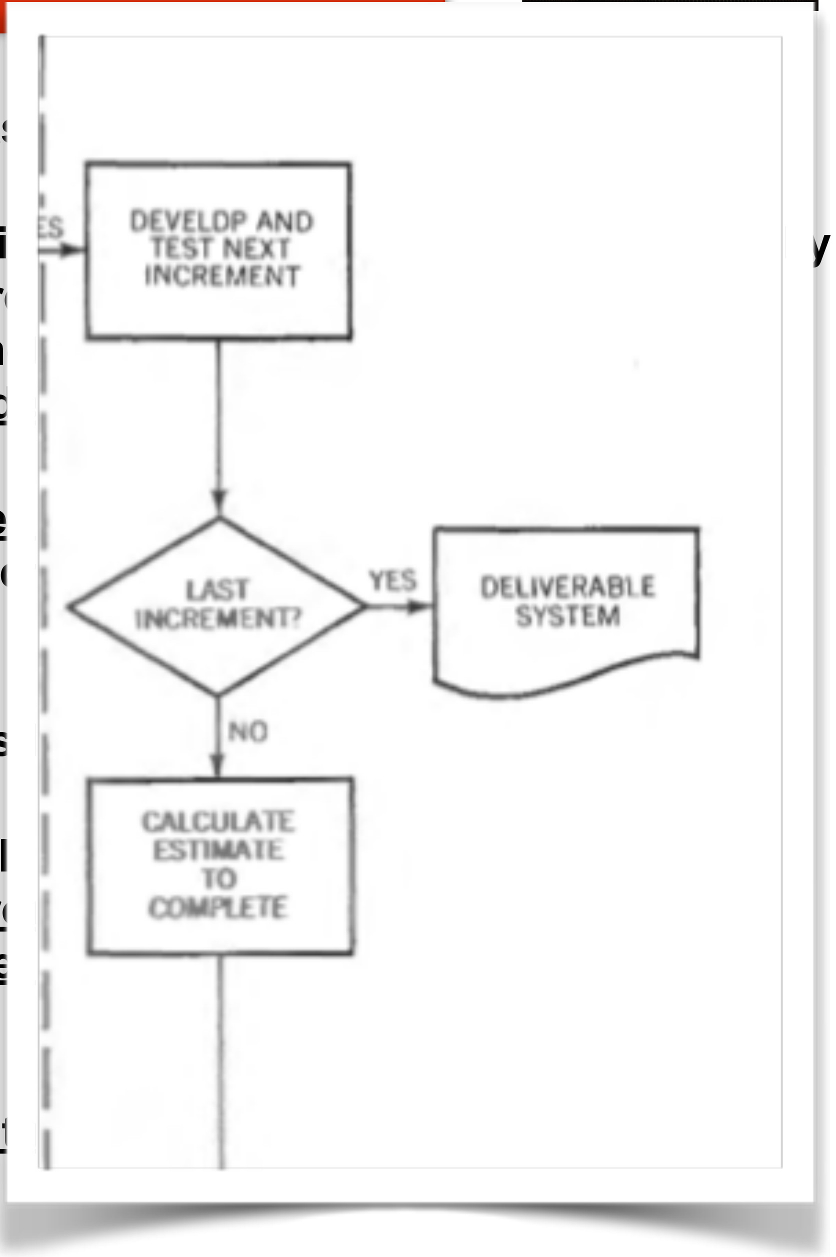
'Co nce. Our practi
int al practices ar
en gement. The m
Fig uring that the c

'pl st targets by e
inc for a single in
s.'

'De of the previous

roach. Not onl
ap hey iterate thr
rec ning from expe
de

> **"an estimate to complete the remaining increments is computed."**
>
> (See Figure 3 above for flowchart)


ES
DEVELOP AND TEST NEXT INCREMENT
LAST INCREMENT?
YES
DELIVERABLE SYSTEM
NO
CALCULATE ESTIMATE TO COMPLETE
y

'When the development and test of an increment are complete, <u>an estimate to complete t</u>
<u>computed</u>.' (p. 474)

Source: Robert E. Quinnan, 'Software Engineering Management Practices', IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 466~77
This text is cut from Gilb: The Principles of Software Engineering Management, 1988

Technoscopes Tool Area 3 of 5.

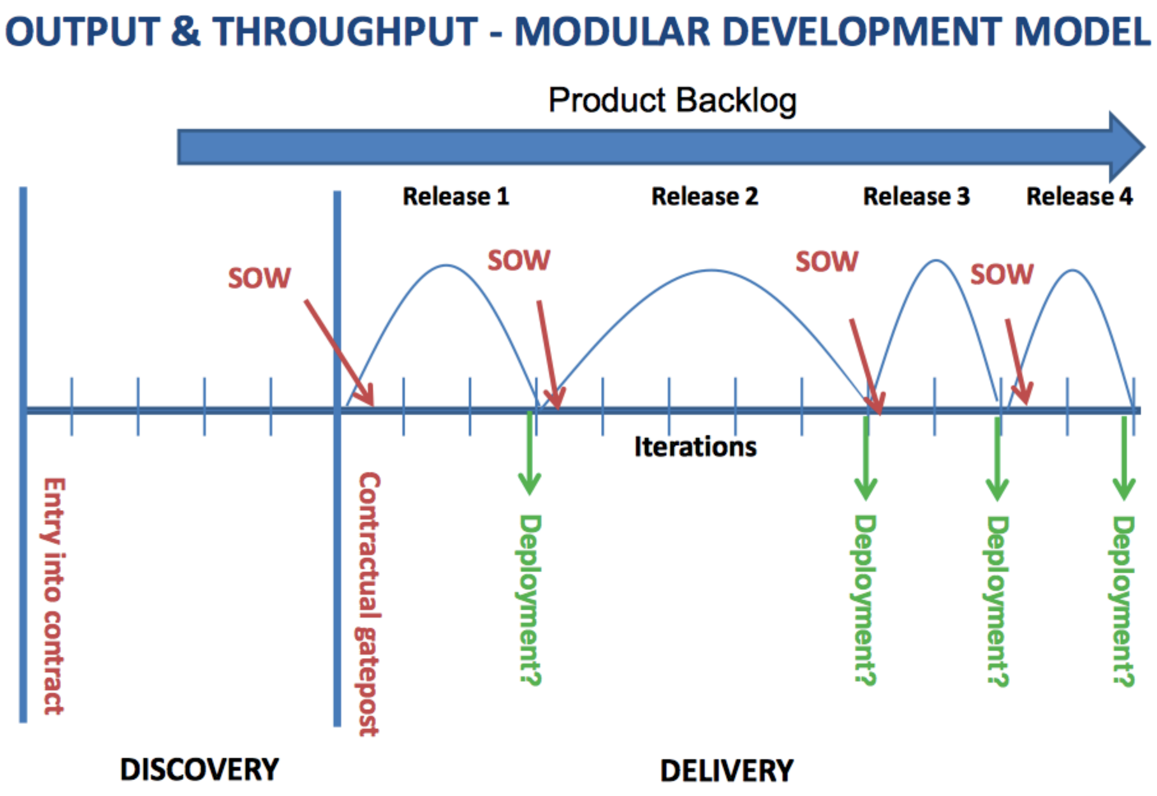
by

‘contracting for results’,

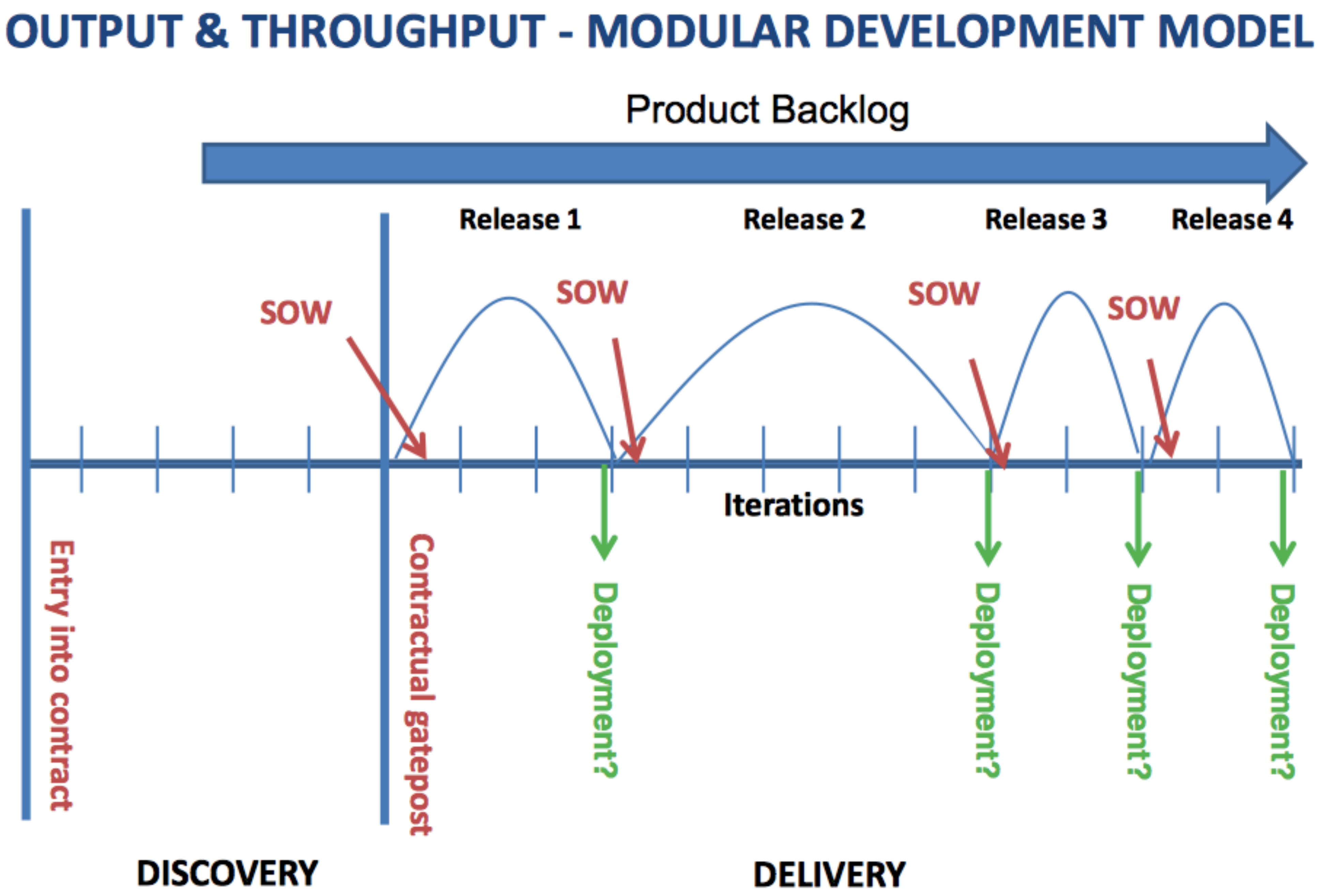
not **contracting for ‘work and work product’ (like code)**

How does ‘contracting for results’ Help us deal with system complexity?

**Because you can change the ‘inside of the black box’
Or get the contractor or Dev Team to do so
Anytime**



Detailed Result/Payment Deal At each 'Sprint': No Cure-No Pay

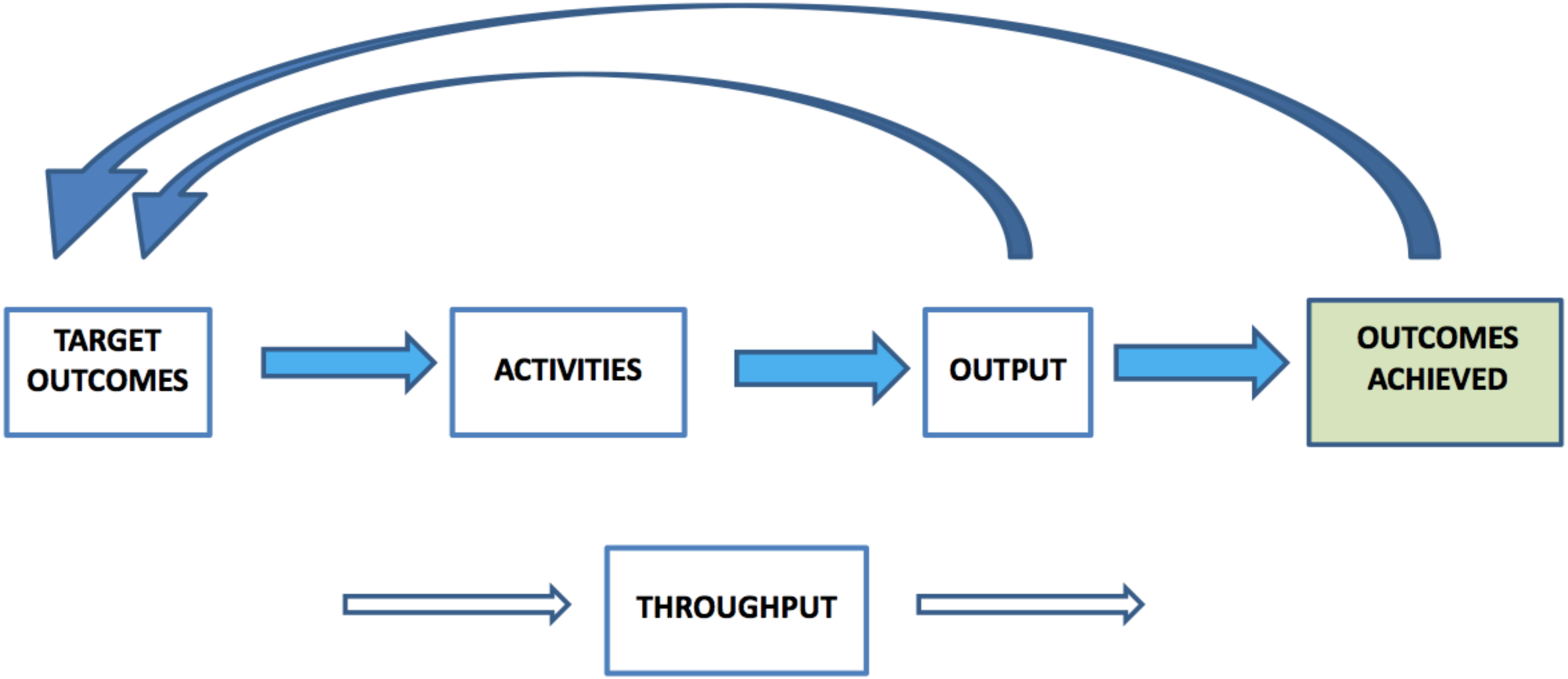


How to fight the complexity of a long-term large-scale sub-contract

SOW = Statement Of Work

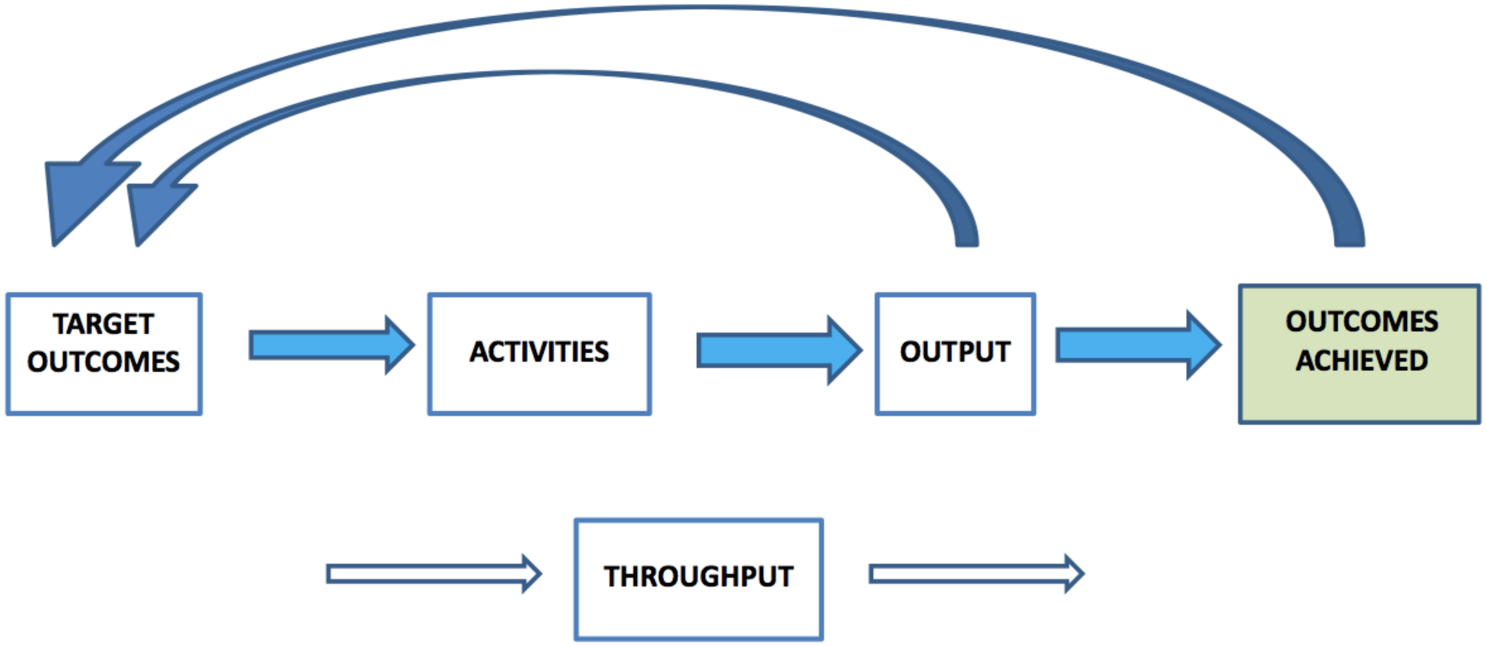
Outcome-driven view of contract metrics:

If they didn't really deliver value, try again.
If they cannot deliver, stop the supplier relationship



Why 'Result Contracting' and 'Result Payment' deals with complexity

1. **You are not bound to big contracts if you cannot control the complexity**
2. **You do not have to understand the total costs or duration up front, for large complex projects**
3. **Neither does the sub-contractor have to**
4. **You can focus on *quick wins* and *sure things*, until you reach a 'level of incompetence', regarding complexity.**
5. **You can discard a complex and failed delivery step, 2% loss, and continue the project by finding a simpler more risk-free designs**



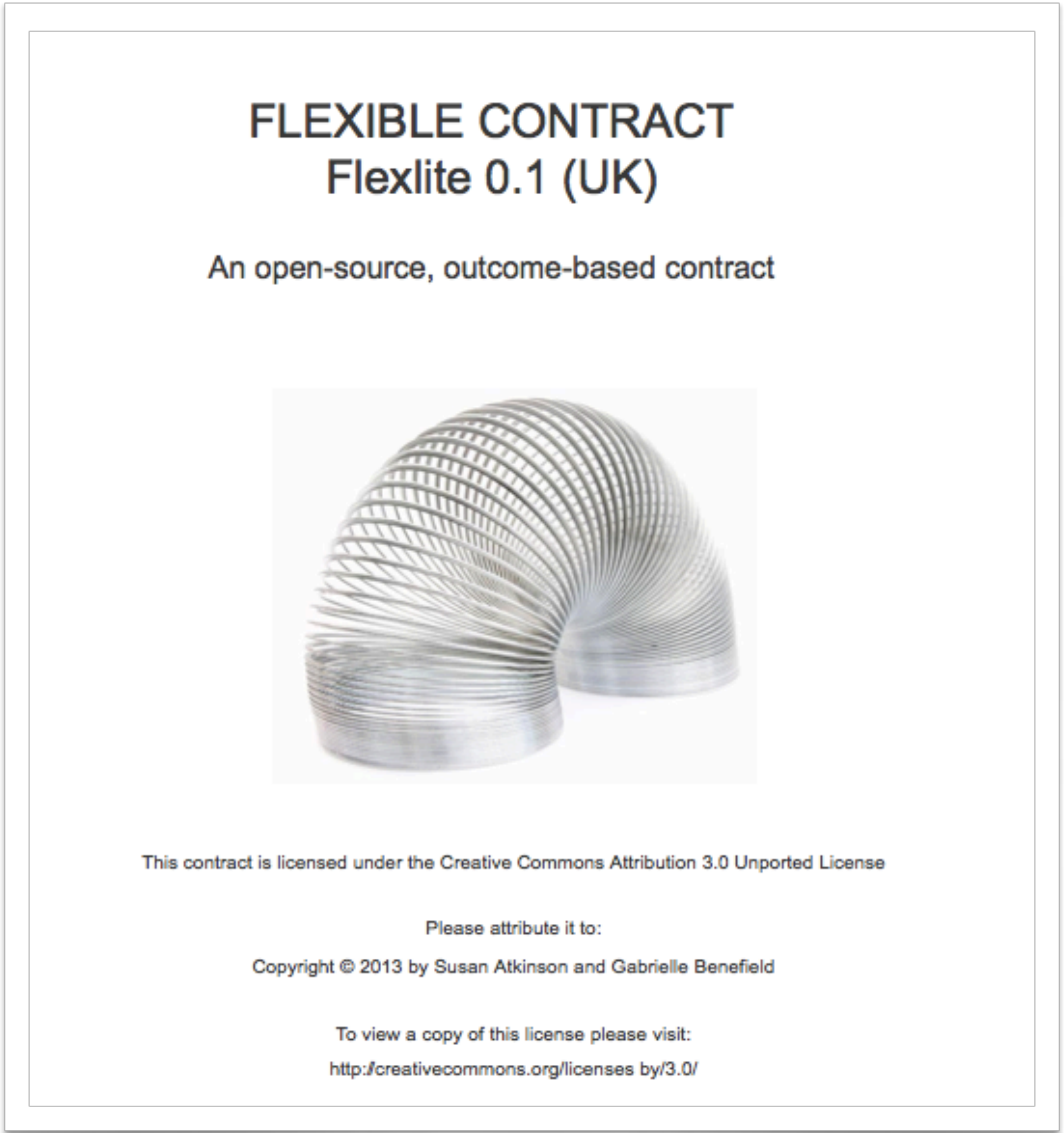
Pros and cons of the Flexible Contract

- Pros:
  - ➢ ▪A structured approach **for focusing on the customer’s strategic plan**
  - ➢ ▪The creation **of shared goals** helps to **align** the interests and motivation of the parties
  - ➢ ▪**The supplier is motivated** to achieve the target outcomes in the most cost-effective way
  - ➢ ▪**Outcomes** *are less susceptible to change*, than **output**
  - ➢ ▪The parties can **learn rapidly** what works and what doesn’t by measuring progressively the outcome delivery
- Cons:
  - ➢ ▪Lack of method and contract-process familiarity
  - ➢ ▪Outcomes are not as straightforward as other contract metrics, and require some training

The Flexible Contract is more tuned in to agile

Contract / Characteristics	Shorter development paths	Velocity-based	*'Agile methodology'*	Flexible Contract Model
Evolutionary and emergent solution				●
Experimental approach				●
Fast feedback loops / learning cycles	●			●
Rapid response to embrace changes	● ½	● ¾	● ½	●
Collaborative relationship				●

Contract Templates available for free



FLEXIBLE CONTRACT
Flexlite 0.1 (UK)

An open-source, outcome-based contract

This contract is licensed under the Creative Commons Attribution 3.0 Unported License

Please attribute it to:

Copyright © 2013 by Susan Atkinson and Gabrielle Benefield

To view a copy of this license please visit:
http://creativecommons.org/licenses by/3.0/

http://www.flexiblecontracts.com

WHAT IS A FLEXIBLE CONTRACT?

WHAT IS A FLEXIBLE CONTRACT?

A 'flexible contract' is an adaptive, outcome-based contract, which is intended to maximize the delivery of customer value. It achieves this in several ways:

The contract focuses on outcomes (that is, business objectives), which are less susceptible to change than output (such as features). By focusing on outcomes the contract also creates shared goals between the customer and supplier, which helps to align their interests and motivation.

The supplier is given the freedom to achieve the target outcomes in any way it deems effective as long as it honors the terms of the contract and stays within any constraints specified by the customer.

The fees (or at least part of the fees) should be payable on the achievement of target outcomes. The supplier is incentivized to achieve the target outcomes in the most cost-effective way, which is also of benefit to the customer.

The contract is structured as a master services agreement for the full version, or the 'lite' version using the Terms and Conditions, under which short-term statements of target outcomes (SOTOs) are called off. SOTOs work in the same way as a Statement of Work, but instead of 'work' in the form of outputs and activities, we measure outcomes achieved. The parties can respond to acquired knowledge and changes in the environment in subsequent SOTOs.

In respect of each SOTO the supplier addresses each target outcome by means of short feedback cycles. So the parties can learn rapidly what works and what doesn't by measuring outcomes achieved progressively.

The contract adopts lightweight contractual provisions. This is made possible because the parties only commit to one SOTO at a time, so the financial exposure of the customer to the supplier is minimized. This in turn means that the contract is easier to understand and requires less administrative cost, both to create and to manage. The contract is deliberately NOT focused on the activities of the supplier or the technical processes by which this value is delivered.

Define what you want, as you go, in small increments.

Learn what works

Focus on business results, not 'code'

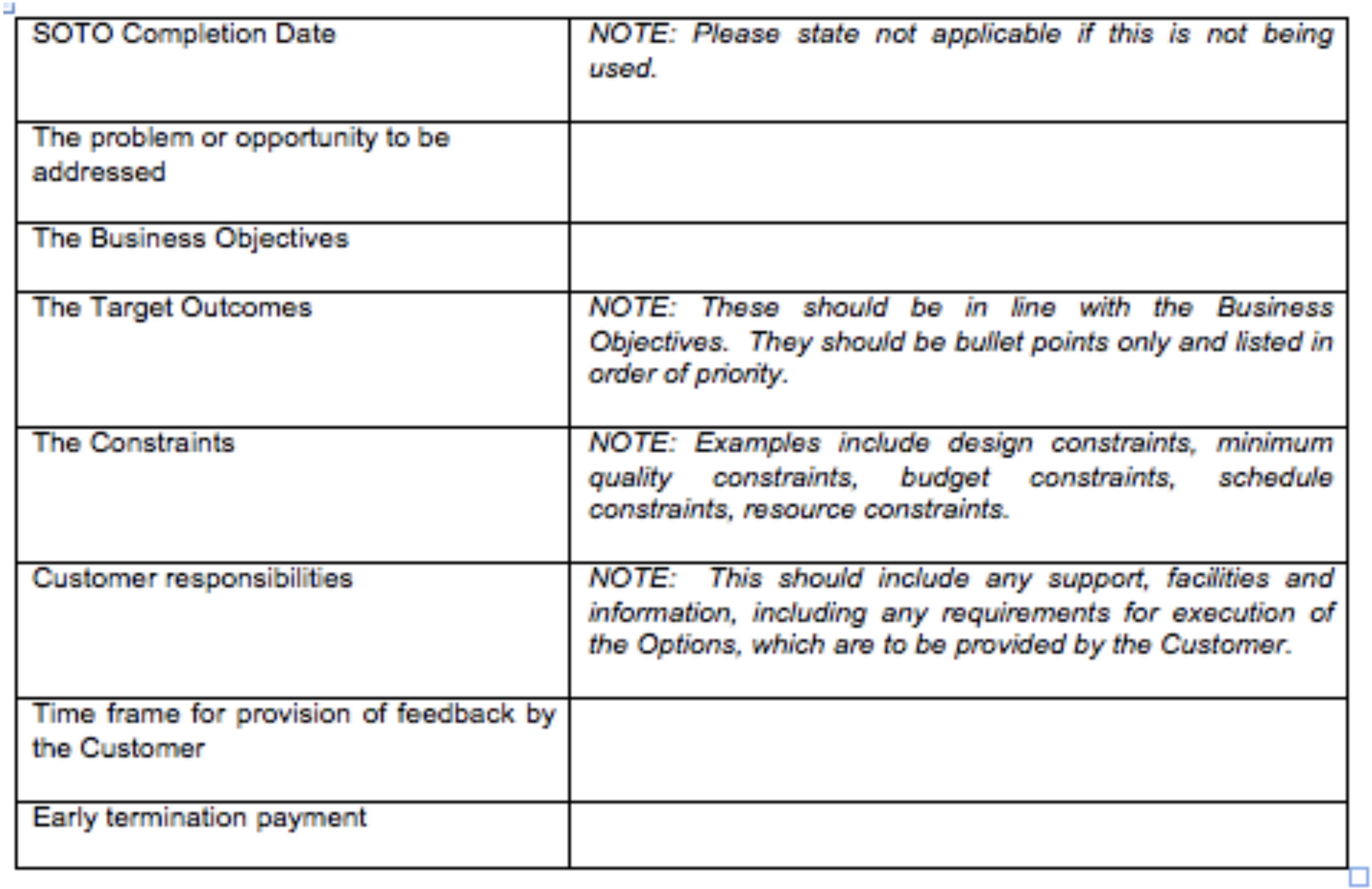
Pay for real value delivered

Prioritize high value results early.

Very low risk

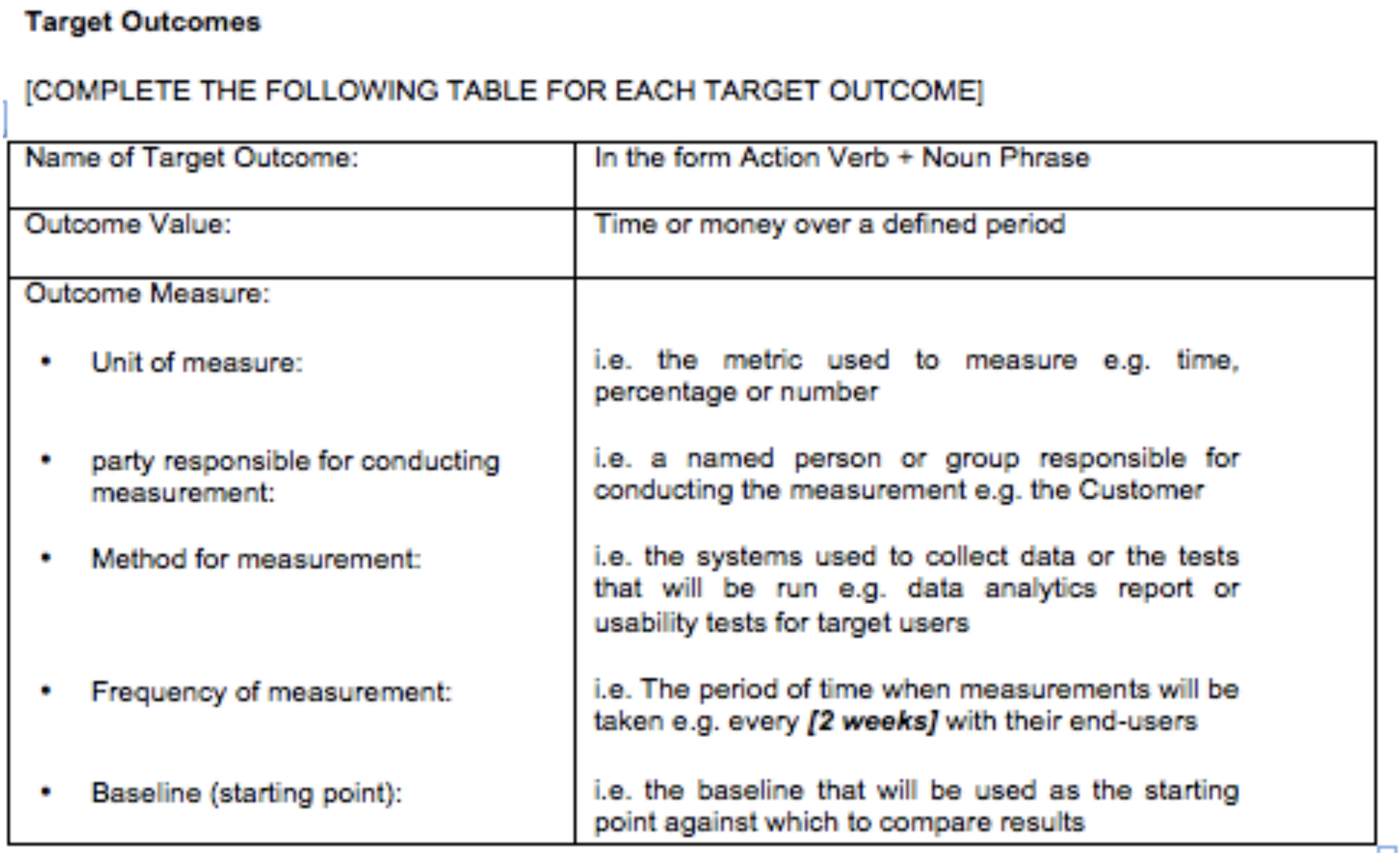
Not tied in to suppliers who cannot deliver

SOTO Specification
(from contract template)
short-term Statements Of Target Outcomes



SOTO Completion Date	*NOTE: Please state not applicable if this is not being used.*
The problem or opportunity to be addressed	
The Business Objectives	
The Target Outcomes	*NOTE: These should be in line with the Business Objectives. They should be bullet points only and listed in order of priority.*
The Constraints	*NOTE: Examples include design constraints, minimum quality constraints, budget constraints, schedule constraints, resource constraints.*
Customer responsibilities	*NOTE: This should include any support, facilities and information, including any requirements for execution of the Options, which are to be provided by the Customer.*
Time frame for provision of feedback by the Customer	
Early termination payment	

Result quantification Template
(based on Gilb Planguage)

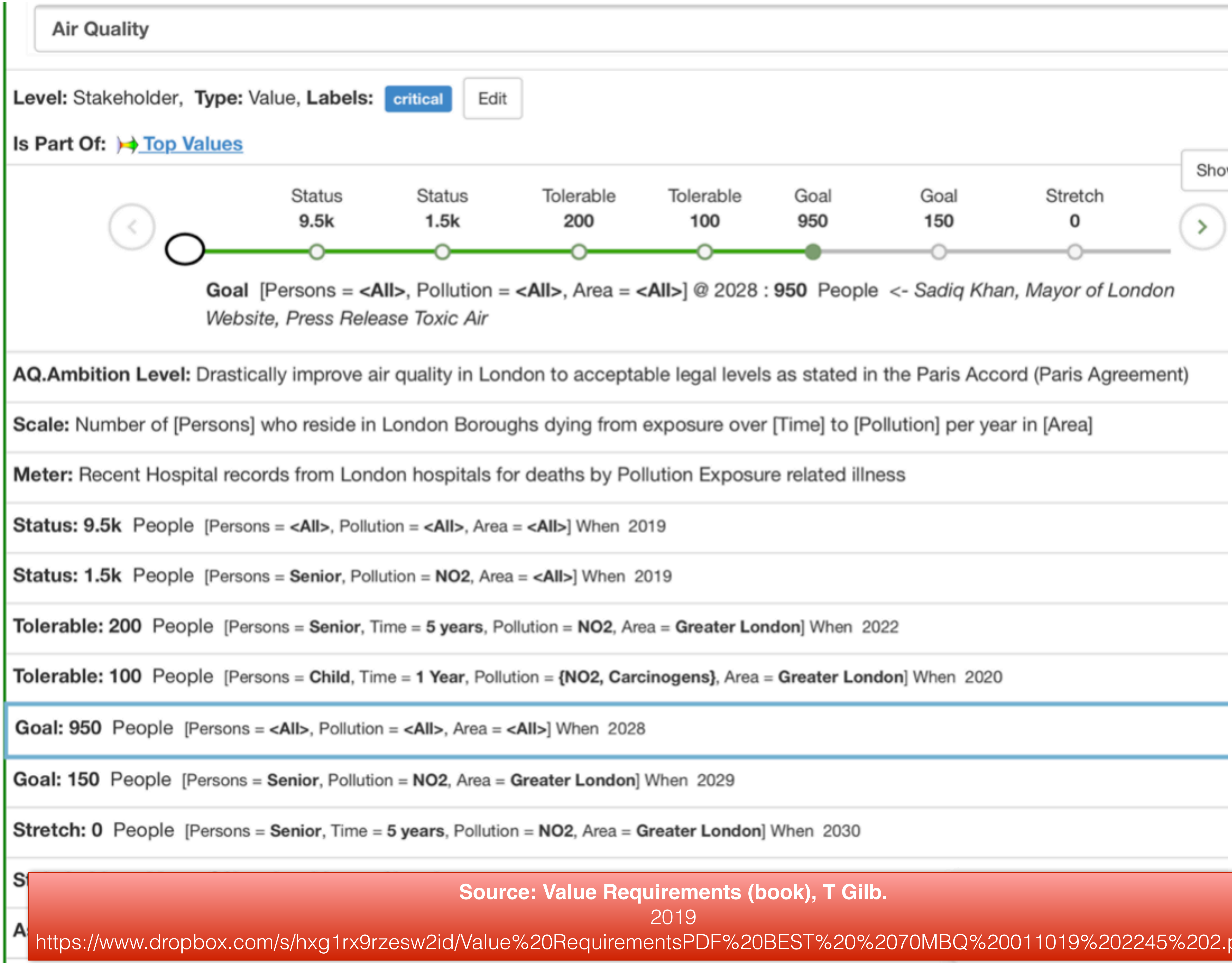


Target Outcomes

[COMPLETE THE FOLLOWING TABLE FOR EACH TARGET OUTCOME]

Name of Target Outcome:	In the form Action Verb + Noun Phrase
Outcome Value:	Time or money over a defined period
Outcome Measure: • Unit of measure: • party responsible for conducting measurement: • Method for measurement: • Frequency of measurement: • Baseline (starting point):	i.e. the metric used to measure e.g. time, percentage or number i.e. a named person or group responsible for conducting the measurement e.g. the Customer i.e. the systems used to collect data or the tests that will be run e.g. data analytics report or usability tests for target users i.e. The period of time when measurements will be taken e.g. every *[2 weeks]* with their end-users i.e. the baseline that will be used as the starting point against which to compare results

Example of thoroughly defined quantified Value for Contracting



Air Quality

Level: Stakeholder, **Type:** Value, **Labels:** critical Edit

Is Part Of: Top Values

Goal [Persons = **<All>**, Pollution = **<All>**, Area = **<All>**] @ 2028 : **950** People <- *Sadiq Khan, Mayor of London Website, Press Release Toxic Air*

AQ.Ambition Level: Drastically improve air quality in London to acceptable legal levels as stated in the Paris Accord (Paris Agreement)

Scale: Number of [Persons] who reside in London Boroughs dying from exposure over [Time] to [Pollution] per year in [Area]

Meter: Recent Hospital records from London hospitals for deaths by Pollution Exposure related illness

Status: 9.5k People [Persons = **<All>**, Pollution = **<All>**, Area = **<All>**] When 2019

Status: 1.5k People [Persons = **Senior**, Pollution = **NO2**, Area = **<All>**] When 2019

Tolerable: 200 People [Persons = **Senior**, Time = **5 years**, Pollution = **NO2**, Area = **Greater London**] When 2022

Tolerable: 100 People [Persons = **Child**, Time = **1 Year**, Pollution = **{NO2, Carcinogens}**, Area = **Greater London**] When 2020

Goal: 950 People [Persons = **<All>**, Pollution = **<All>**, Area = **<All>**] When 2028

Goal: 150 People [Persons = **Senior**, Pollution = **NO2**, Area = **Greater London**] When 2029

Stretch: 0 People [Persons = **Senior**, Time = **5 years**, Pollution = **NO2**, Area = **Greater London**] When 2030

St

A

Source: Value Requirements (book), T Gilb.
2019
https://www.dropbox.com/s/hxg1rx9rzesw2id/Value%20RequirementsPDF%20BEST%20%2070MBQ%20011019%202245%202.pdf?dl=0

So how does Flexible Contracting deal with Complexity?

- You can stop delivering when complexity is not possible to deal with or not profitable

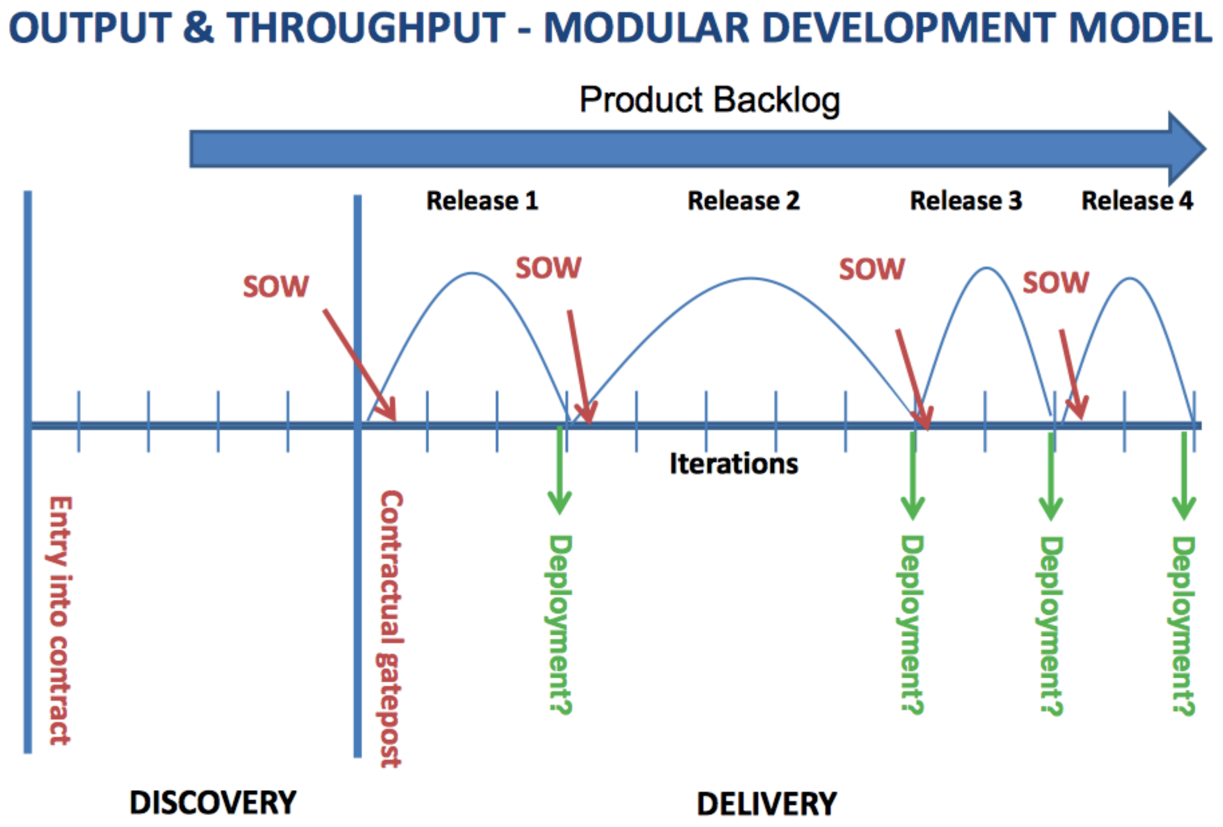
- Unexpected complexity, can be discovered early, and alternative design strategies possibly found to simplify

- No need to think about *total* cost in advance (which may be impossible because of hidden complexity)

- Potentially complex increments can be piloted early (and defeated or found OK)

- Maximum loss for discovered over-complexity is step size (2% of budget)

- Complexity can be dealt with in a 'backroom' (off line to value delivery increments (See Posem book, Evo)

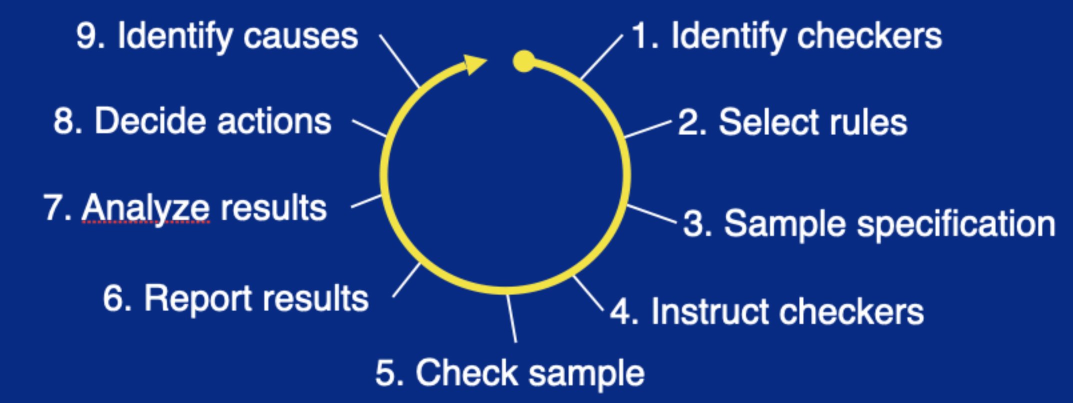


Technoscopes Area 4 of 5. By being ‘lean’: = early, = preventative.

Defect Prevention methods (SQC, DPP)
(lean, early)
Help us deal with complexity
Because

1. **They reduce the total volume of defects, in later stages, considerably, (10X, 100x more)**
2. **So we are not *overwhelmed* by the volume of defects later**

What is Specification Quality Control?
(a method for quick, cheap, frequent, continuous measurement of software development work quality)



1. Identify checkers
2. Select rules
3. Sample specification
4. Instruct checkers
5. Check sample
6. Report results
7. Analyze results
8. Decide actions
9. Identify causes

Specification** Quality Control (SQC) is a method for ensuring specifications meet established quality goals according to objective, measured standards

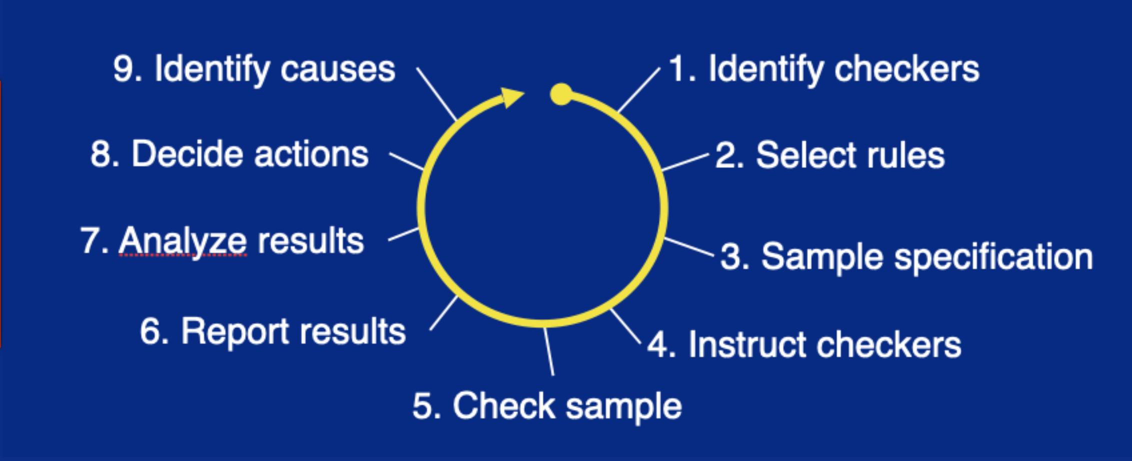
SQC prevents poor-quality specifications from **moving downstream**

Specification Quality Control emphasizes

- Cost and TTM reduction
- Defect **prevention**
- Resource efficiency
- **Early** learning
- Author confidentiality
- Quantified specification quality

** Specification: A written or electronic representation of information used to design, architect, construct, or test a system or its parts.

What is Specification Quality Control?

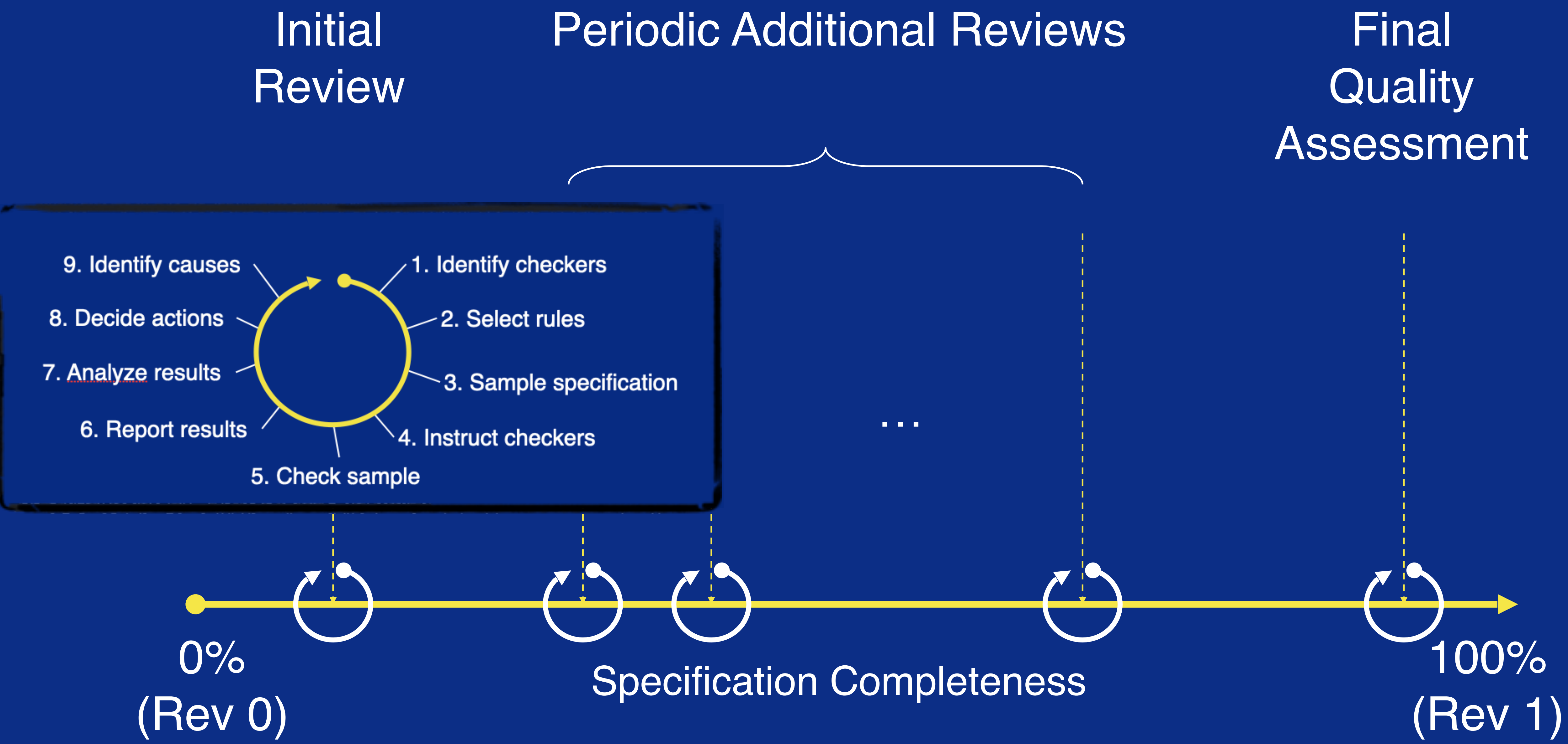


SQC is similar to traditional techniques for reviews, walkthroughs, and inspections, but has important differences and improvements:

- SQC's goal is to ***measure defect density***, not to "clean up" the specification by finding every defect in it
- SQC saves time by **checking only samples** of the specification rather than the entire thing
- SQC focuses on **major** defects – those that will take at least **10x more to correct later** than now
- SQC follows a **rigorous process**, with trained participants to help **guarantee consistently good results**

Specification Quality Control forms the backbone of an effective, efficient review structure

SQC At a Glance
(early QC, continuous QC, final QC)



Initial Review

Periodic Additional Reviews

Final Quality Assessment

9. Identify causes
8. Decide actions
7. Analyze results
6. Report results
5. Check sample
4. Instruct checkers
3. Sample specification
2. Select rules
1. Identify checkers

…

0% (Rev 0)

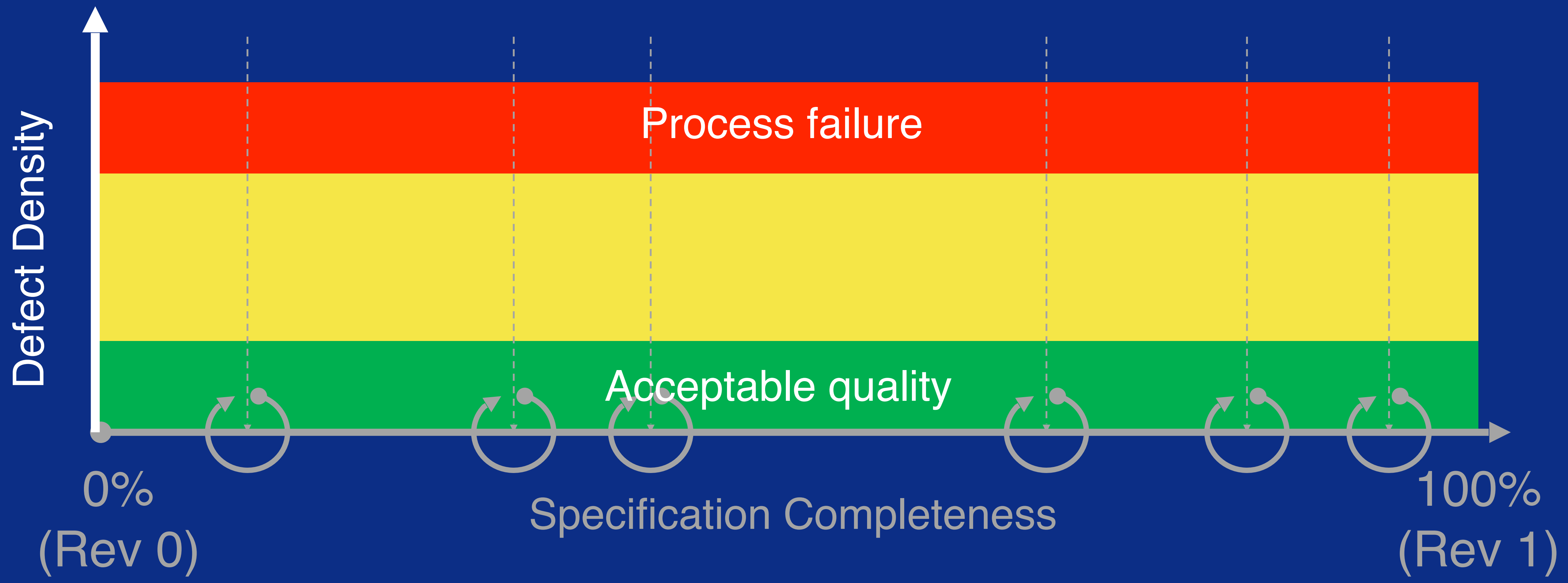
Specification Completeness

100% (Rev 1)

Specification Quality Control consists of a series of short, intense reviews that measure the defect density of a specification

SQC is Data-Driven
(based on objective facts)

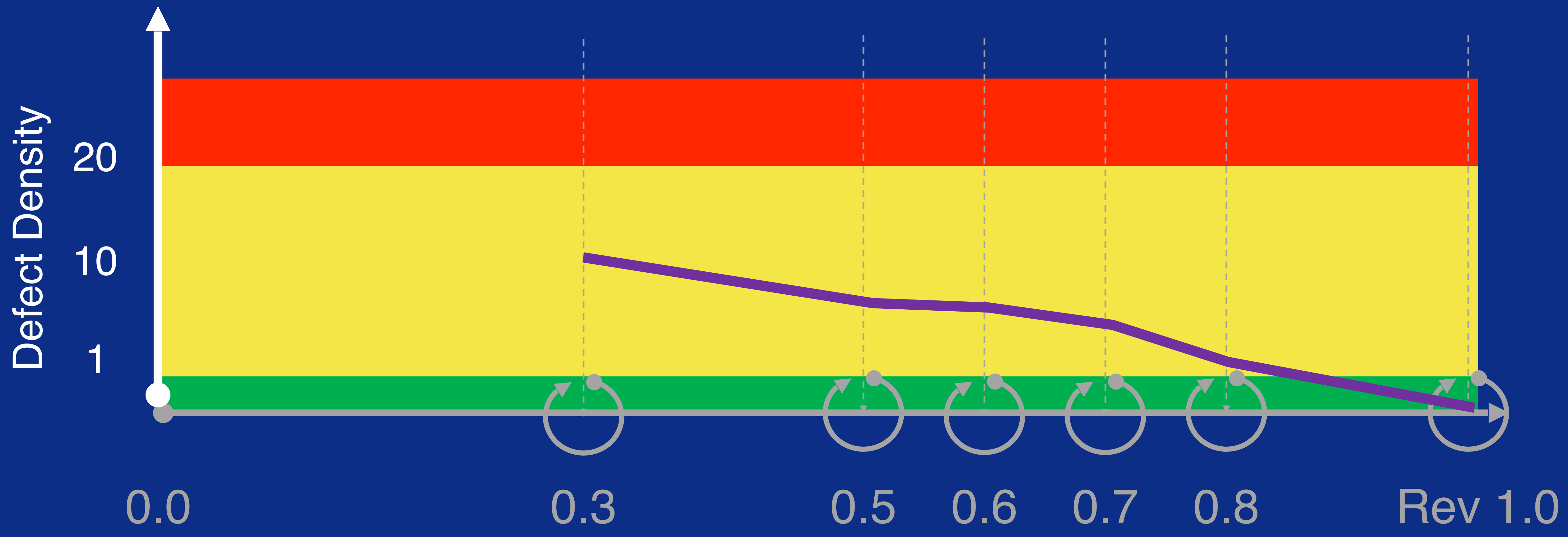
SQC tracks defect density over time to ensure good quality in the work products:



Early evidence of specification quality allows for timely corrective action, before rework costs go unbearably high

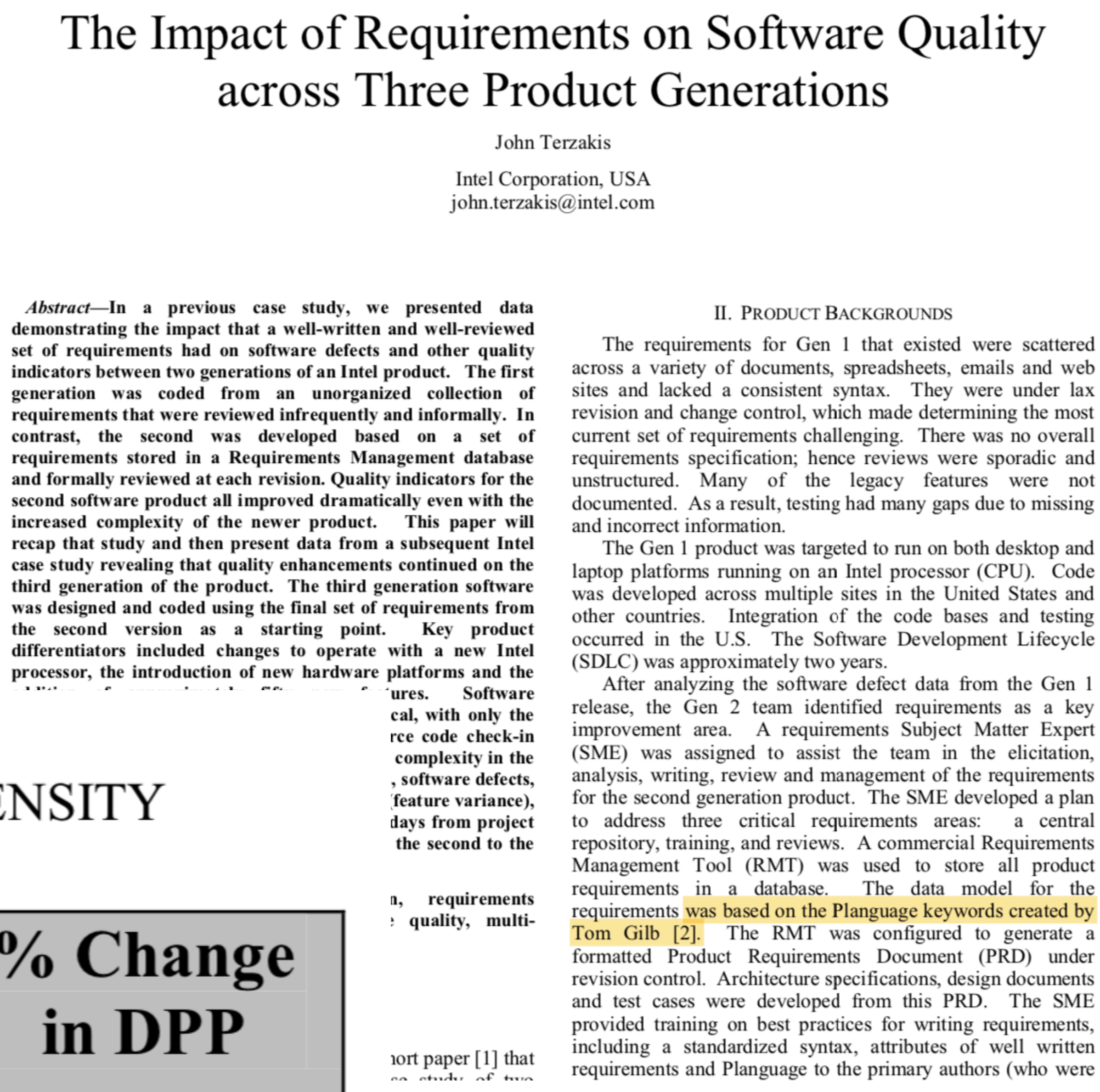
Example
(Intel published experience)

A team in Client BIOS used SQC to reduce requirements defect density by 98% over six cycles:



This effort had *significant* benefits to downstream work, including improved productivity (+233%), time to test, and customer quality

Intel Case Studies 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

ures. Software
cal, with only the
rce code check-in
complexity in the
, software defects,
feature variance),
days from project
the second to the

n, 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

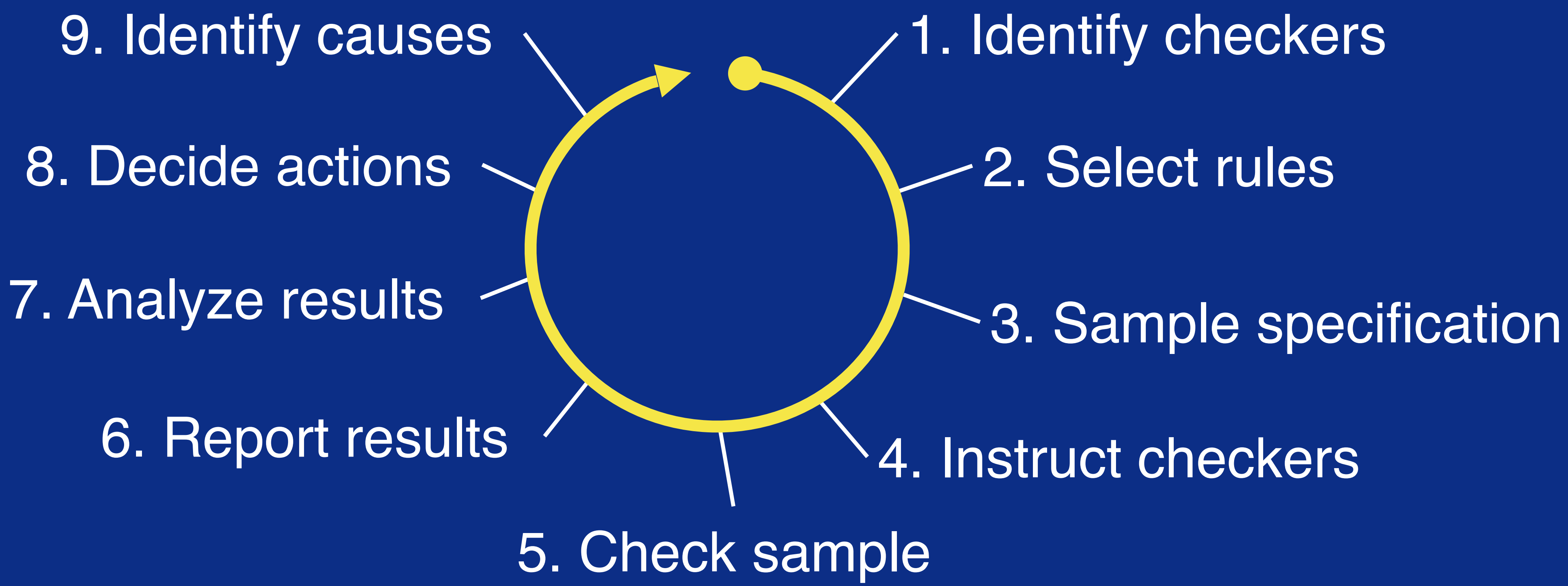
PRD Revision	# of Defects	# of Pages	Defects/ Page (DPP)	% Change in DPP
0.3	312	31	10.06	-
0.5	209	44	4.75	-53%
0.6	247	60	4.12	-13%
0.7	114	33	3.45	-16%
0.8	45	38	1.18	-66%
1.0	10	45	0.22	-81%
Overall % change in DPP revision 0.3 to 1.0: ***-98%***				

** Methods = Planguage Requirements and SQC

SQC Review Cycles

Note: Defect Density Measurement, NOT find and fix defects.
Motivate engineers to follow best practice standards.

Each SQC review cycle follows the same simple process:

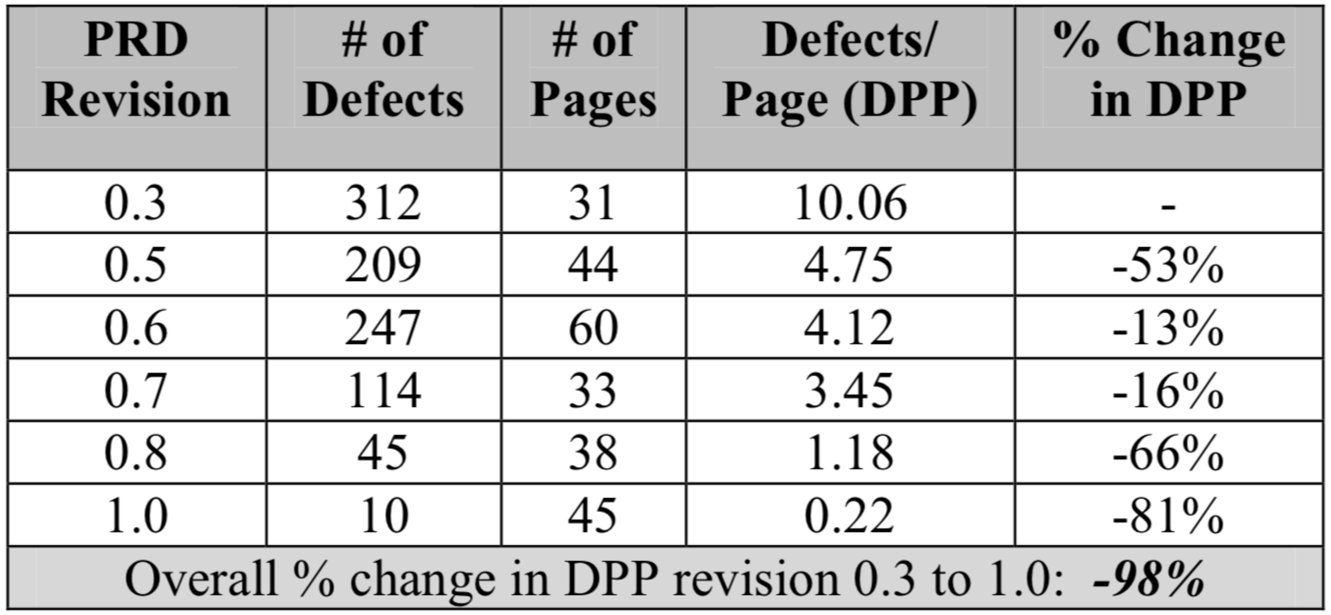


Typical time investment for one cycle: 60-120 minutes, depending on sample size

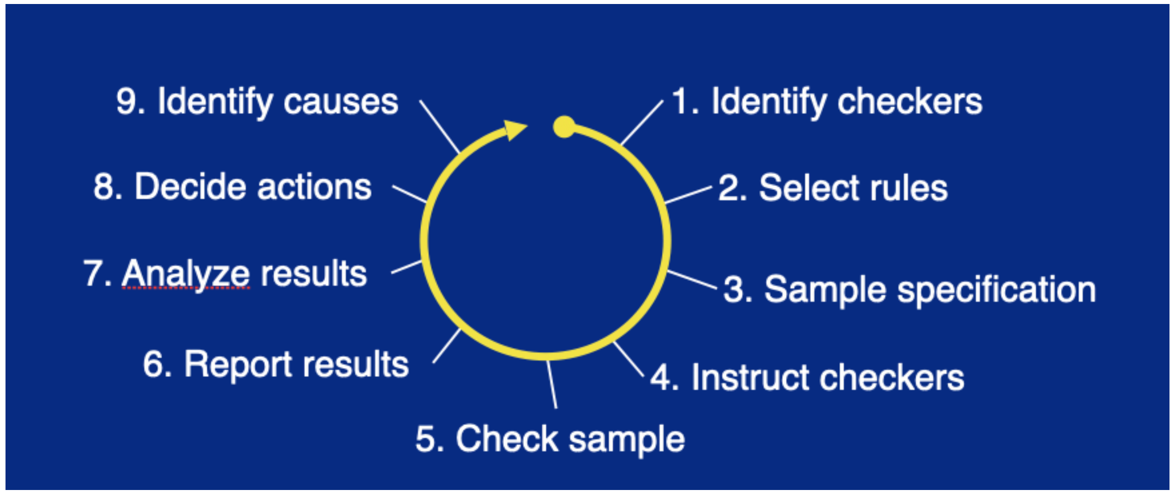
How does Spec QC Help with Complexity?

- **Sampling allows us to reduce time and cost of measurement by about 50X (take a 2% representative sample)**
- **Reduction of future problems inside the black box is about 50x (10.06 to 0.22 see table)**
- **It forces people to really learn best practices, on the job and reduces the complexity of learning a discipline by a handbook courses. (McDonnell Douglas experience)**
- **It doubles (233% Intel) software engineering productivity**

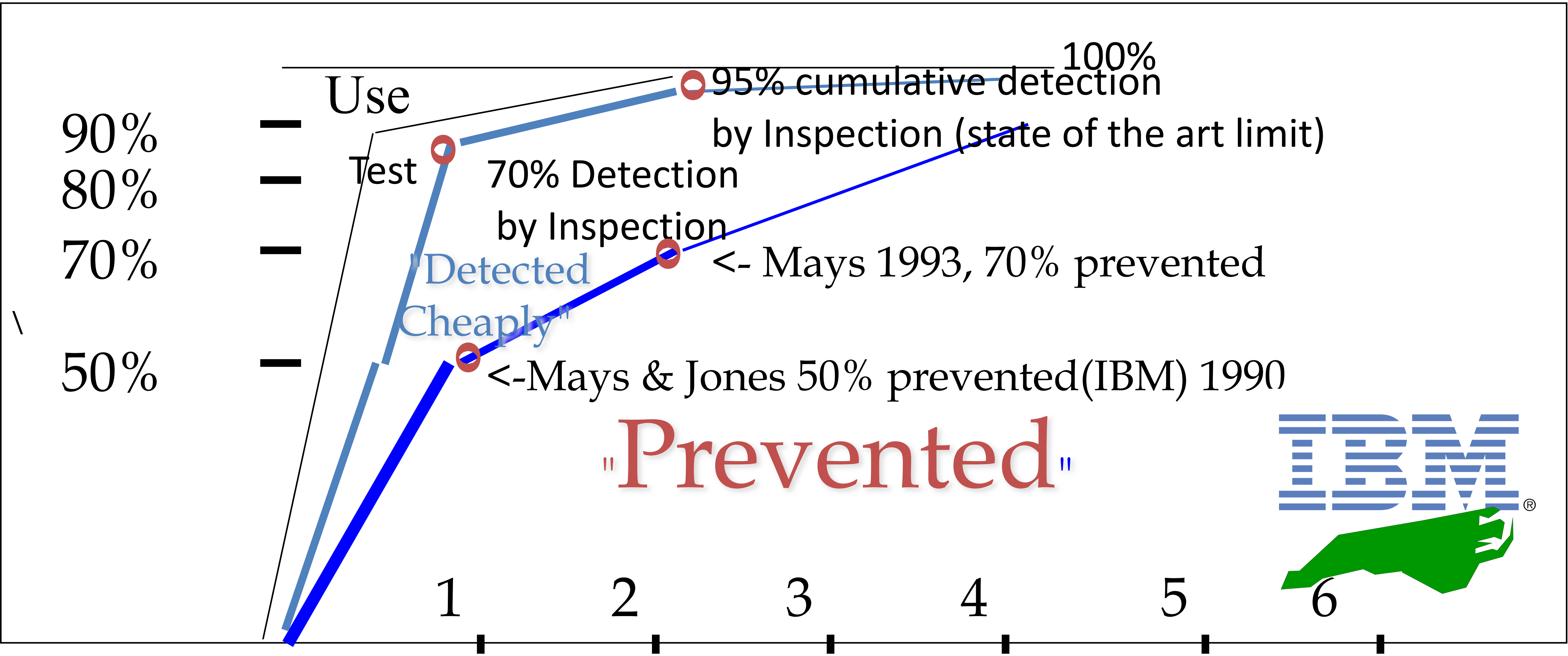
TABLE I: GEN 2 REQUIREMENTS DEFECT DENSITY



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%***				



Prevention + Pre-test Detection is the most effective and efficient: **a set of processes for fighting complexity**



- Prevention data based on state of the art prevention experiences (IBM RTP), Others (Space Shuttle IBM SJ 1-95) 95%+ (99.99% in Fixes)
- Cumulative Inspection detection data based on state of the art Inspection (in an environment where prevention is also being used, IBM MN, Sema UK, IBM UK)

IBM MN & NC DPP Experience

Attacking large organization software engineering complexity by DELEGATION OF POWER TO GRASS ROOTS TO IMPROVE THE ORGANISATION

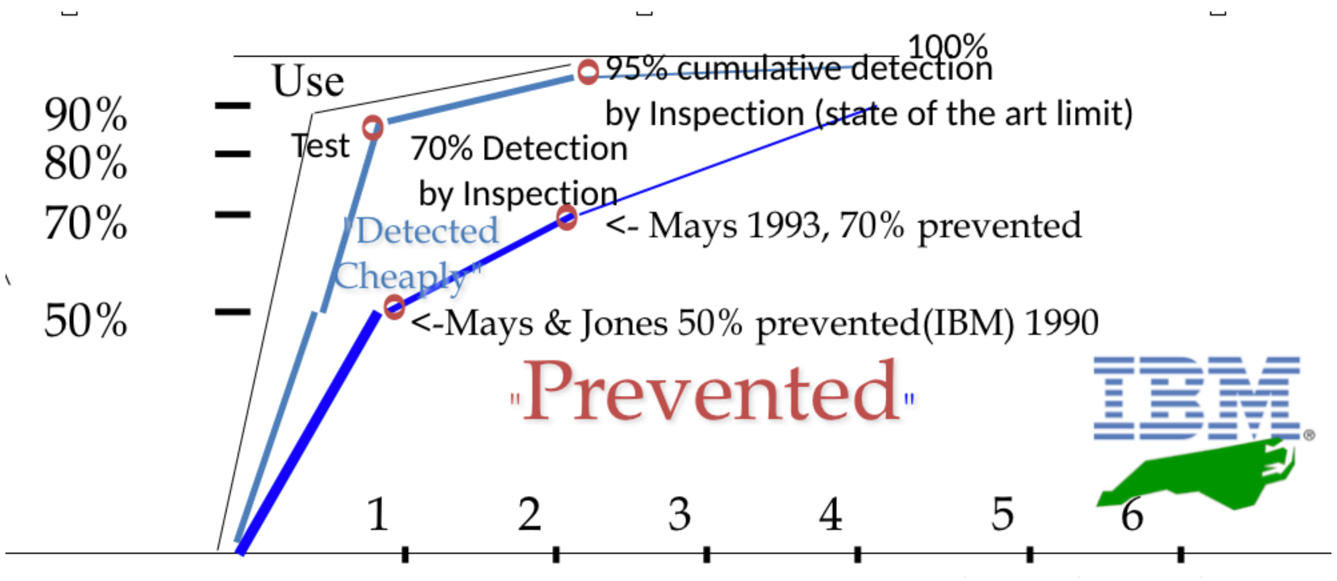
- **2162 DPP Actions implemented**
  - between Dec. 91 and May 1993 (30 months)<-Kan

- RTP about 182 per year for 200 people.<-Mays 1995
  - 1822 suggested ten years (85-94)
  - 175 test related
- RTP 227 person org<- Mays slides
  - 130 actions (@ 0.5 work-years
  - 34 causal analysis meetings @ 0.2 work-years
  - 19 action team meetings @ 0.1work-years
  - Kickoff meeting @ 0.1 work-years
  - TOTAL costs 1% of org. resources

- ROI DPP 10:1 to 13:1, internal 2:1 to 3:1
- Defect Rates at all stages 50% lower with DPP

How does Defect Prevention Process deal with Complexity?

- **It gives a radical reduction (50% to 99%) reduction in problems (like bugs) which occur at all**
- **The grass roots staff themselves will quickly see recurrent problems**
- **The grass roots staff will only suggest the simplest process change they can live with**
- **The staff will not suggest changes that they think makes their work unnecessarily complex**



Technoscopes Area 5 of 5

By using ‘scale free’ methods:

‘scale’ does *not matter*

‘Bigger’ does not threaten you with complexity

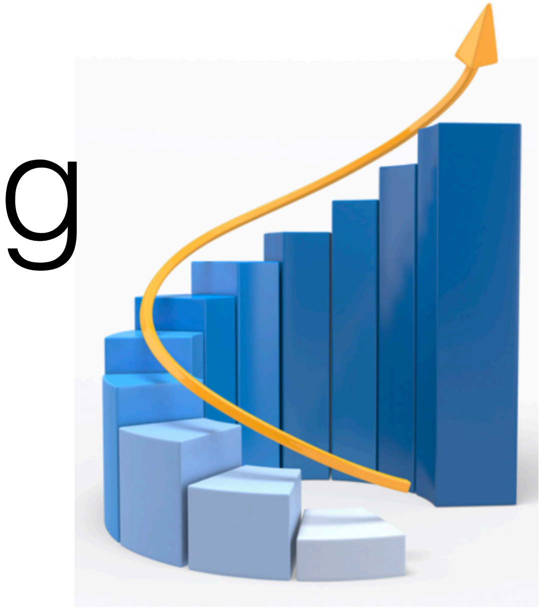
If we are scale-free then we do not have to worry about

‘rapid or large scaling up’ of a system

causing it to be

too complex to handle easily

Erik Simmons, Intel Scaling



- “ I’m deeply interested in scale-free practices.
- I’m also interested in specific practices tuned to large, small, complicated, and complex projects,
- but I find particular power in scale-free practices.
- Your work for decades has been focused on a very good set of these.
  - SQC, for example, works on any size specification. It does not (need to) scale.
  - SQC: (Specification Quality Control).see immediately previous slides in Technoscopes area 4 of 5.
-
- BTW, I think the agile principles are also quite scale-free. But most Scrum practices are definitely not.
-
- So, perhaps you can chart a better course by advocating for use of scale-free core practices,
  - augmented with a set of specific, tailored practices
  - that are effective for the size of the project in question.” <- ES, Intel

My Scale Free **Methods:** most are in this talk

1.**Quantification of Values [10***, VP 1.1**].**

2.**Quantification of short term and long term costs [VP 3.4, VP 4.5, VP 6.7].**

3.**Design to Cost: Top Level Architecture [VP 7.9, 10].**

4.**Dynamic Design to Cost: Each Delivery Cycle [12 C, VP 4.5, VP 2.5, VP 2.3, 5, 10, 12].**

5.**Quality Control of Plans, Contracts, Code and all written artifacts [VP Part 2, VP Part 4, VP 7.7].**

6.**Flexible Contracting [12, VP 4.5].**

7.**Value delivery Cycle Measurable Feedback, Learning and Change [4, VP 7.3, VP 9.8, VP 6.7, VP 8.6, 2, 9, 10, 11, 14].**

8.**Value Decision Tables (Impact Estimation Tables) [9, VP 2.3, VP 4.4, VP 5.3, 13].**

9.**Risk Management in all aspects of planning and Management [VP Ch. 7], 12.**

10.**Intelligent Prioritization Policies: for short term and long term [VP Ch. 6, 12, 13, 14].**

*** = "Beyond Scaling: Scale-free Principles for Agile Value Delivery - Agile Engineering.

40 practical Engineering ideas for scaling agile development successfully all the time."

A very short pdf paper, supported by references to necessary detail.

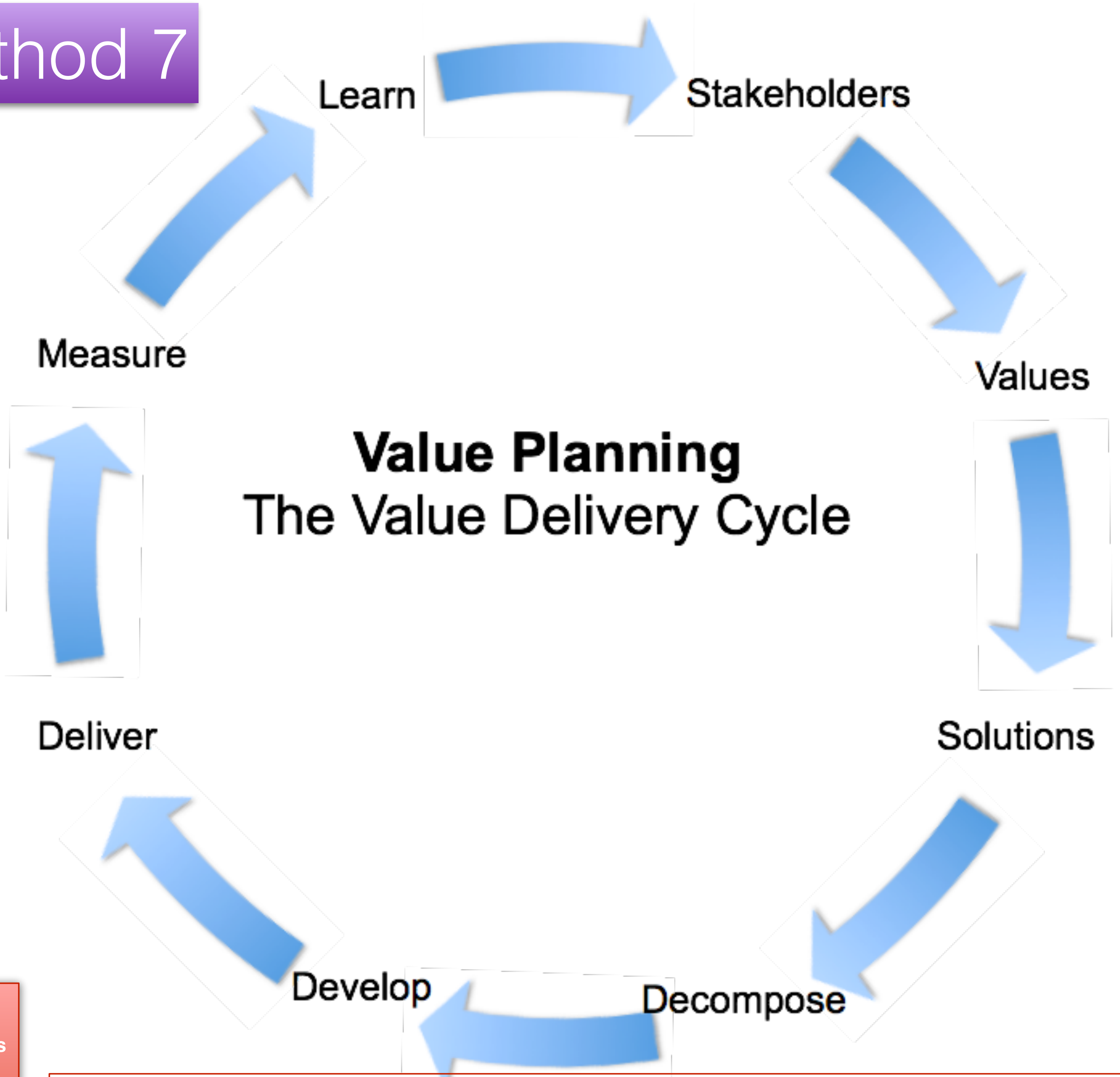
http://www.gilb.com/dl865

https://tinyurl.com/OSWAVP

** 'VP' = Value Planning book by chapter

Method 7. Value delivery Cycle Measurable Feedback, Learning and Change [4***, VP 7.3 **, VP 9.8, VP 6.7, VP 8.6, 2, 9, 10, 11, 14].

A detail of Method 7



Value Planning Cycle = 'Evo'
Cycle of Value delivery of any size project

*** = "Beyond Scaling: Scale-free Principles for Agile Value Delivery - Agile Engineering.

40 practical Engineering ideas for scaling agile development successfully all the time."

A very short pdf paper, supported by references to necessary detail.

http://www.gilb.com/dl865

https://tinyurl.com/OSWAVP
** = Value Planning book by chapter

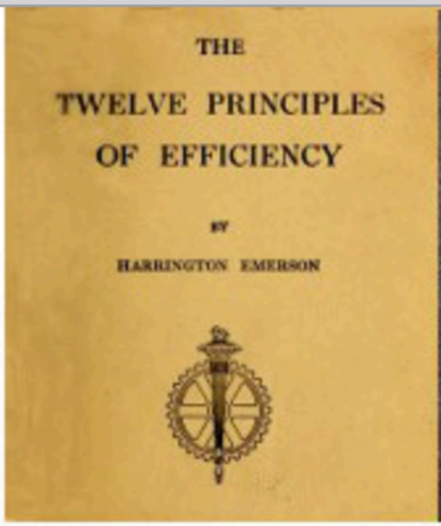
Emerson's Principle that Principles beat methods

- "As to **methods**, there may be a million and then some, but **principles** are few.
- The man who grasps principles can successfully select his own methods".

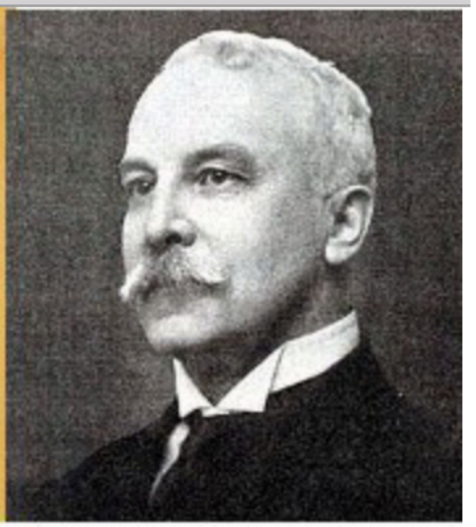
- - **Emerson, Harrington**
- (Not as thought, R W E)

  –

R. W. Emerson



Book Cover

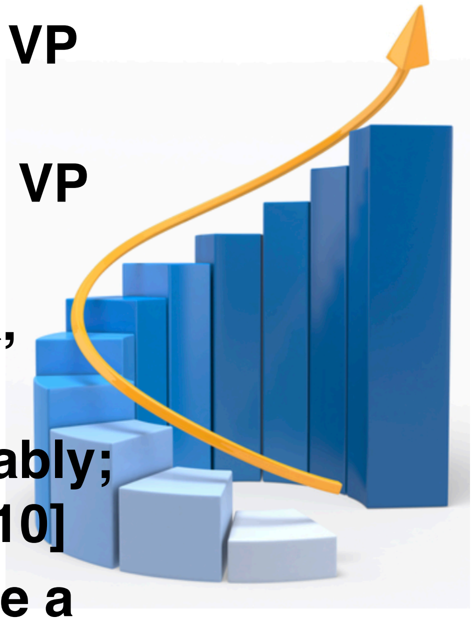


Harrington Emerson

My Scale-free **Principles**

(most are treated in this talk)

1. **Keep focus on measurable delivery of critical values and their costs. [3***, 4, 5, 6, 9, 10, 12, VP (20) Part 1, VP 10.6 **]**
2. **Deliver value early, quickly and regularly: in roughly 2% increments. [14, 11, VP Ch.4, 2, 5]**
3. **Do NOT focus on code delivery; focus on overall system value and costs. [VP Ch.4, 10D, 10F, 13, VP 3.4, VP 2.10, VP 9.8, 4, 12]**
4. **Focus on quantified *critical stakeholder* values. [19, VP 3.4, VP 3.7, VP 3.9, VP 3.10 VP 4.2, 10]**
5. **Synchronize all teams in terms of measurable value delivery. [VP 3.3, VP 3.4, VP Part 1, VP 3.6, VP 3.8, VP 8.4 , 11, 12, 13]**
6. **Solve big problems through ingenious architecture; not through coding faster. [VP 4.5, VP 5.1, VP 5.3, VP 7.2, 15]**
7. **Decompose the large problems by incremental value deliveries: not code deliveries. [7, VP Ch. 5, VP 5.1, VP 5.6 , 10, 11, 13, 15]**
8. **The software component needs to be integrated into the total system of hardware, data, people, culture. [VP 5.2, 10]**
9. **If your team cannot deliver small increments of real value early, frequently, and predictably; they are incompetent and need to be abandoned for those who can deliver. [7, VP 2.8, 10]**
10. **Never commit to contacts for *work done* or *code delivered* alone: there must always be a sufficiently large contractual protection, of paying for measurable value delivered. [12, 15].**



*** = "Beyond Scaling: Scale-free Principles for Agile Value Delivery - Agile Engineering.

40 practical Engineering ideas for scaling agile development successfully all the time."

A very short pdf paper, supported by references to necessary detail.

http://www.gilb.com/dl865

https://tinyurl.com/OSWAVP

** 'VP' = Value Planning book by chapter

My Own scale-free ‘Project Development Process’:
Uses ‘delegation of **design** to implementors and programmers’

Competitive Engineering: Book 2005
http://tinyurl.com/CEset2015
(link tested 31 03 2020)

- Make *developers* **responsible**
  - **for delivery of the ‘quantified’ critical requirements levels**
    - (Performance, Qualities, cost, deadline)
- Give them the **freedom to decide the ‘right’ designs**
  - With immediate responsibility to *measure* that they are delivering the results
- Get the ‘unprofessional’ users and **customers ‘off their backs’**
  - Avoid receiving features and stories; avoid ‘architecture from managers’.
    - which are usually amateur design, by people who have no overview or responsibility or design ability (users and customers, and managers)
- **Elevate your talent** by becoming a **real ‘software ENGINEER’**
  - With coding-expert craftsmanship, as your basic talent

Cases and real examples
‘Value Driven Project Management’ slides
Includes ‘Confirmit’ Case, slide 70 on.
http://www.gilb.com/dl152

How does ‘scale free’ deal with complexity?

- **Rapid and unexpected scaling up** (think Covid-19, NHS) **will not break the system.**

- **No need to learn and apply quite different development process methods, for large complex projects** (think Scrum, Safe)

- **Reduction of failed projects, which would fail due to *unexpected* complexity**

‘Scale Free’ end note

We need to take these scale-free engineering ideas seriously

if we are to get better control over large-scale software and systems engineering projects.

- The ideas have serious practical international experience,
- and can be tried out one-by-one.
- They can be added to any other practices, that are, or will be successful for you,
- They are free ideas.

‘This stuff works!”
(Erik Simmons, Intel
Experience 1999 to 2016 for 20,000 engineers)

Just do it!

COMPETITIVE
TOM GILB
COMPETITIVE ENGINEERING
A HANDBOOK FOR SYSTEMS ENGINEERING, REQUIREMENTS ENGINEERING, AND SOFTWARE ENGINEERING USING PLANGUAGE
ELSEVIER

Get a free e-copy of ‘Competitive Engineering’ book.
https://www.gilb.com/p/competitive-engineering

End of 1 hour and 30 minutes minute talk

Blog (Based on Value Planning book):
www.Gilb.com/blog

Principles Videos (free first 4)
https://www.gilb.com/p/principles

See my TEDx Talk Quantify the un-quantifiable

Pictures from Spring edition of the Masterclass
7 / 41

below is a set of slides which go deeper into handling complexity than we can deal with in the talk timing. but I include them for advanced study.

They are based on my paper
CONFRONTING WICKED PROBLEMS:
and some Planguage Tools to deal with them.
http://www.gilb.com/dl866

WICKED PROBLEMS ARE POSSIBLY SIMPLE; IF YOU KNOW HOW

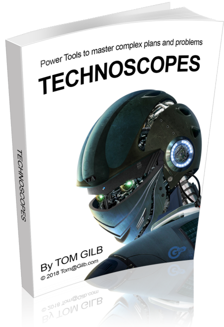
Tom’s Personal View of Evil
23 June 2016, GilbFest

BASED ON T GILB PAPER

“CONFRONTING WICKED PROBLEMS:

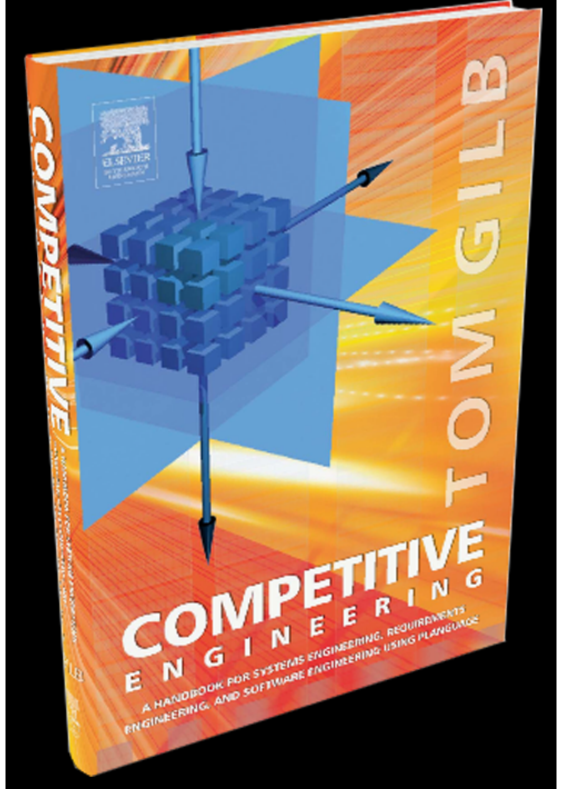
and some *Planguage* Tools to deal with them”

10 January 2016,
gilb.com/dl866



How Planguage Tools Help Whack Wickedness.

Technoscopes.(free digital book about 100 tools to fight complexity) ONLY FOR TALK!
https://www.dropbox.com/sh/p439ms65ifyid72/AAAbL26ly1ss2mH5K6RB2F_la?dl=0


COMPETITIVE ENGINEERING
TOM GILB
A HANDBOOK FOR SYSTEMS ENGINEERING, REQUIREMENTS ENGINEERING, AND SOFTWARE ENGINEERING USING PLANGUAGE

The following slides
Are discussion, or rebuttal
of the
Wicked Problems Characteristics
(slide 4) list
They are not intended
for presentation in this short talk,
But I hope some individuals can use the
Information and arguments to understand
complexity better.
And to be skeptical of
delivered truths from academics

Get a free e-copy of
'Competitive Engineering'
book.
https://www.gilb.com/p/competitive-engineering

https://tinyurl.com/OSWAVP
Value Planning book
by chapter

W1. There is no *definitive formulation* of a wicked problem.

Planguage (The Planning Language) does not need or expect a 'definitive formulation' of a problem.

Planguage allows you to specify any set of *problem statements* (value objectives, constraints, assumptions, constrained strategies, budgets, deadlines, stakeholders) that might be useful.

They can all be modified at any time. They can be versioned. They can be officially sanctioned or approved, until further notice. They can be quality controlled. The quality , relevance, correctness and usefulness of any class of problem statement can be gradually enhanced.

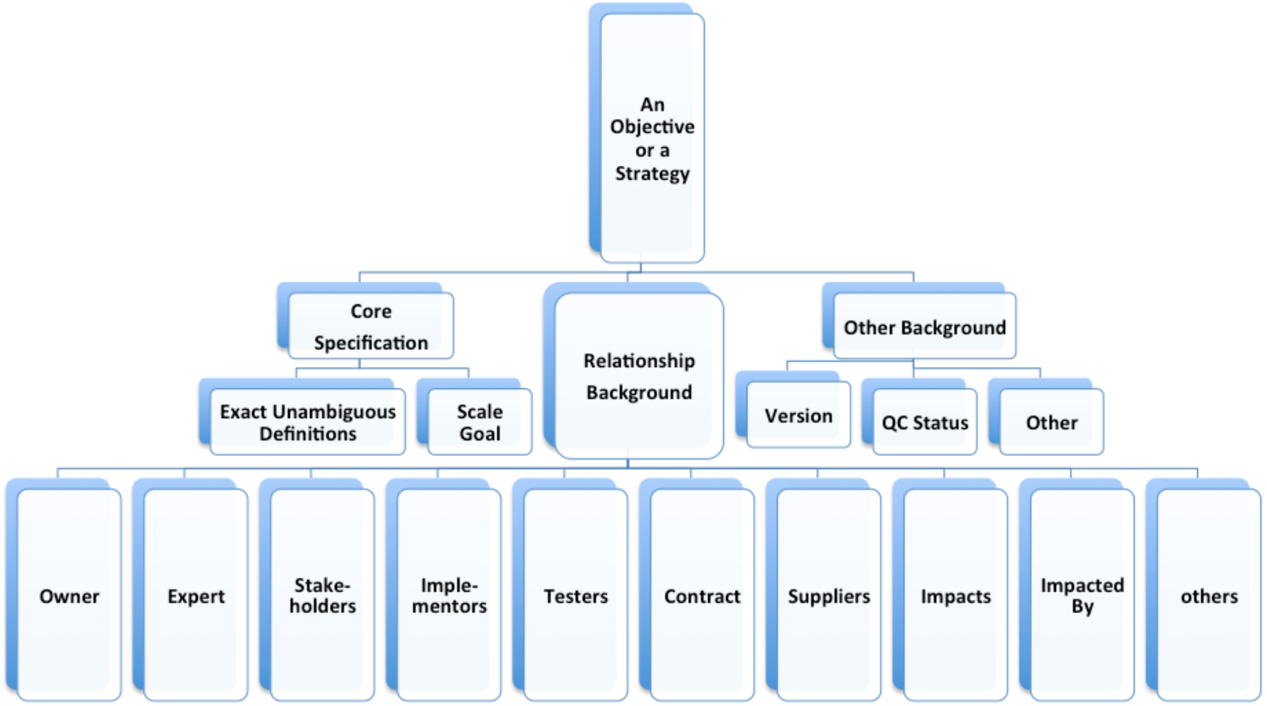
Problem statements can easily be integrated with each other, and their relationship to solutions (strategies, designs, architectures) can easily and automatically be mapped and tracked.

Problem statements can be directly and measurably related to emerging value delivery, and costs: giving real time feedback to the planning model.

A rich set of *background* specifications is expected and encouraged for all problem statements. These include such items as issues, assumptions, constraints, sources, evidence, risks, stakeholders, and very much more [VP 2.2, 3.1, 4.2 for detail].

The *background* specification for a problem statement (like an 'objective') does not change the *core problem specification*. But it enables us to sense the larger and more complex relationships involved (for example multiple *risks* and *stakeholders* for every single problem statement).

Background specification triggers and motivates us to analyze deeper and improve our view or model of the problem space.



W2. Wicked problems have no stopping rule.

Planguage makes no assumptions about stopping a development, or the existence of a ‘final state’.

Planguage assumes that the systems it is planning for, already have a life in the real world, and will continue to have a life for the foreseeable future.

Planguage is all about high-priority incremental improvement, towards the current long-term objectives, using resources actually available.

The Planguage planning process is merely a tool for keeping track of concerns, and solution ideas.

The tool is used to keep track of the current state of the system, from any interesting, and all useful emerging, multiple viewpoints.

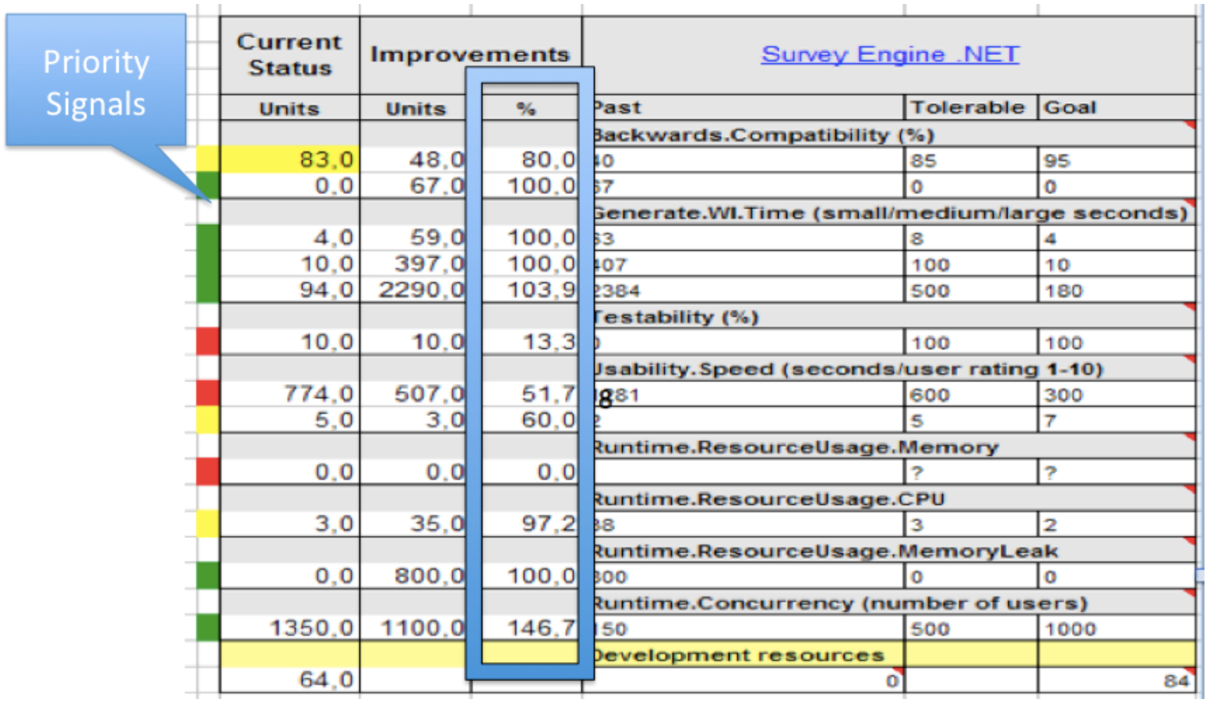
Planguage assumes conflict and change are normal, and natural: and tries to make the best decisions in that light.

The nearest thing we have to ‘stopping rules’ (knowing when to quit planning, or investing in change) are locally formulated policies, such as:

Stop when the next cycle of change is not profitable enough (VP 4.7, 5.9). Stop when no credible solutions are on the table. (VP 4.9, 4.8, 5.8)

Stop when planned results have repeatedly not been delivered (VP 4.5).

W E Deming taught me that the Plan Do Study Act cycle (PDSA) was expected to continue, ‘as long as there is competition’. We do not think in terms of any ‘big stop’: just focus on smart prioritization.



	Current Status	Improvements		Survey Engine .NET		
	Units	Units	%	Past	Tolerable	Goal
				Backwards.Compatibility (%)		
🟨	83,0	48,0	80,0	40	85	95
🟩	0,0	67,0	100,0	67	0	0
				Generate.WI.Time (small/medium/large seconds)		
🟩	4,0	59,0	100,0	63	8	4
🟩	10,0	397,0	100,0	407	100	10
🟩	94,0	2290,0	103,9	2384	500	180
				Testability (%)		
🟥	10,0	10,0	13,3	0	100	100
				Usability.Speed (seconds/user rating 1-10)		
🟥	774,0	507,0	51,7	1881	600	300
🟨	5,0	3,0	60,0	2	5	7
				Runtime.ResourceUsage.Memory		
🟥	0,0	0,0	0,0		?	?
				Runtime.ResourceUsage.CPU		
🟨	3,0	35,0	97,2	38	3	2
				Runtime.ResourceUsage.MemoryLeak		
🟩	0,0	800,0	100,0	800	0	0
				Runtime.Concurrency (number of users)		
🟩	1350,0	1100,0	146,7	150	500	1000
				Development resources		
	64,0			0		84

Planguage is unusually quantified regarding problem statements (VP Part 1, Chapter 1). All values and qualities are normally quantified. No management BS allowed [10].

So the quantified worst-case levels,and target levels of a desired value, give us a very specific device to know when to stop: when to stop planning, when to stop inventing, when to stop delivering improvements, when to stop and NOT deliver changes at all.

Planguage has a rich variety of tools and specifications for stopping, when that is appropriate. For a rich variety of reasons and conditions. But it has a ‘lust for life’ to try to keep delivering value to stakeholders. And it makes that possible by quickly stopping low-priority activity. (VP Chapter 6).

Your own culture needs to decide on your own values and priorities regarding when to stop and go.

Planguage has rich built-in specifications that even automatically point out red lights and green lights. Planguage specifications can compute what to prioritize (Green) and what to stop (Red Light). (VP 6.7)

W3. Solutions to wicked problems are not true-or-false, but good-or-bad.

Planguage has no preconceived notion that solutions are ‘correct or good’ or not.

It *explicitly* recognizes that:

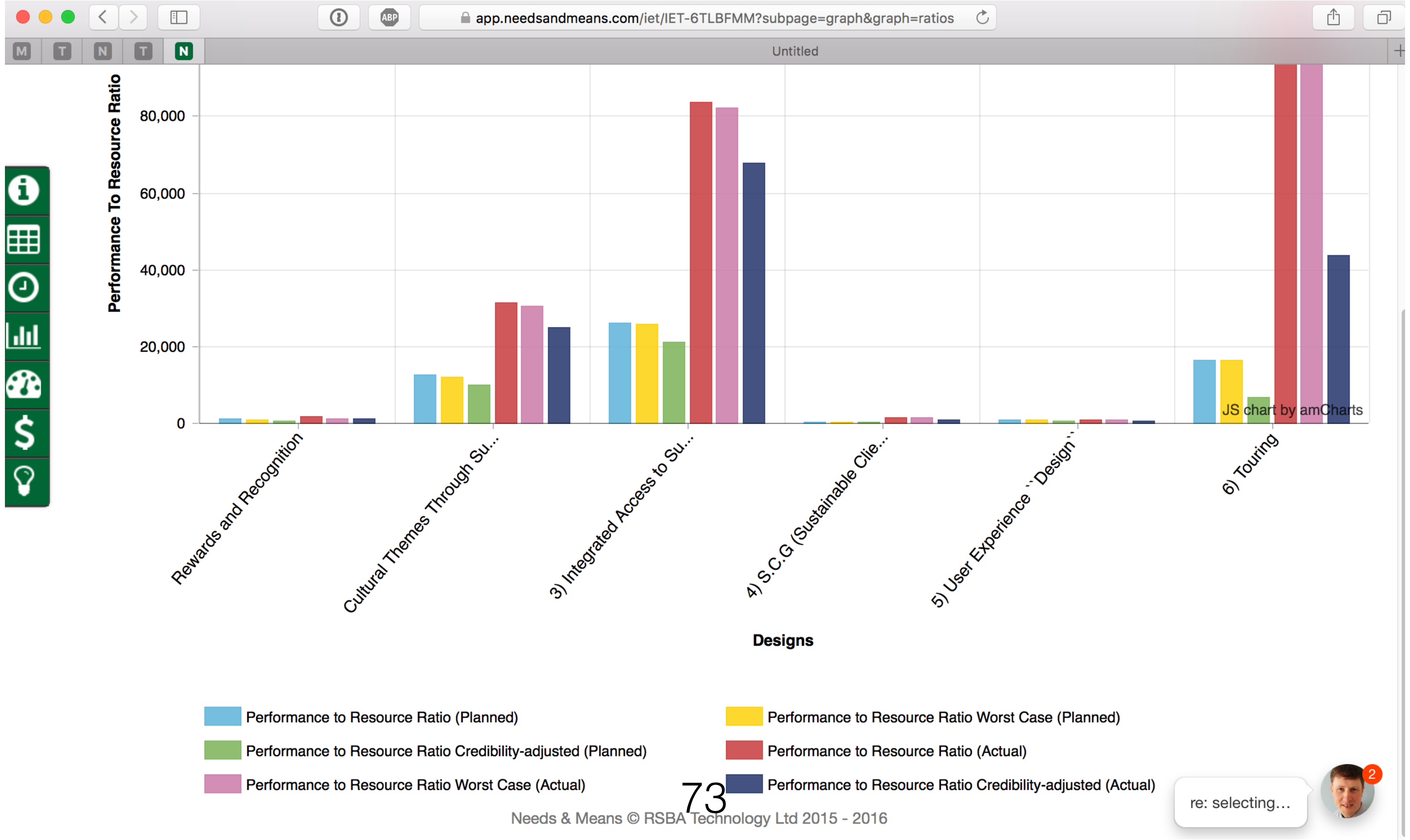
- **Problems** (objectives, constraints) are the best currently available subjective stakeholder compromise.

- Problem specification is subject to constant change pressure.

- **Solutions** (strategies, architecture) are the best available ‘hypothesis’ as to how to solve a set of problems (‘solve’ = delivery sufficient [Target level] overall value delivery, within constraints, at lowest budgeted resource costs; with regard to risk-of-deviation from expectations (estimates). The degree of solution ‘goodness’ is directly related to the current problem specification.

- The degree of goodness is numerically computed in the Planguage tool ‘Impact Estimation table” (sample IET in above VP [8] Figure 6.7). This can be supported by automation as in the example below.

I conclude that **Planguage is well suited to this ‘good or bad’ aspect of Wicked Problems.**


app.needsandmeans.com/iet/IET-6TLBFMM?subpage=graph&graph=ratios
M
T
N
T
N
Untitled
Performance To Resource Ratio
80,000
60,000
40,000
20,000
0
JS chart by amCharts
Rewards and Recognition
Cultural Themes Through Su...
3) Integrated Access to Su...
4) S.C.G (Sustainable Clie...
5) User Experience ``Design``
6) Touring
Designs
🟦 Performance to Resource Ratio (Planned)
🟨 Performance to Resource Ratio Worst Case (Planned)
🟩 Performance to Resource Ratio Credibility-adjusted (Planned)
🟥 Performance to Resource Ratio (Actual)
🟪 Performance to Resource Ratio Worst Case (Actual)
🟦 Performance to Resource Ratio Credibility-adjusted (Actual)
re: selecting…
2
Needs & Means © RSBA Technology Ltd 2015 - 2016

W4. There is no immediate and no ultimate test of a solution to a wicked problem.

This problem is not particular for Wicked Problems. It applies to all problems, all efforts, all changes. Butterfly Effect: the sensitive dependence on initial conditions in which a small change in one state of a deterministic nonlinear system can result in large differences in a later state. It is tough to make predictions, especially about the future (Yogi Berra et al).

In Planguage, using the Project management subset 'Evo' Value Delivery [7], we do in fact measure, in the short term (typically weekly, as in Confirmit example above [Figure W2], the impacts on all the critical factors that interest us.

In this Confirmit case, we estimate and later measure on a set of critical top level Performance Values, and we estimate and later measure time and effort needed to do the change.[See VP [8] Section 8.6 Getting early short-term feedback.]

Later, for example at quarterly release, addition measurements are made. This takes into account the changes made after the earlier changes. It accounts for the parallel changes made by other teams. it typically is more sophisticated testing and measurement (pre release to world market).

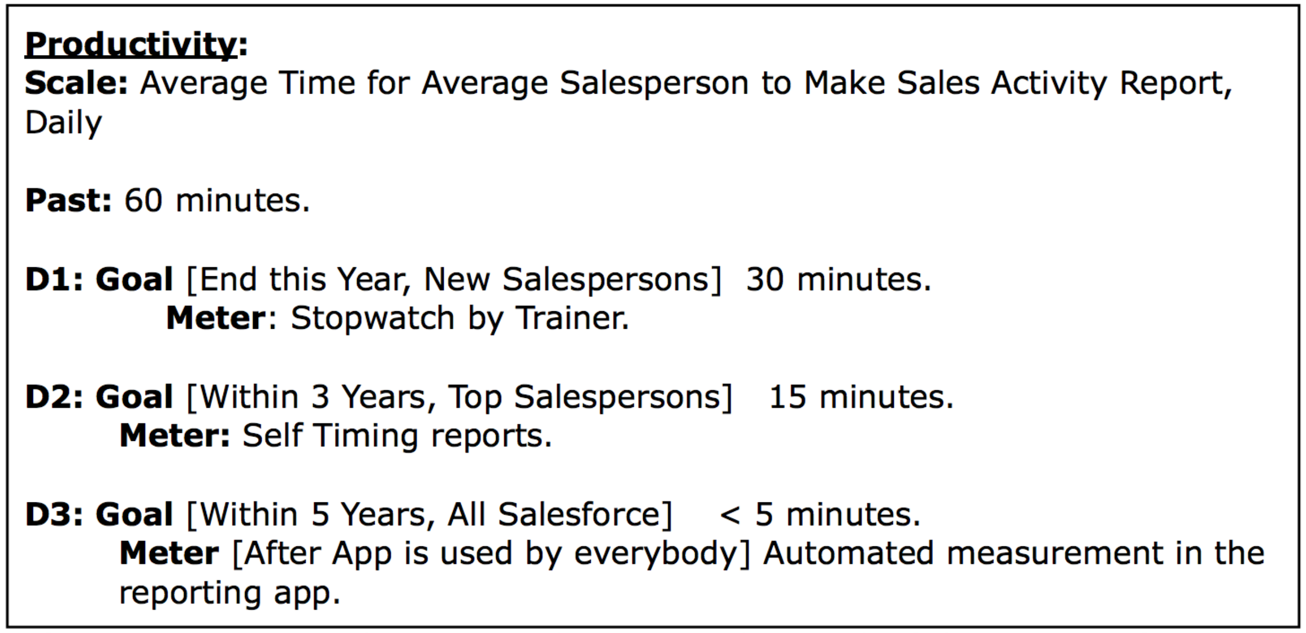
After a release of changes we can *continue* to measure the factors of interest, as they affect real world users of a system. We can certainly expect feedback if they are unhappy!

Finally, in the next round of changes, the critical performance values will *again* be measured, as demonstration that they have held up, or not, over time.

We do not need 'ultimate tests in infinite time'. We need to keep reasonable track of reality in a cost effective manner, and Planguage [Evo] gives us rich numeric opportunity to do so.

All critical problems (of improvement) are always *quantified* in Planguage, or at least 'testable' for presence: that is the basic idea of Planguage.

Reasonable and sufficient measurement and testing is invariably possible.



Productivity:
Scale: Average Time for Average Salesperson to Make Sales Activity Report, Daily

Past: 60 minutes.

D1: Goal [End this Year, New Salespersons] 30 minutes.
Meter: Stopwatch by Trainer.

D2: Goal [Within 3 Years, Top Salespersons] 15 minutes.
Meter: Self Timing reports.

D3: Goal [Within 5 Years, All Salesforce] < 5 minutes.
Meter [After App is used by everybody] Automated measurement in the reporting app.

***Figure W4**. Source VP Planguage 6.7: here is a simple example of planning a set of Meters. Meters are a process for measuring in practice, along a single defined Scale of measure. The Meter statement is usually a rough outline only. The detailed 'test' planning to be done by others, such as system testers. Notice we are dealing with short term and long term measurements here at the same planning specification.*

W4 (continued) I believe one central reason that ‘Wicked Problems’ ***appear* to be** so wicked, is because we have such a ***poor culture of quantification*** of critical factors.

Words and ‘poetry’ (‘state of the art competitiveness’, ‘end world hunger’) substitute for clear thinking and clear problem specification.

This quantification, and background clarification, does not itself, and alone solve the problem central to Wicked problems (the very complex and voluminous nature of real systems).

But lack of quantification of critical system performance problems makes even short-term and real- time understanding of the problem impossible. But that is NOT a Wicked Problem; it is simply our professional incompetence.

We then *falsely* blame our lack-of-understanding on the ‘system complexity’: when we in fact have *not even taken very basic steps* to clear the fog in front of our faces (to quantify critical variables).

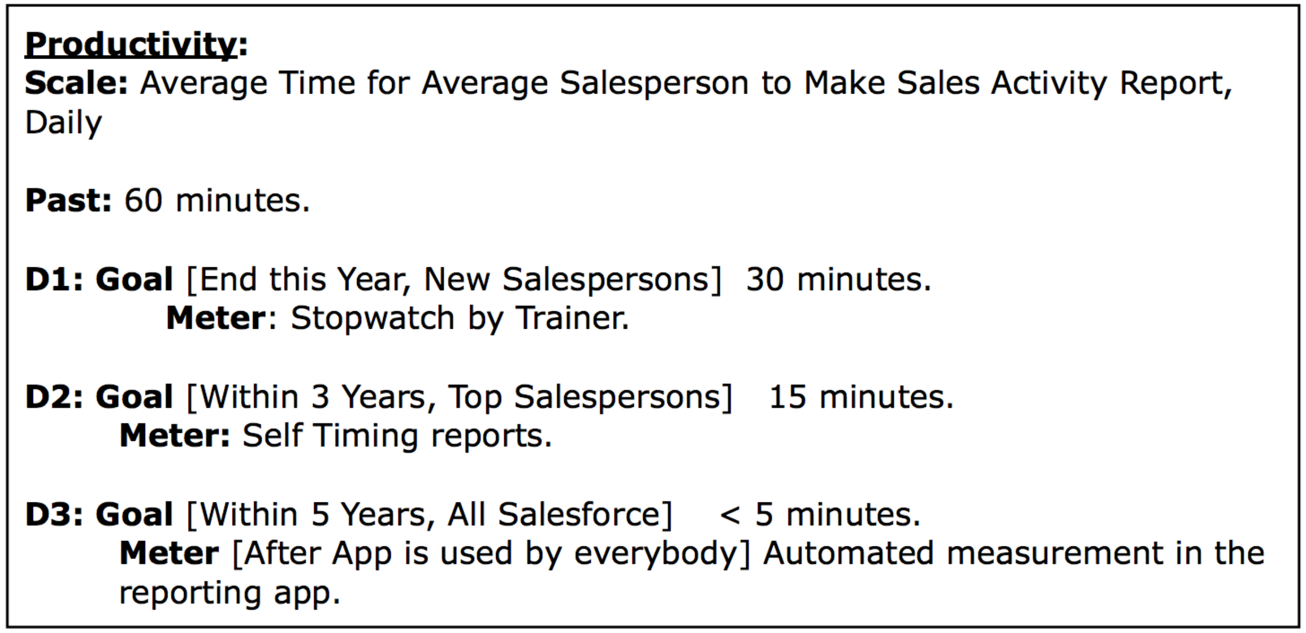
In conclusion:

1. **we can normally get immediate and continuous, tests and measurements, of solutions, in**

   relation to clear problem statements, if we want them.

2. **we do not need to worry about *unrealistic ideas* like ‘ultimate test’ of a solution.**

‘Ultimate tests’ would be nice, of course, but they are not necessary, and they are never possible in the real world.



<u>Productivity</u>:
Scale: Average Time for Average Salesperson to Make Sales Activity Report, Daily

Past: 60 minutes.

D1: Goal [End this Year, New Salespersons] 30 minutes.
Meter: Stopwatch by Trainer.

D2: Goal [Within 3 Years, Top Salespersons]   15 minutes.
Meter: Self Timing reports.

D3: Goal [Within 5 Years, All Salesforce]   < 5 minutes.
Meter [After App is used by everybody] Automated measurement in the reporting app.

***Figure W4**. Source VP Planguage 6.7: here is a simple example of planning a set of Meters. Meters are a process for measuring in practice, along a single defined Scale of measure. The Meter statement is usually a rough outline only. The detailed ‘test’ planning to be done by others, such as system testers. Notice we are dealing with short term and long term measurements here at the same planning specification.*

W5. Every solution to a wicked problem is a "one-shot operation"; because there is no opportunity to learn by trial-and-error, every attempt counts significantly.

This is another example of a possibly 'artificial' problem which is not inevitably inherent in complex systems.

It might be, but another possibility is that the planner has simply not learned to decompose 'big strategies' into smaller, deliverable and possibly retractable 'experiments.

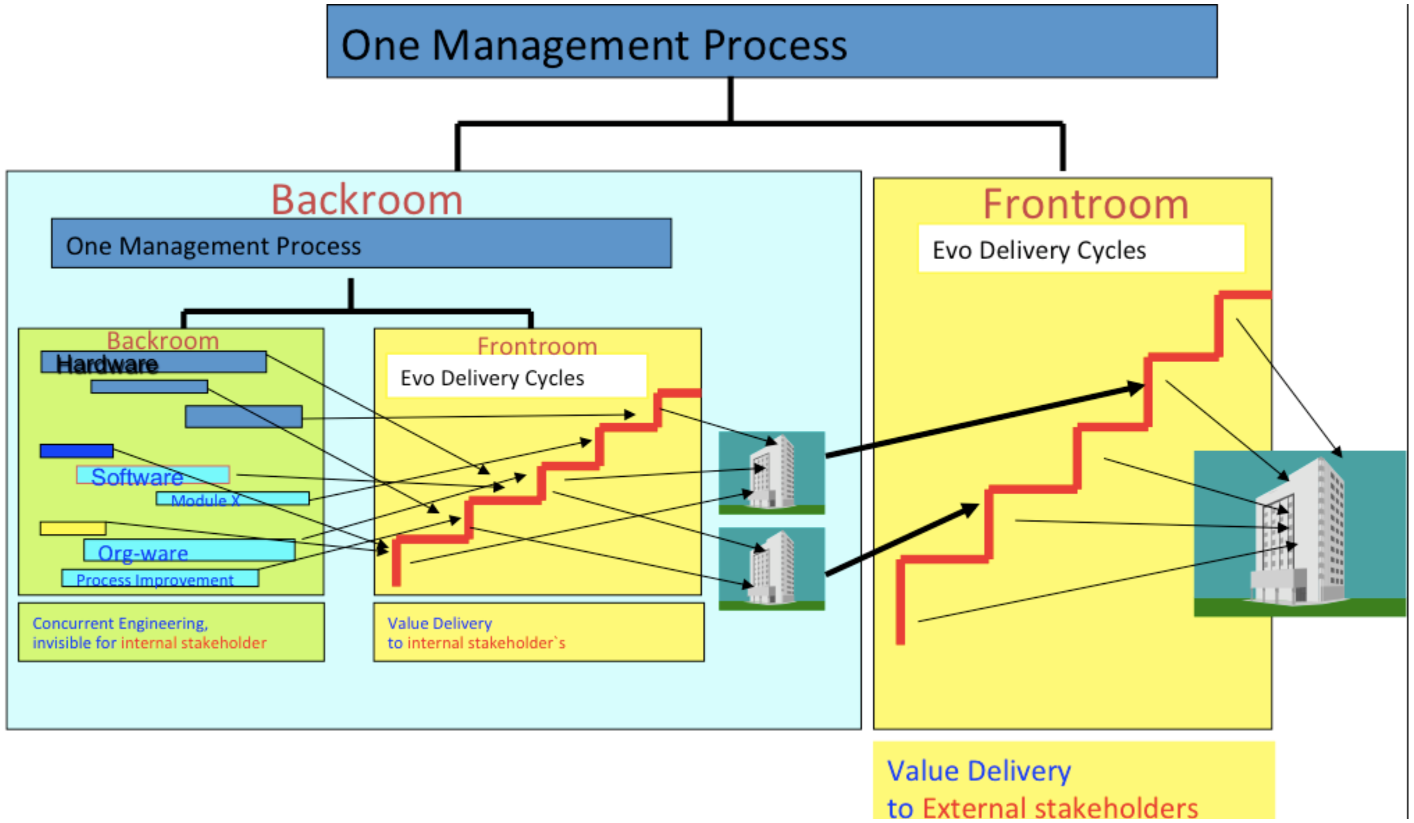
I view this widespread inability do decompose big strategies as 'professional incompetence'. The incompetence is caused by lack of knowledge, and training, in decomposition.

Notice that decomposing solutions into simple experimental components is fundamental to both scientific experiment and to engineering. And there are some very big hairy problems they tackle. Think 'Space' and 'Universe'.

Planguage tries to deal with this problem of decomposition, at length, with constructive and teachable methods. [8, Evo. and VP Chapter 5. Decomposition (by value, by responsibility) page 363 to p. 415].

Imagination, intelligence, experience, motivation will allow professionals to figure out how to decompose. I had to learn it by practical experience over decades. But most professionals have not learned such methods explicitly. Half of them are in illogical denial (it 'cannot' be composed). So it is time to teach the methods, rather than hope people will figure it out in a few decades, personally.

I conclude that some problems appear more 'Wicked' than they really are, because people are not trained in decomposition methods, which would allow us to avoid the 'every attempt counts significantly' problem.



W6. Wicked problems do not have an enumerable (or an exhaustively describable) set of potential solutions, nor is there a well-described set of permissible operations that may be incorporated into the plan.

Well, with this point of view, absolutely all real life problems, about people, culture and technology are 'Wicked'. Again this is an *unnecessary* and *unrealistic* expectation ('exhaustively describable') to real world problem solving.

Unrealistic and Unnecessary:

Even in chess, where the solution space is theoretically exhaustively describable using a computer, there is a time limitation, and even a constraint n real players not using computers in real play. There is too often far too many combinations of play. And this is irrelevant, as long as you either win, or sometimes 'draw'. You do not need all possible solutions, you need a 'pretty good' or 'good enough', on time, to meet your deadline (chess clock).

Planguage as a planning tool has a large number of tools to support this concept of 'good enough, on time'. We will explain a few, as a sample of the toolset.

The first concept is what we call a 'scalar constraint'. It is used in problem formulation. For example "The room temperature must be at least 15 degrees C". One Planguage term we use is to call this a Tolerable Level of theValue.

So if at least one potential solution, to the temperature problem are estimated to give us 'at least 15 degrees C', then the solution is theoretically sufficient. There is not need to look for 1,000+ other possible solutions. This logic is built into the Impact Estimation table in Planguage. And you se it in Figure W4 above. If the level delivered is at or above the Tolerable level, we get a 'yellow' light signal. The solution is 'sufficient', to meet minimum requirements.

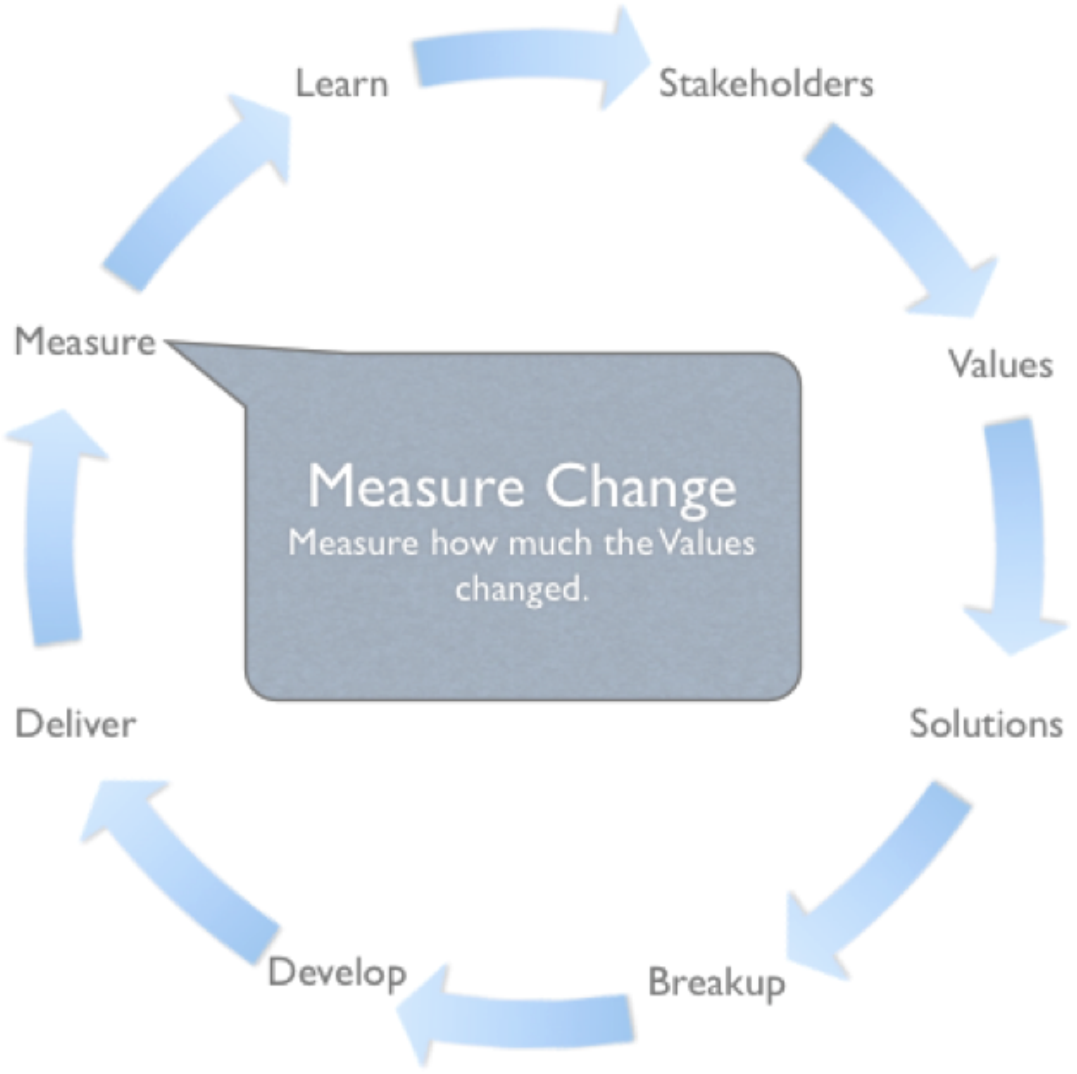
In the next stage of deciding we have enough solutions, we ask in Planguage is the solutions will potentially (later when applied, 'really reach the target levels) reach our Target levels (a formal numeric definition of success and sufficient problem solving). If any set of solutions will reach our success, sufficiency, levels that there is hotpoint is considering the entire solution space exhaustively. That would cost far more than any benefit. it would delay delivery of benefits to the real world in good time. it is silly to even hint that this is 'necessary', to exhaust the solution space at all. Can we use common sense here , please?

of course the above explanation is a simplification, to show the principles involved. Even fairly simple (not especially Wicked) problems require us to think about many other factors, when considering if we have explored the solution space sufficiently. For example costs, interaction between solutions, changes in the stakeholder space, poor implementation of otherwise theoretically good solutions, and much more. I can assure you that these factors are all systematically considered, and we have tools for them built into basic Planguage.

See for yourself. VP Part 1 to 5 (50 pages, free book sample [8,]) and really most detailed in the larger books [8, 7]

Wicked Problems

I conclude that this **'Enumerable Solutions' Wicked Problem characteristic [W6] is artificial, academic theory, of no practical use in the real world. A waste of time to worry about at all. The real problem is finding sufficient for defined purpose solutions.**



W7.1 Every wicked problem is essentially *unique*.

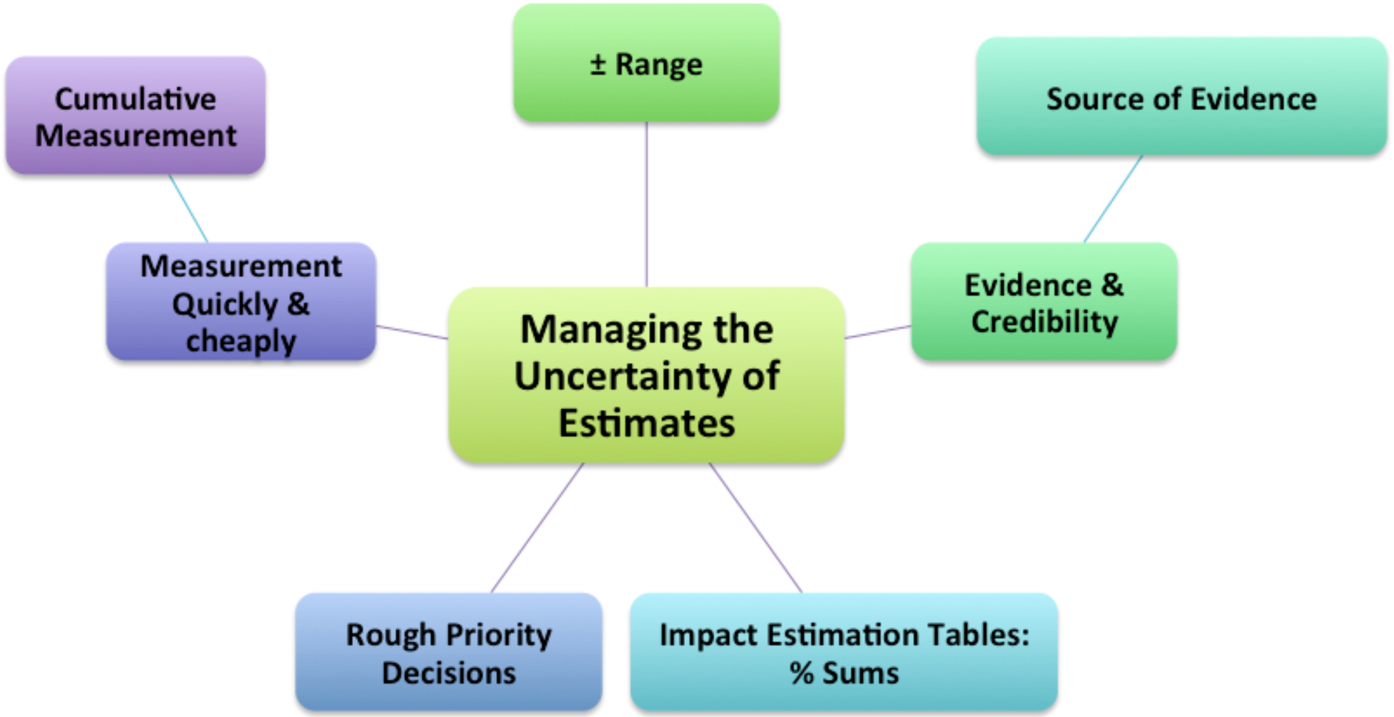
‘Wickedness’: needs an improved definition.

Again, I am getting a bit tired of the fact that these Wickedness characteristics are not especially for ‘Wicked’ problems. Maybe I need to define ‘Wicked’ to my *own* satisfaction?

One immediate thought is that ‘Wickedness’ is *not about the problem itself*. It is the *combination* of ‘the problem’ and ‘the methods-we-know-about; and are willing to use to deal with the problem set’.

And maybe, we need to include, some other factors like resources, constraints and motivations.

The essential ideas are that it is about our real-life current *ability to solve the problem*; not about the problem *itself*. Maybe Wicked projects’, or ‘*Wicked processes*’ (eternal cycles) better capture what we are dealing with here.


Cumulative Measurement
± Range
Source of Evidence
Measurement Quickly & cheaply
Managing the Uncertainty of Estimates
Evidence & Credibility
Rough Priority Decisions
Impact Estimation Tables: % Sums

W7.2 Every wicked problem is essentially *unique*.

‘ Uniqueness is the norm. ‘Identity’ is an impractical ideal.

Let us bring in ‘obvious common sense’ again.

Surely absolutely every problem we humans deal with is in some senses unique.

So what. Absolutely identical problems are not really very interesting. The implication of W7 is that if the problem were identical, we might know the solution. So what?

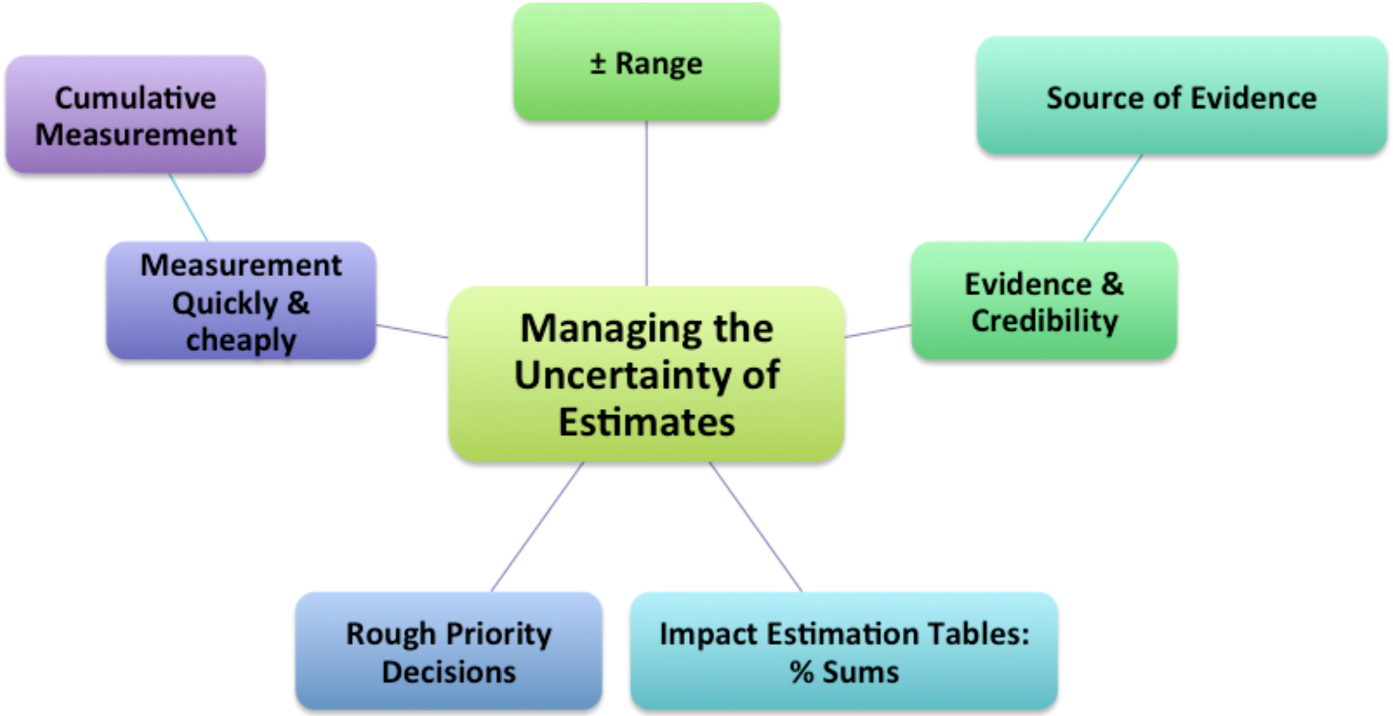
Identical problems, that have already been successfully solved, do not guarantee that the solution used is known, or knowable to us - in *time*. An earlier solution may be secret, hidden, undocumented, or even misunderstood (what the real solution was, as opposed to a publicly documented solution).

Anyway, *nothing* is really identical. And we need to find workable solutions if possible anyway. And it does not really matter if there *was* another known solution that worked once. it may be easier for us to just ‘get something to work’, and move on, than to research, at unknown costs and success of finding it, the ‘real solution used in the past by someone’.

Planguage has no such notion as identical problems, and corresponding solutions. Why waste time asking if there is an ‘identical problem’, anyway. Focus on solving the problem.

Planguage does have well-articulated concepts of asking for *evidence* of past values and costs for any proposed solution [VP Part 2, 4.4]. The solutions and problems are never identical. We know that. But they do not have to be identical. Just *good enough*.

We are not trying to be identical, but we are trying to improve the probability that we will discover, prioritize, use, and measure - *pretty good* solutions quickly.


Cumulative Measurement
± Range
Source of Evidence
Measurement Quickly & cheaply
Managing the Uncertainty of Estimates
Evidence & Credibility
Rough Priority Decisions
Impact Estimation Tables: % Sums

W7.3 Every wicked problem is essentially *unique*.

In conclusion:

‘Unique problems’ is **not a *useful* concept.**

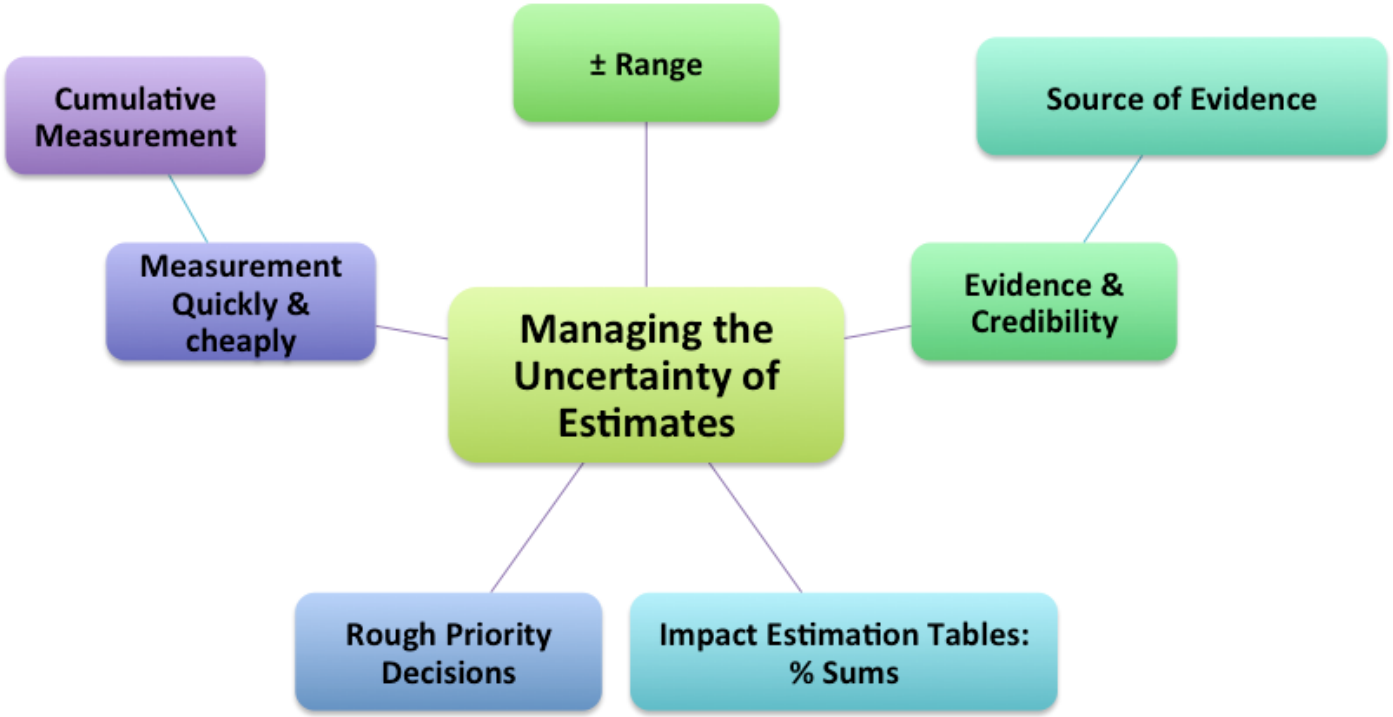
It is not a clear and useful distinguishing characteristic of a problem.

‘Uniqueness of identical problems’ is not a helpful concept.

We need to

focus on finding a solution stream,

cumulating to a useful potential set, of solutions:


± Range
Cumulative Measurement
Source of Evidence
Measurement Quickly & cheaply
Managing the Uncertainty of Estimates
Evidence & Credibility
Rough Priority Decisions
Impact Estimation Tables: % Sums

W8.1 Every wicked problem can be considered to be a symptom of another problem.

Again, we do not need to bring in ‘Wicked’ at all.

Every problem is a symptom of another problem.

That is just the way things are at any level of complexity.

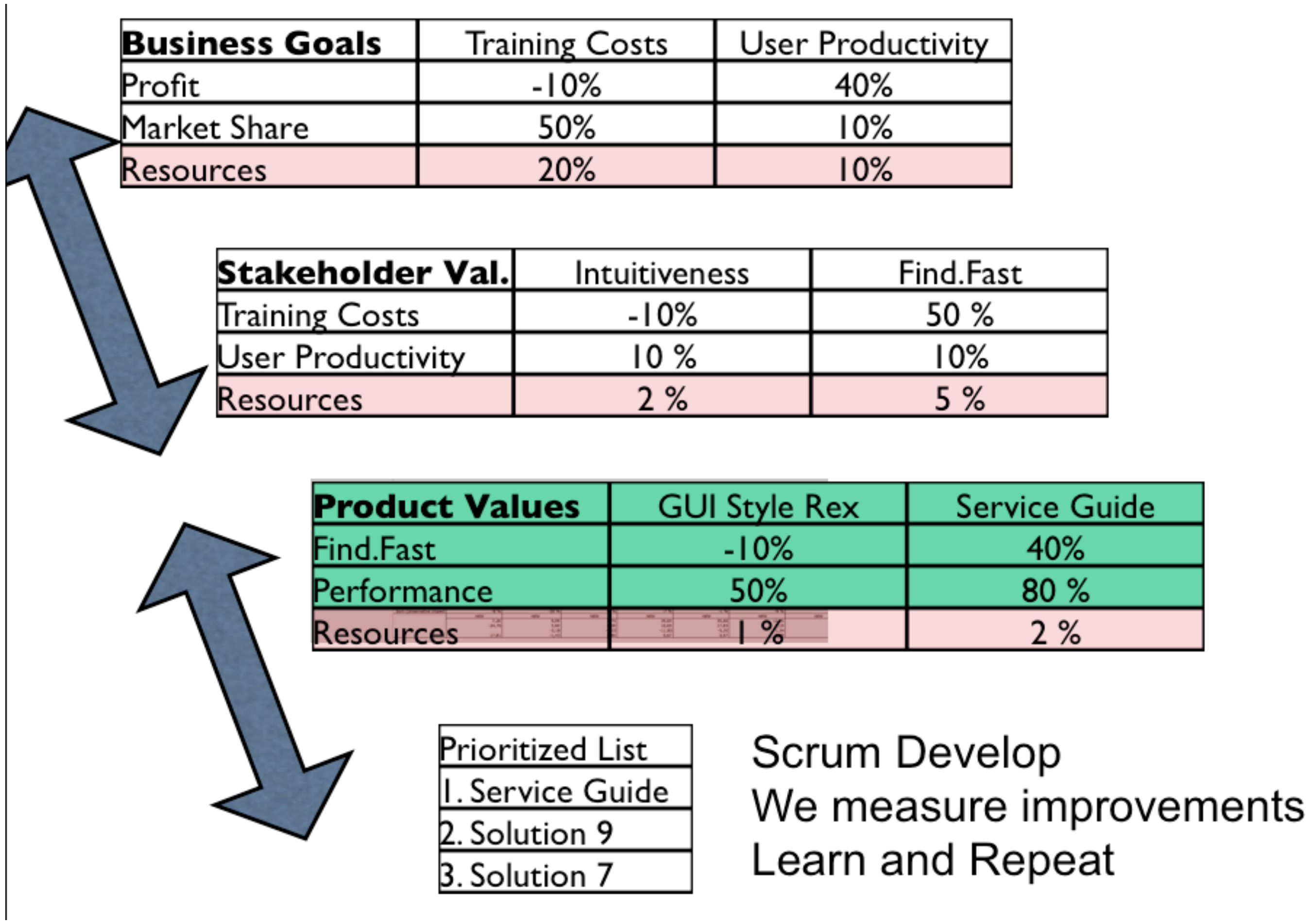
My boss’ solution becomes *my* problem, and my solution becomes my *teams* problem.

Planguage **explicitly acknowledges *many* related levels of concern.**

Stakeholder levels.

Planguage ties these related stakeholder levels *together*, in a variety of ways.

The most interesting method in Planguage is using **Impact Estimation tables to model, quantitatively, the relationships between any set of problems:** any above, any below, or any sideways stakeholder levels.



Business Goals	Training Costs	User Productivity
Profit	-10%	40%
Market Share	50%	10%
Resources	20%	10%

Stakeholder Val.	Intuitiveness	Find.Fast
Training Costs	-10%	50 %
User Productivity	10 %	10%
Resources	2 %	5 %

Product Values	GUI Style Rex	Service Guide
Find.Fast	-10%	40%
Performance	50%	80 %
Resources	1 %	2 %

Prioritized List
1. Service Guide
2. Solution 9
3. Solution 7

Scrum Develop
We measure improvements
Learn and Repeat

W8.2 Every wicked problem can be considered to be a symptom of another problem.

practical application of this was by Kai Gilb at the Transport Company ‘Bring’ [11].

Figure W8. [11] in order to save a large IT Scrum project that failed initially, (the new system drastically killed sales!). Kai modelled the (obviously, ‘it failed’) ‘wicked system’. He built one Impact Estimation Table (aka Value Decision Table) for the top level of the Bring (Norwegian Post Office essentially) organization. This succeeded to resurrect the system, because it mapped the connection between technology and the higher levels of organizational objectives. The IT Development team was then instructed to focus on developing things that led to business (sales!) success. An extremely *simplified* example is above [For more detail see 11].

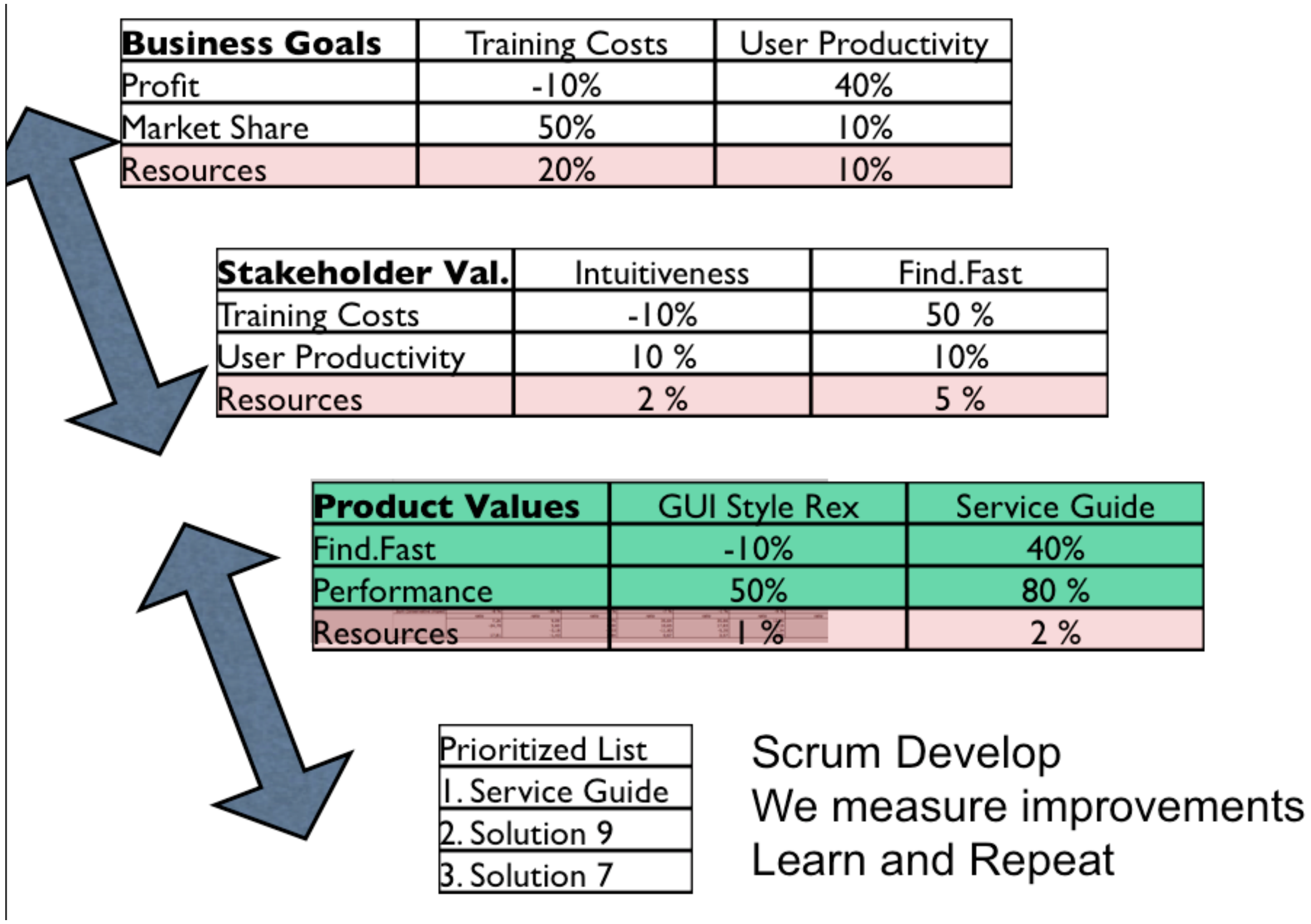
Business Goals: The top management stakeholder level has problems, like *Increase Profit* and *Market Share*. Solutions have been identified (reduce *Training Costs*, and improve *User Productivity*). The expected, estimated, impact of these solutions on the (elsewhere, see Figure W4 for ‘how it looks’) *quantified* Problems, is given by the numbers estimated (later ‘measured as a result) at their intersection. For example Training Costs reduction, if the solution works as expected, promised to move us 50% of the way towards our Market Share objective (the Problem,

Stakeholder Value: These solutions become the the Problem at the next level. The Stakeholder level. Think of these as the 30 or so individual transport companies that had been bought and merged to form Bring. It looks like the Solution named ‘Intuitiveness’ is estimated to contribute 10% of the progress we need towards the User Productivity problem objective. All objectives are, of course, quantified, elsewhere.

Product Val.: At the third level (Product Values), ‘Find.Fast’ (one of the Stakeholder solutions, is considered an IT System objective (a problem statement).

It looks like ‘Service Guide’ is a solution that is expected to contribute 40% towards the ‘Find.Fast’ Problem solution. And ‘Service Guide’ *also* is expected to contribute 80% towards a Performance problem.

Scrum Level: The Service Guide solution will be developed and implemented by the Scrum Team. Hopefully its impact will be approximately as expected, and will impact several levels up towards the Business Goals.



Business Goals	Training Costs	User Productivity
Profit	-10%	40%
Market Share	50%	10%
Resources	20%	10%

Stakeholder Val.	Intuitiveness	Find.Fast
Training Costs	-10%	50 %
User Productivity	10 %	10%
Resources	2 %	5 %

Product Values	GUI Style Rex	Service Guide
Find.Fast	-10%	40%
Performance	50%	80 %
Resources	1 %	2 %

Prioritized List
1. Service Guide
2. Solution 9
3. Solution 7

Scrum Develop
We measure improvements
Learn and Repeat

W9.1 The existence of a discrepancy in representing a wicked problem can be explained in numerous ways. The choice of explanation determines the nature of the problem's resolution.

This is confusingly written up in the literature [1]. Let me try to suggest what it means.

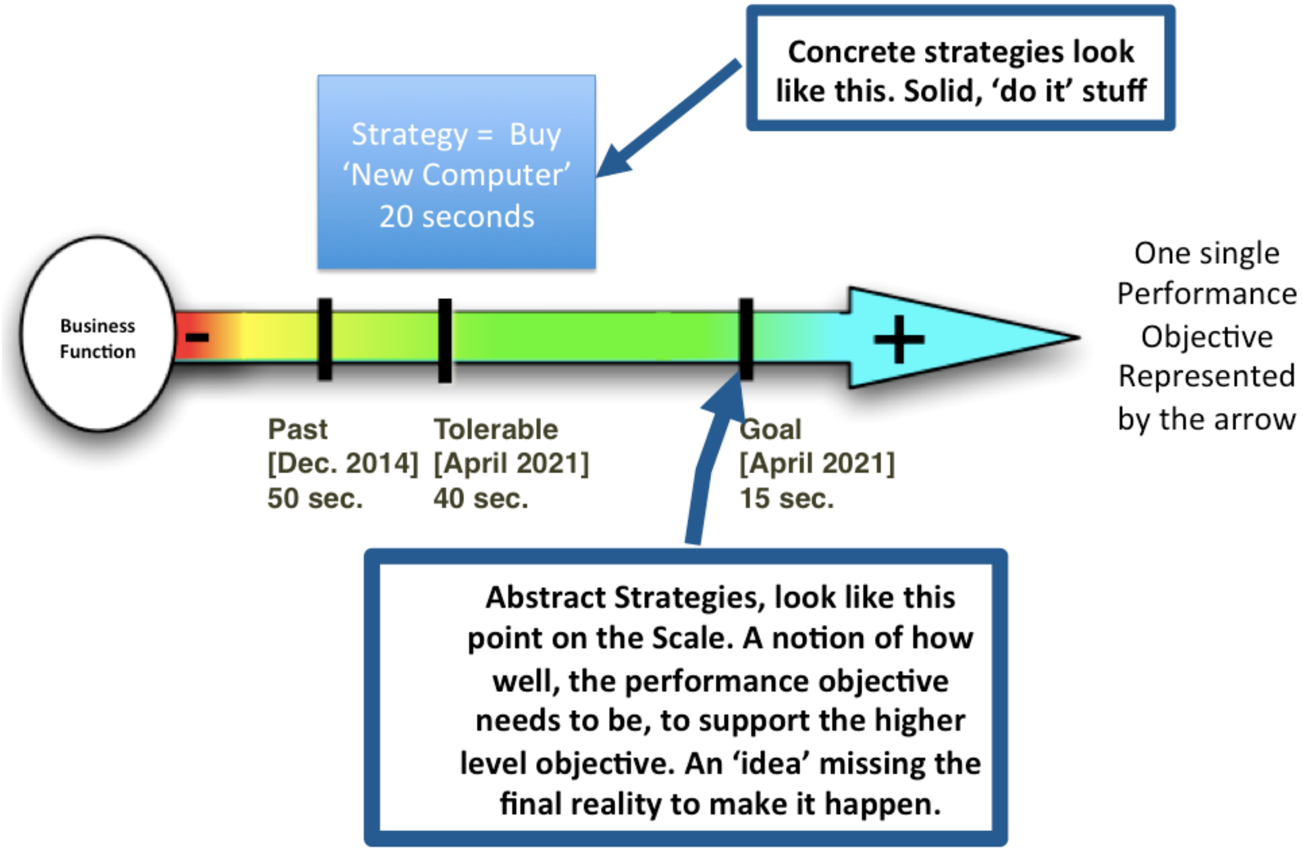
If there is more than one way any people can identify, to solve a problem,

<u>that *alone*</u> allows you to classify the problem as ‘wicked’. (W9 says)

The actual choice of solution, to a Wicked Problem is arbitrary,

and based on the point of view of the planner.

Normal scientific methods of evaluating



Strategy = Buy ‘New Computer’ 20 seconds
Business Function
-
+
Past [Dec. 2014] 50 sec.
Tolerable [April 2021] 40 sec.
Goal [April 2021] 15 sec.
One single Performance Objective Represented by the arrow

Concrete strategies look like this. Solid, ‘do it’ stuff

Abstract Strategies, look like this point on the Scale. A notion of how well, the performance objective needs to be, to support the higher level objective. An ‘idea’ missing the final reality to make it happen.

W9.2 The existence of a discrepancy in representing a wicked problem can be explained in numerous ways. The choice of explanation determines the nature of the problem's resolution.

In Planguage:

1. **any solution that works, delivers value for money, and does not violate any constraints, is 'acceptable'. It does not matter that it is one of many possibilities, or that it is a subjective, comfortable, choice by an arbitrary planner.**

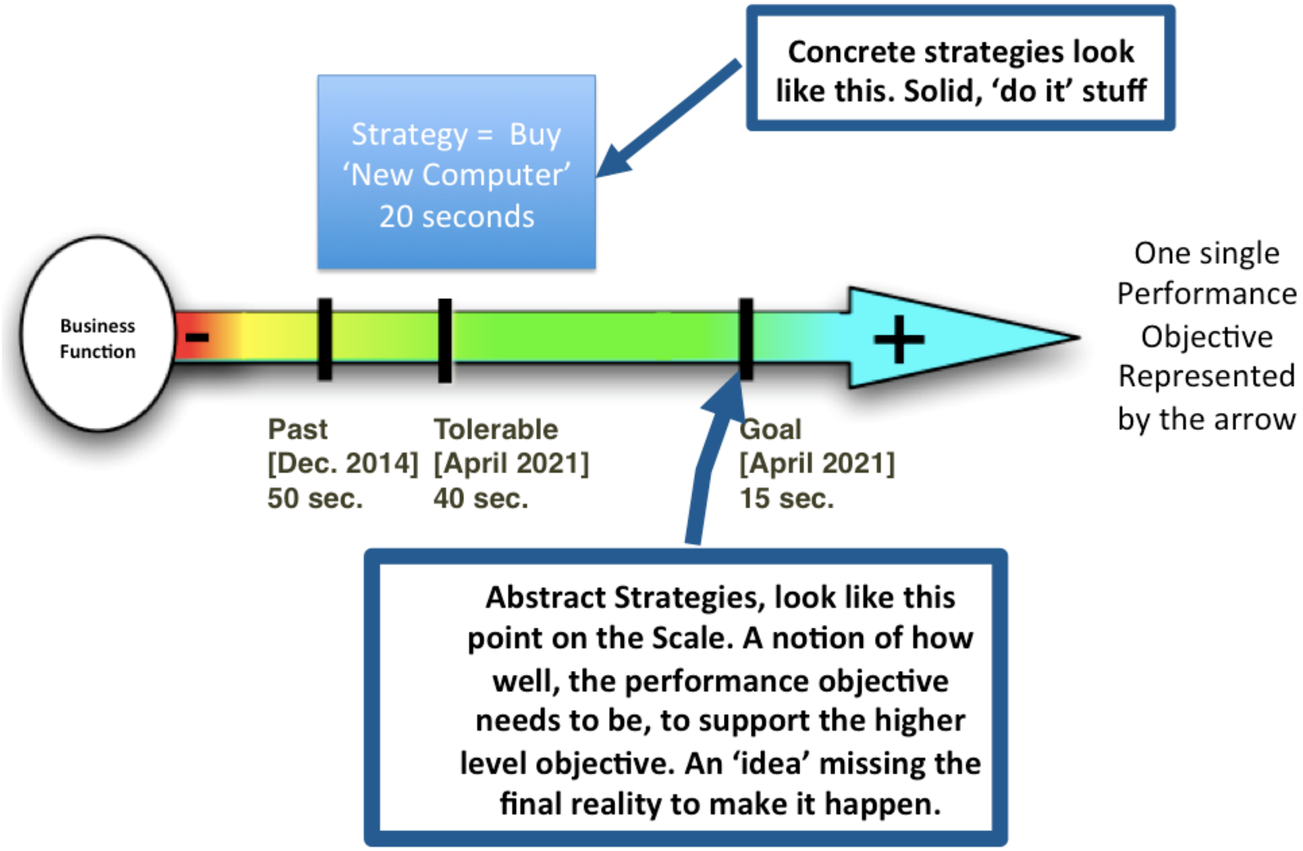
2. **we are happy to document the points of view (stakeholders, sources), and to analyze their 'credibility'. But, if it is legal and it works, we will use it.**

3. **there are many notions of 'priority'. This in clouds value for money, cultural power, riskiness, credibility, and pleasing other people. We can make these priority explicit, or documented and accepted. The important thing is to aware of acceptable and official priorities. And to be able to question and change priorities, because of other priorities [12].**

Conclusion:

This characteristic does not give me any useful insight.

But that could be because I do not understand it yet.



Concrete strategies look like this. Solid, 'do it' stuff

Abstract Strategies, look like this point on the Scale. A notion of how well, the performance objective needs to be, to support the higher level objective. An 'idea' missing the final reality to make it happen.

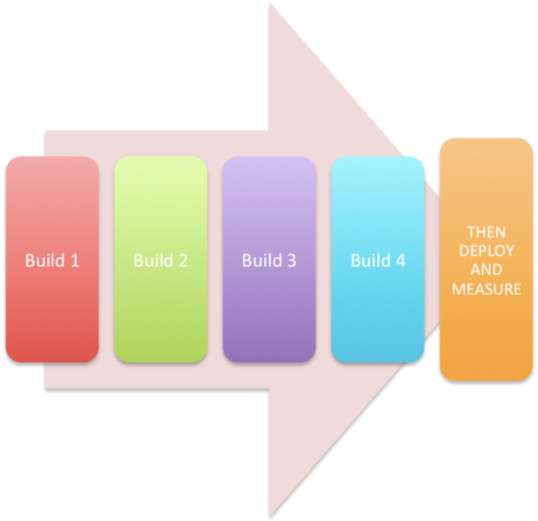
Strategy = Buy 'New Computer' 20 seconds
Business Function
-
+
Past [Dec. 2014] 50 sec.
Tolerable [April 2021] 40 sec.
Goal [April 2021] 15 sec.
One single Performance Objective Represented by the arrow

W10. The ‘planner’ (designer) has no ‘right to be wrong’.

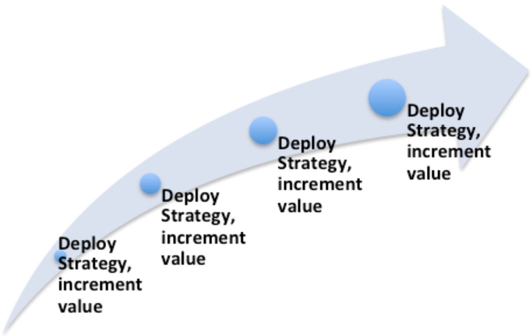
My interpretation, based on [1].

- A *scientist* can live with a wrong hypothesis, if the refutation process leads to greater knowledge and truth.
- A Planner cannot afford the luxury of this scientific process.
- Planning is not about ‘finding the truth’
- Planning is about making thing better for people.
- The planning consequences of a ‘bad’ hypothesis has real, and possibly very negative, impacts on real people.
- So, planners cannot ethically have ‘philosophical fun’ with possibly bad hypothesis.
- They have to get their solution (and problem) right enough to do no damage, and hopefully right enough, to do good, for people.

Not decomposition for this



More like this



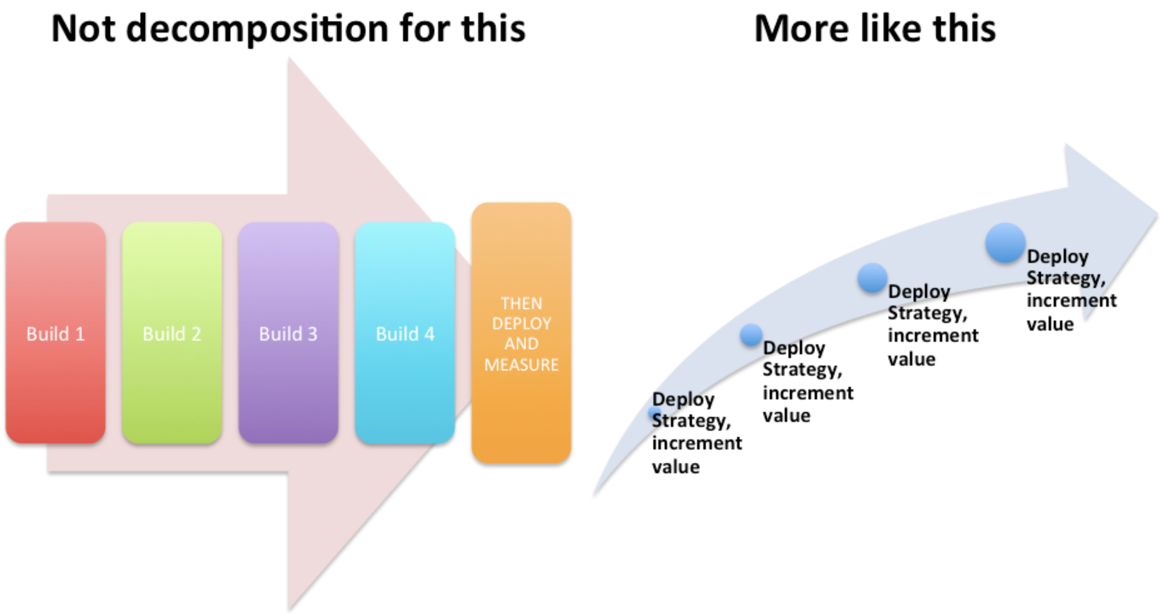
Deploy Strategy, increment value
Deploy Strategy, increment value
Deploy Strategy, increment value
Deploy Strategy, increment value

W10. **The ‘planner’ (designer)**

has no ‘right to be wrong’.

. Planguage very much supports this process, do-gooding, rather than truth-finding.

It does so in a large number of large-and-small tools, principles, methods, and processes.


Not decomposition for this
More like this
Build 1
Build 2
Build 3
Build 4
THEN DEPLOY AND MEASURE
Deploy Strategy, increment value
Deploy Strategy, increment value
Deploy Strategy, increment value
Deploy Strategy, increment value

One of many examples of this is

- **the primary Evo process**

of trying to deliver the largest possible stream of value improvement as early and continuously as possible,

while learning through feedback how to improve on this process itself.

Wicked Ideas Detailed Discussion Summary

I think the 'Wicked Problem" ideas are more misleading than useful.

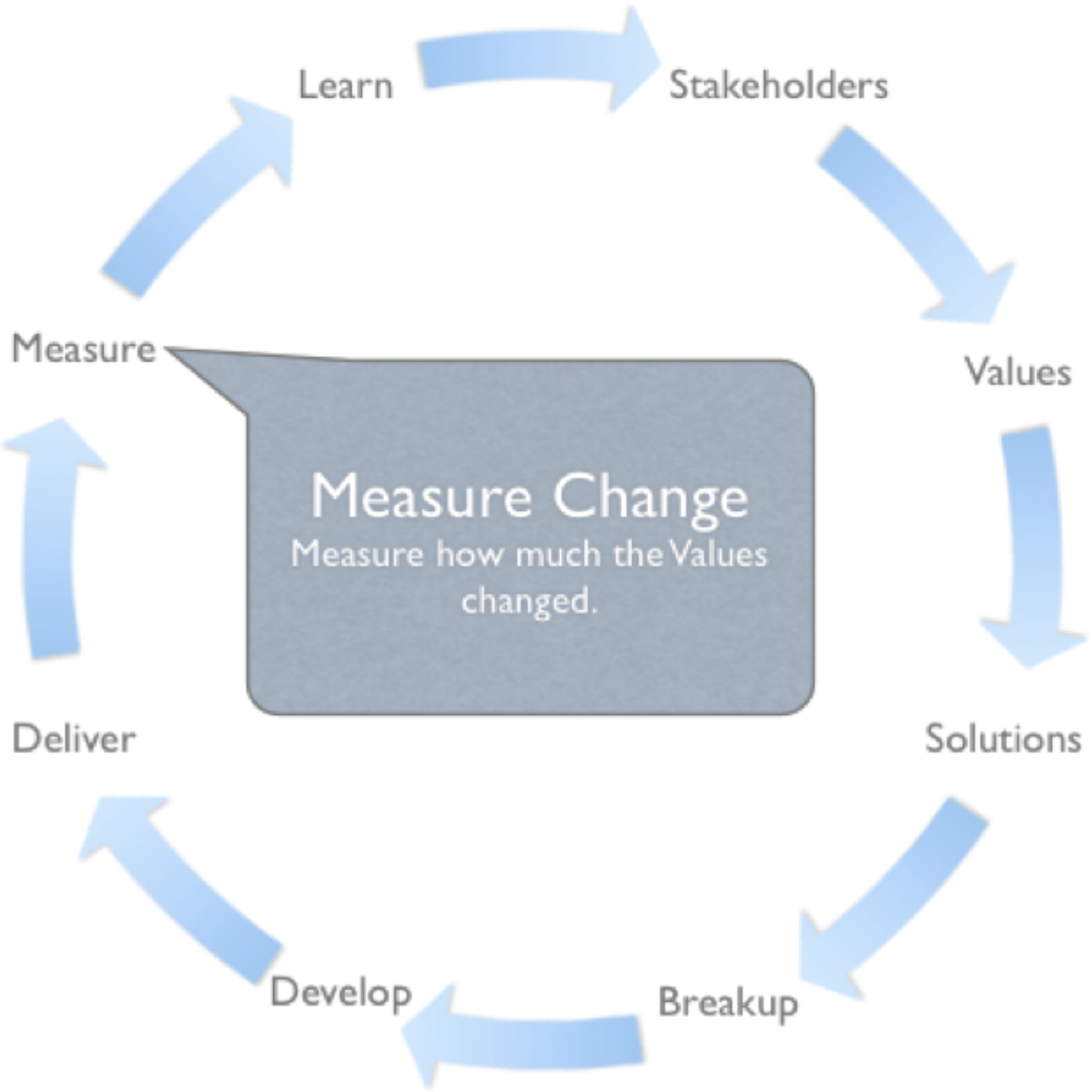
There are a wide variety of methods for handling large and complicated systems in reasonable ways, in addition to the ones I have presented [5 is constructive], here and in my books.

Most intelligent professionals that I encounter, do not seem trained in these methods, and are not aware of the many tools they can use to tackle complicated ('Wicked') systems.

I think we need to focus our attention on mastering a variety of methods for delivering stakeholder value. We are nowhere near good enough, with extremely high failure rates. Failure rates which should shame any professionals with responsibility and pride.

The conditions telling us that we are good enough, or much better are:

- **more than 95% of our projects result in the value improvements we have promised, on time and within budget. We already have the knowledge to do that. Do you ? [F1]**
- **no excuses about 'Wicked Problems'**


Learn
Stakeholders
Values
Solutions
Breakup
Develop
Deliver
Measure
Measure Change
Measure how much the Values changed.