Power Tools to master complex plans and problems

TECHNOSCOPES

By TOM GILB

TECHNOSCOPES

100 Powerful Thinking Tools in Planguage

By **Tom@Gilb.com**

‘Technoscope Tools’: In 5 Parts of 20 Tools Each

1 Requirements
2 Designs
3 Value Tables (Value Decision Process)
4 Spec QC
5 Evo

This booklet will focus on at least 100 ‘tools’, for understanding complex systems and plans. I call them ‘**Technoscopes**’.

Each tool can generally be used stand alone. Pick it up, use it today.

But the tools can be connected together, to give even better understanding, of the products, services, organizations, political decisions, or organizational improvements, that you are working to improve.

The booklet is my personal idea-toolkit, that I have invented, and used, for decades. I want to share these power-tools with you. I would love to know that my ideas helped make you a superior professional! You would not be the first ones! Do please share your experiences!

'Technoscopes’ is more advanced than anything you probably have learned, or used, before, because it focusses on **delivering multiple quantified values and qualities at the same time**. Do let me know if you know about anything better at this!

It also is **logical engineering**, rather than touchy-feely, yellow stickies.
It is about methods that can initially be used on *whiteboards*, but can easily be made into ‘digital tools’.

So, it is ‘too advanced’ for some of us. But it is the *right stuff* for ambitious right-brained techies. As Erik Simmons, Intel, said after years of implementing it at Intel, for 20,000 engineers, “This Stuff Works”. Find out for yourself. Welcome to a ‘more serious’ culture of planning.

March 9-10 2019 Edit

Detailed Structure

Tool Area

Part 1: Requirements

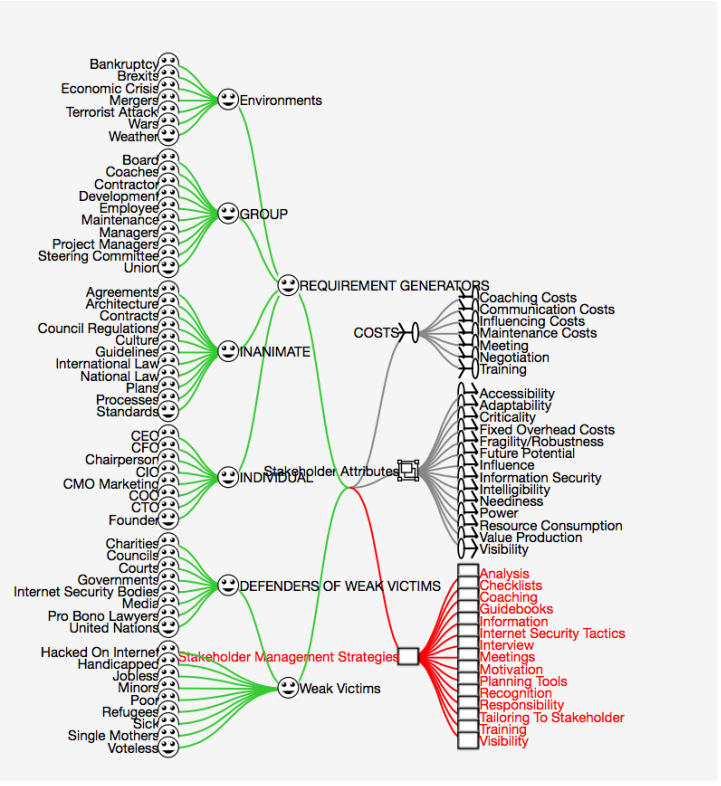
1 Tags
2 Scales
3 Meters
4 Scale Parameters
5 Type
6 Ambition
7 **Benchmarks**
8 Past
9 Trend
10 Ideal and Record
11 Status
12 **Scalar Constraints**
13 Tolerable,Worst Case
14 OK
15 **Targets**
16 Wish
17 Goal
18 Stretch
19 Stakeholder
20 Source <-

1. In this example

a wide variety of planning objects are listed.

They are grouped by type of object,

and they each have a unique reference tag.



1. Tags: The name of any planning object.

- The objectives, stakeholders, and strategies, are all assigned a unique, and long term ‘tag’, for all related planning and delivery processes.
- Tags are in **A**ll **C**apitals, for all words.
- Tags are things that can be used to reference any planning object.
- Tags are unique. They only refer to one thing. They are ‘unambiguous’.
- Tags can be hierarchical: ‘Stakeholders**.**Group**.**Board’
- Tags are essentially a cross-reference device.
- Tags Can Have synonyms: EU**:**Europe
- Tags help us keep track of large quantities of planning objects.
- Tags help us to *reuse* a specification, rather than redefine it anew, elsewhere
- Tags imply that something is *already defined*, or that we *plan* to define it later.
- Tags should be reasonably short. Mnemonic, but just like a person’s name, *not* a full description.
- Tags should not be misleading, but cannot describe most details about a planning object.
- Behind a single tag, we could have dozens of *sub-objects* (like details in a requirement), and each sub-object could be described by a dozen or more attributes (like Deadline, Country, Task, for a Scale of a requirement)
- If we did not have tags, if we had just bullet points, or many different numeric lists, all beginning with 1, 2 , 3 then we would have confusion, and difficulty keeping track of a large plan or model
- Tags are quite usable as digital links: one click and you get the detail.
- I like to emphasize the tag on its definition by **bold** and <u>underlined.</u> **<u>Tom</u>**: Name: Tom Gilb.

Service Availability:
Type: Critical Product Line Objective.
Stakeholders: Service Personnel, Product Users, Consumer Associations.

Ambition: highest service availability in our business.

Scale: % of [**Instances**] for defined [**Users**] for defined [**Products**] and defined [**Problems**] where service can begin within the hour, and be completed within an hour.

Tolerable [Instances = Life Threatening, Users = Medical, Products = Body Monitors, Problems = Total Malfunction] 99.0%

OK [Instances = Life Threatening, Users = Individual Patients at Home, Products = Body Monitors, Problems = Total Malfunction] 99.5%

Goal [Instances = Unable to Modify, Users = Medical, Products = Body Monitors, Problems = Pain or Discomfort] 99.98%.

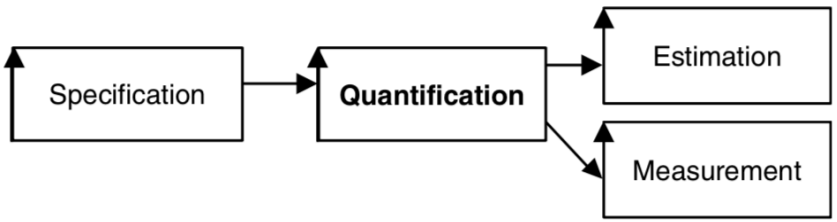
2. THE 'SCALE' PARAMETER DEFINES A PLANNING CONCEPT IN TERMS OF A 'SCALE OF MEASURE', SO THAT WE CAN SPECIFY DIFFERENT TYPES OF NUMERIC LEVELS ON THAT SCALE; FOR EXAMPLE BENCHMARKS, TARGETS (GOAL) AND CONSTRAINTS (TOLERABLE, OK).

2. Scale: a definition of a variable attribute, like a quality, or a cost.

- 'Scale' is a *definition* of something that varies. Like a quality, Value, Performance Level, or Resource.
- **Scale** is perhaps the *most important single idea* in this book, because it moves us from 'lack of clarity' to 'extremely good clarity', regarding the *critical attributes* (values, qualities, costs, constraints) of your system.
- 'Scale' definition moves us from *chat* to *engineering*, from 'misunderstanding' to 'got it'
- Scale definitions can be reused: both many times, in a single requirement specification. And also by reusing a scale definition in different projects, and companies.
- Scales apply to a variety of purposes like: specify requirement, get feedback and measurement, evaluate solution attributes, determine project progress towards goals.
- A Scale can be copied from a library of previously used Scales, or standard Scales
- A Scale can be tailored, or modified, from a Scale Template, or from an earlier-used variation.
- A Scale can have a Tag for Cross reference like: '**Reliability**: Scale: Mean Time Between Failure.'
- A Scale can be found, or at least a template or structure for it found, by internet search (like 'organizational flexibility metrics')
- There are very few standardized scales, for most values, qualities and costs. You have to *expect to tailor* them yourself.
- The Scale can be derived from an 'Ambition Level' statement. (see 'Ambition' below). Scale is more precise, than Ambition, and must always be 'numeric'
- Scales can be enriched considerably by adding '[Scale Parameters]', which give added dimensions (see below)
- Numbers on the Scale, for benchmarks, targets and constraints, automatically and implicitly, refer to the one single Scale, directly above them.
- The Scale can be reused, any useful number of times
- The fact that the scale is *written once*, and *reused*, permits us to write a *rich* and *detailed* scale, without shortening it 'for convenience in later use'. If we do need to explicitly refer to it, we can use a Tag on the Scale. This results in more realistic and accurate modeling of your system world.

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.
Report ST: **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.

3 A. AN EXAMPLE OF METER SPECIFICATION, ALL RELATED TO THE ONE SCALE, BUT TAILORED FOR 3 DIFFERENT TYPES OF GOALS



3B. QUANTIFICATION AND ESTIMATION USES A 'SCALE'. MEASUREMENT IS DONE WITH THE 'METER'

3. Meter: the test or measurement process specification.

- The Meter parameter specifies how we intend to measure ‘which numeric level’, a system currently is at.
- The Meter specifies, or at least refers to, a *process* for measurement.
- A Meter does not have to be specified initially, the Scale is sufficient for us to understand a requirement.
- A Meter spec might be only or initially, a rough suggestion to a test planner, or to a system builder; who will design and detail the measurement process later, when necessary. And even change the actual Meter spec, when necessary, to reduce costs, or to increase credibility.
- A Meter has a large number of quality and cost attributes, which must meet the needs of a given project.
    - For example a Meter spec has these attributes: Accuracy, Consistency, Credibility, Standards Compliance, Setup Time, Cost to Measure, Automation Capability.
- There can be several *different* Meters which can measure along a single defined Scale.
- And we can specify any useful number of different Meters, for different situations.
- Meters can have Tags to reference them, and they can be selected from a library of measuring processes
- Meters are intimately related to a particular Scale. You cannot define a relevant Meter, until the Scale definition is settled.

Scale: the average % defined [**Responses**] from defined [**Customers**] for defined [**Reasons**] per year.

Goal [Deadline = First Release, Market = Asia, **Responses** = Complaints, **Customers** = Long Term, **Reasons** = Product Failure, **If** Product is NOT in Beta] less than 1%

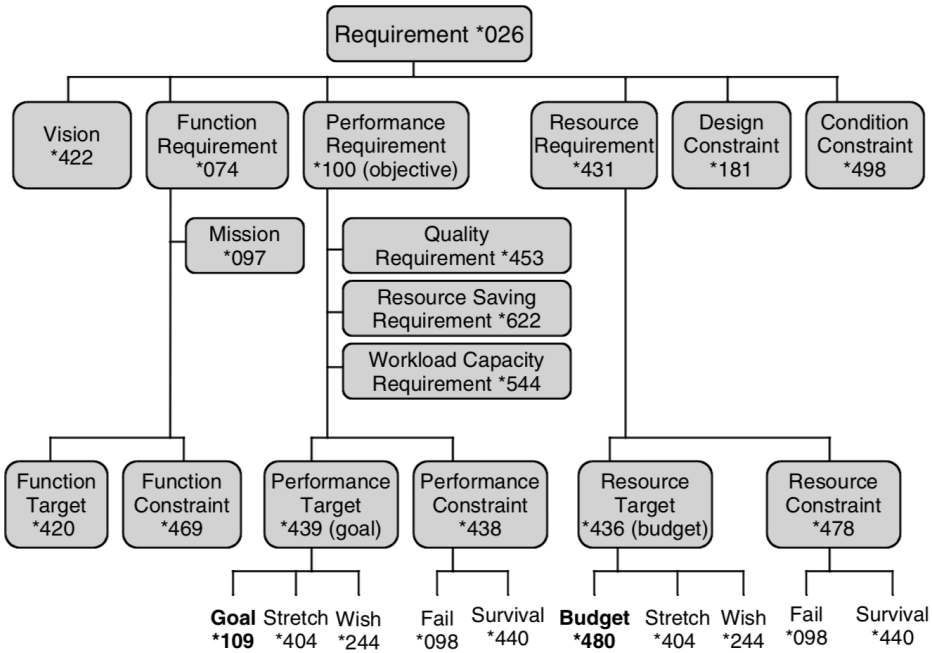
4. AS YOU CAN SEE, WE CAN ADD ANY OTHER [CONDITIONS] OF INTEREST, IN ADDITION TO THE ONES DEFINED IN THE SCALE PARAMETERS. WE ADDED 'DEADLINE', 'MARKET' AND 'IF', IN THIS EXAMPLE.

THIS ALLOWS US TO MODEL OUR REALITY, MORE EXACTLY AND MORE RELEVANTLY.

WE AVOID UNNECESSARY GENERALIZATION. WE GET REALISTIC.

4. Scale Parameters: Dimensions in time, space, activity and conditions.

- Scale Parameters, are tags in [Square Brackets] inside a Scale definition
- They demand explicit and specific definition in any Scale *Level* specification (like 'Goal') below the Scale.
- Additional parameters, not specified in a Scale, may be employed in a Scale Level specification (see example at left)
- The options, in the actual interpretation, of a 'Scale Parameter', can be invented on the fly, as we actually specify Scale Levels
  - Directly in Benchmarks like Past, Targets like Wish, Constraints like Tolerable.
  - Like: **Goal [Deadline : Next Year, City = Stockholm, Task = Report Incident.] 60%**
- The Scale Parameter Options can also be specified as a set, in advance. These can be extended, later, as needed.
  - Examples **[Cities]: London, Oslo, Paris,** **[Tasks]: Setup, Operated, Maintain, Restore**
- A Scale Level Specification can specify 1 or more Scale Parameter Options (as examples above), and in addition, can specify generic groups, such as **All, All Others**, **Any**, or **None.**
  - Example **Goal [Cities = All, Tasks = All Except Restore] 42%**
- The Scale Parameters are useful for avoiding a large number of different variations of Scale definitions. Scale Parameters 'generalize' the scale.
- Scale Parameters help us think about the various dimensions of our problem, that we might want to consider.
- Scale Parameters allow us to specify different deadlines, and different performance levels on the Scale, for essentially different circumstances, like *people and places, and tasks* dimensions.
- Pre-defined sets of Scale parameters allow us to check for 'complete specification', for all valid or interesting combinations of the Scale Parameter dimensions. It enables Quality Control.
- Scale Parameters are a key Technoscope for modelling complex, large-volume, and changing systems, such as a National political decision, or plan. For example Leaving a Treaty, or Improving schools and healthcare.



5. SOME PLANGUAGE ‘TYPES’ OF REQUIREMENT SPECIFICATION, AS FORMALLY DEFINED IN THE PLANGUAGE GLOSSARY [CE BOOK]

5. ‘Type’: the class of specification, which has special or distinguishable needs and forms.

- The specification of ‘Type’ in a planning object, announces to the reader, more-exactly, what they are looking at.
- Each type has specific expected characteristics (like ‘variable’, like ‘a resource’) which make it different from the other Types.
- This saves the reader the effort of asking themselves: **‘what exactly is this specification about generally?’**
- Type enables the reader (and diagnostic apps) to determine what to expect, and what is not valid in the specification
- Type enables a Digital app, or a textbook, to immediately suggest a suitable *template of specification options*, along with some potential extra options.
- Type forces the planner to ‘make up their mind’ about what they are doing: commitment to the idea
  - many people have very different ideas for example, of ‘what are ends’ and ‘what are means’ to the ends.
- Type helps us enforce rules of specification (see Rules, below)
  - Like: the specification Rule **‘all qualities must be defined by at least one Scale’**
- Standard: Type enables standards to be taught, and used in organizations, for many people, in many places, over a longer time period.

Development Capacity:
Version: 3 Sept 20xx 16:26
Type: Main <Compound/Elementary> Objective for a project.

Ambition Level: radically increase the capacity for developers to do defined tasks. <- Tsg

Scale: the Calendar Time for defined [Developers] to Successfully carry out defined [Tasks].

Calendar Time: defined as: full working days within the start to delivery time frame.

Owner: Tim Fxxx.

Past [at 20xx, Site = {Bxx, Lxx, Gxx}, If QA Approved Processes used, Developer = Architect, Task = Draft Architecture] 15 days ±4 ?? <- Rob

Goal [By 20xx, Site = { Bxx, Lxx, Gxx }, If QA Approved Processes used, Developer = Architect, Task = Draft Architecture] 1.5 days ± 0.4 ?? <- Rob

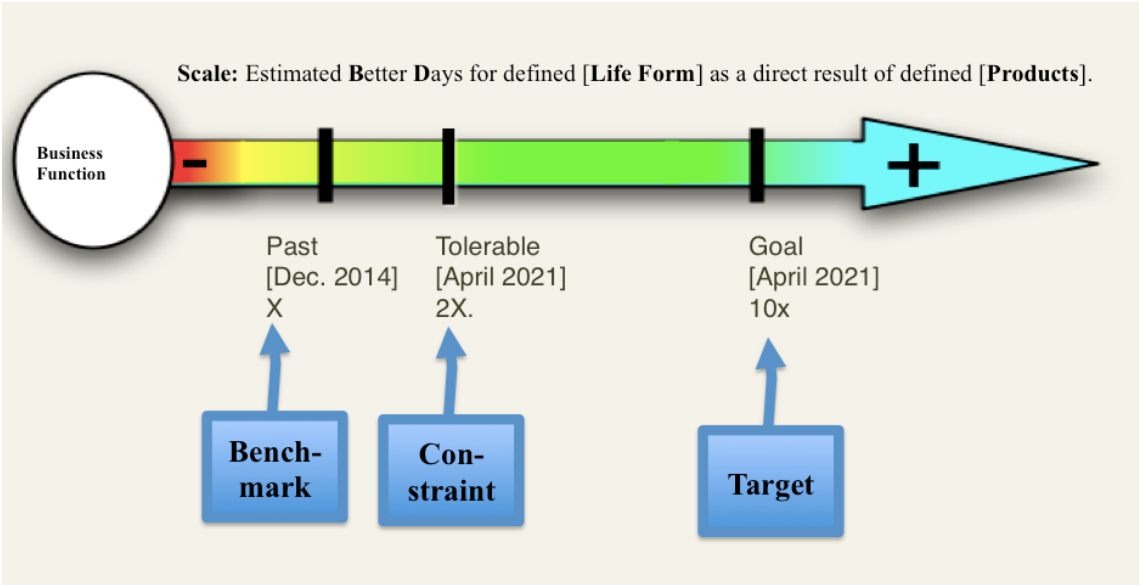
Justification: Really good architects are very scarce, so we need to optimize their use.

Risks: we mis-use effort that should be directed to really high volume, or even more-critical areas (like Main Objective).

6. ACTUAL CAPTURE BY ME, AT A MEETING, IN 20 MINUTES TIME, WITH DEPARTURE POINT, A BADLY WRITTEN OBJECTIVE, 'IMPROVED DEVELOPMENT ENVIRONMENT'. I AM LISTENING IN TO REMARKS OF 4 CLIENT PEOPLE, AND ASKING SOME STRUCTURED PLANGUAGE QUESTIONS. 'XX' = REAL DATA OBLITERATED IN THIS EDIT. <- VP BOOK, PLANGUAGE EXAMPLE 9.2

6. Ambition (Level): the informal, but possibly official improvement objective.

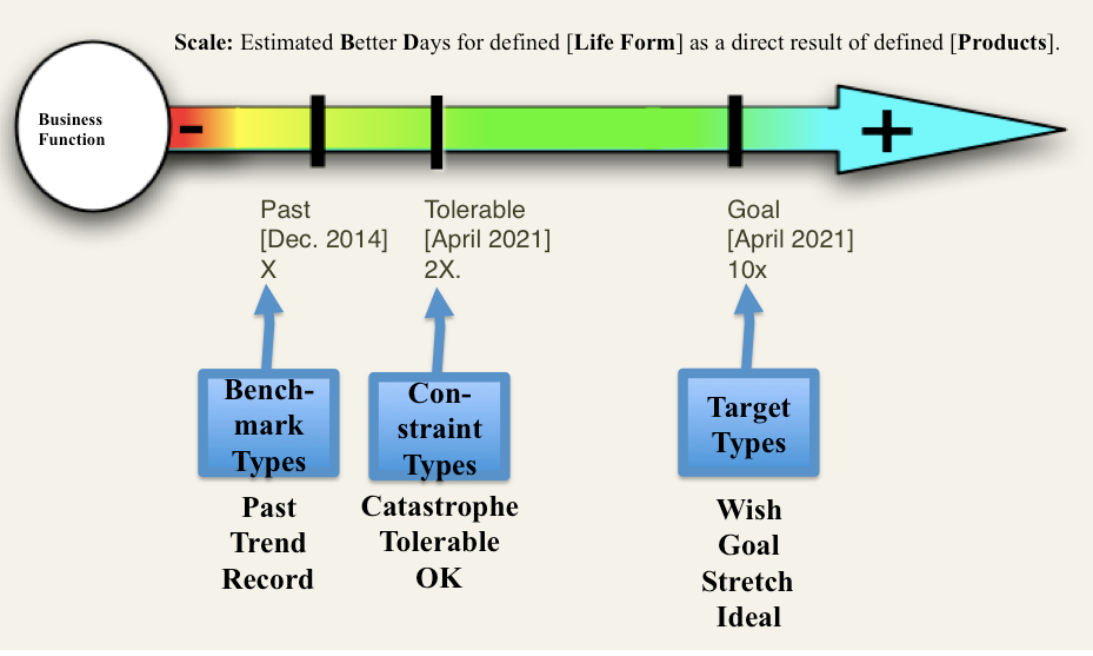
- The Ambition parameter. The Ambition level statement, describes a requirement level, or ‘objective level’, in more detail than the Tag, but in far less detail than a subsequent set of Planguage statements.
- Ambition is typically 5 to 15 words or so.
- Ambition might be an attempt at a *summary, of the value requirement,* by the planners.
- Or Ambition can be viewed as a **Headline** for a many-line specification, so people can decide if they want to look at the detail. Like a newspaper headline, allowing you to decide if you want read the detail.
- Ideally Ambition is a “quote” from some officially named person, or official plan: and our job below it, is to flesh it out in appropriate detail, with better clarity, than a few ‘Ambition’ words can give.
- Ambition must contain some notion of a ‘level’, even if it is vague, like ‘*improved*’, ‘*radically* transformed’
- It is vital that the exact source of the Ambition statement, be documented, even if that is just ‘a planner’
- The source annotation, for Ambition, serves several purposes: *credibility, priority, power, showing respect, alignment, and access to changes*
  - Example:
    - **Ambition**: to be 10 times better in product and service quality than any competitor within a decade <- HP CEO



7. 'BENCHMARKS' ARE CLEAR FACTUAL NUMERIC LEVELS, TO HELP US DECIDE AND UNDERSTAND, PLANNED *FUTURE* LEVELS (CONSTRAINTS AND TARGETS)

7. Benchmarks: numeric levels of 'system performance', past, present and future, which we can use as a basis for planning requirements.

- **Benchmarks** (Past, Status, Trend, Record),
  - points for *comparison* with future plans.
- Benchmarks are 'levels of performance' *worth knowing about,* in comparison with *future* planned levels.
  - For us, for competitors, for the past, and possible future specification.
- *Capturing* benchmarks initially is often thought of as a 'systems analysis' phase.
- Continuous: benchmarks *can* be 'continuously updated', as a project, and a live system, or product progress. They are not merely static history. 'Status', 'Trend' and 'Record' are good examples.
- Dynamic status agile: in Agile (incremental Evolutionary delivery cycles) updating the benchmarks is a necessary condition for management.
- Integration into Plans: in Pre-agile tradition the benchmarks were used for 'early project analysis', and 'to set future improvement objectives'. We need to *integrate* benchmarks, and update them into the *live project plans*, and we need to be 'agile' with reference to changes in competition, technology, economics and politics.
- Benchmarks are useful tools for helping us analyze risks.
  - Like making a plan to 'go for a level no-one on Earth has achieved yet'. (Risky, right?)
- Benchmarks are tools to help us understand instantaneous priorities
  - like 'catch up with a competitor's new levels of quality'.



8. "THOSE WHO DO NOT REMEMBER THE PAST ARE CONDEMNED TO REPEAT IT." (SANTAYANA)

8. Past Benchmarks: are any *estimated*, or *measured*, level for us, or others, that is interesting to compare our future plans to.

- Past is a basic benchmark, and is a historic, static, record of a level of performance
- One use of Past is as a baseline (i.e. a benchmark) to determine *what level we are at currently*, and therefore what other level constitutes an 'improvement', or even a 'degradation' of a quality, value, performance attribute, or even a cost or resource.
- Past (and other benchmarks like Status) can be chosen as a zero base ('0%') for % improvement, in strategy planning (see Value Decision Making below)
- We are consciously trying to motivate you to adopt a serious 'fact-based culture'.
- People who do not have any Past/Status, or have only a 'dubious' measured level, should not go ahead and set targets and constraints for their future.
- They could end up planning to be 'worse than they already are'!

Customer Service Availability:
Type: Systems Analysis.
Scale: % of 24/7 a customer gets a qualified answer without waiting, or without failing, when they use the qualified answer.

Past [Last Year, Our Main Service System] 95% <- Service Report.

Record [Last Year, Our best competitor] 98% <- Their PR.
Record [Worldwide, Last 10 years, Similar Customer Service. Systems to Ours] 99.98% <- Industry Surveys.

Trend [by Next Year, Based on Last 5 years, Our Main Service System] 93% ?

Trend [Next Year, Our best competitor] 99% ??

9. EXAMPLE OF 3 TYPES OF BENCHMARKS. INCLUDING 2 TRENDS. '<-' MEANS 'SOURCE IS'
NOTICE THIS IS NOT *YET* A REQUIREMENT SPECIFICATION
(NO TARGETS OR CONSTRAINTS HERE): PURE SYSTEMS ANALYSIS

9. Trend: an estimation of the levels, good or bad, that will possibly be reached by us, or others, at defined times in the *future*, and under defined circumstances.

- Trend is a ‘future’ benchmark. It is looking out of the front window of a moving car, instead of the rear-view mirror (Past). Or in the case of some cars the video screen rear-view camera.
- The purpose of Trend specifications is to try to estimate, and understand, how things *might become* in the future, with regard to your own systems, or for competitors, or enemy systems.
  - Competitors might be getting better, much faster than you thought
  - Your own systems/products/services might be degrading, much faster than you thought
- By comparing Trend data, to our current improvement plans (levels, dimensions, timing) we can be ‘agile’ and respond faster to priority changes, in our current plans
- Trend is a tool for *risk management.*
- Trend is a tool for helping us *prioritize* agile value-delivery cycles (more later)
- Trend is something your Marketing people, should be very interested in helping you with. It is part of their job.

Usability.**Intuitiveness:**
Type: Product Line Critical Quality Level.
Version: 18 September 21xx 01:52

Scale: Time needed to Correctly do a defined [**Task**] by a defined [**User**] given defined [**Experience**] and defined [**Knowledge Access**] for a defined [**Product**].

Past [Date = January 2018, Task = Assemble + First Use, User = Average, Experience = None, Knowledge Access = On Product Help + Manual, Product = Product Line X] < 10 minutes.

Ideal:zero time to do task for everybody

Record [Date = June 2018, Task = Assemble, User = Professional, Experience = Years, Knowledge Access = On Product Help + Manual + Internet + Tele Help Line , Product = X 3.0] 1.5 minutes.

Trend [Date = End 202x, Task = Assemble + First Use, User = Novice, Experience = None, Knowledge Access = On Product Help + Manual, Product = X 5.0] 15 minutes. "Getting worse as product gets complex"

Goal [Date = End 202x, Task = Assemble, User = Professional, Experience = Years, Knowledge Access = Online Help + Manual + Internet + Tele Help Line, Product = X 3.0] 1.0 minutes "would be new record".

Goal [Date = Mid 202x, Task = Assemble + First Use, User = Novice, Experience = None, Knowledge Access = On Product Help, Product = X 5.0] 1.5 minutes

10. USEFUL BACKGROUND SPECIFICATIONS IN AN OBJECTIVE. DIFFERENT GOALS FOR DIFFERENT STAKEHOLDERS. THE + INDICATES MEMBERS OF A SET OF THINGS, OR A LIST OF THEM.
IN THIS EXAMPLE, THE PAST, RECORD, AND TREND ARE ALL 'BENCHMARKS' WHICH IS ONE TYPE OF BACKGROUND INFORMATION ABOUT THE PLANNED OBJECTIVE. THEY ARE *NOT THE CORE OBJECTIVE ITSELF (*NOT OUR *FUTURE* TARGETS OR CONSTRAINTS). <- PLANGUAGE EXAMPLE 4.3

10. Ideal and Record: a state-of-the-art extreme, attained under defined conditions.

- These 'Ideal' and 'Record' benchmark levels, are informative *warning* signals.
- The **'Ideal' Level**, I rarely use, except when someone mumbles '24/7' as managers do, and I point out that they, taken literally, are asking for 'perfection'.
  - For example, 24/7 implies '100% availability'. Nice, but real engineers know it cannot be done in the long term, and that the cost of trying to get there, tends towards infinity.
- So the Ideal, is a way of saying, *we have no illusions of doing the impossible quickly*.

- The **Record Level** is a similar nature. A warning sign. Maybe for some an interesting challenge.
- **Record Level** synonym is 'state of the art'. Analogies are 'World Records'.
- Knowing the Record is a sign of *expertise*.
- If your requirements are near, at-or-above the Records, then there is a warning signal, a risk warning, that you had better be more-expert than those who set the Record!
- And even if you can 'beat the record' there is a high risk of delays, and costs that nobody can understand, or forsee yet.

National Security

Business Value ***Label?***

(✎ by tomgilb - 2 months ago)

Permalink

0.0.1

Is Part Of: Stakeholder Values Value

Ambition Level: to reduce terrorist attacks, and identify potential terrorist attacks, and regulate cyber information

Scale: Number Negative [Effects] on [Stakeholders] from [Attack Types] under [Conditions] in [Places] per year for given [Area]

Stakeholders: Prime Minister, Casualties, Council Representatives, Police, Relatives Of Victims, Volunteers

Status: Level: **150** Number Bad Stuff [Effects = { Death }, Stakeholders = { <All> }, Attack Types = { Vehicle Attack,Knife Attack,Gun Attack }, Conditions = { High …

Wish: Level: **10** Number Bad Stuff [Effects = { Death }, Stakeholders = { <All> }, Attack Types = { Vehicle Attack,Knife Attack,Gun Attack }, Conditions = { High Alert.

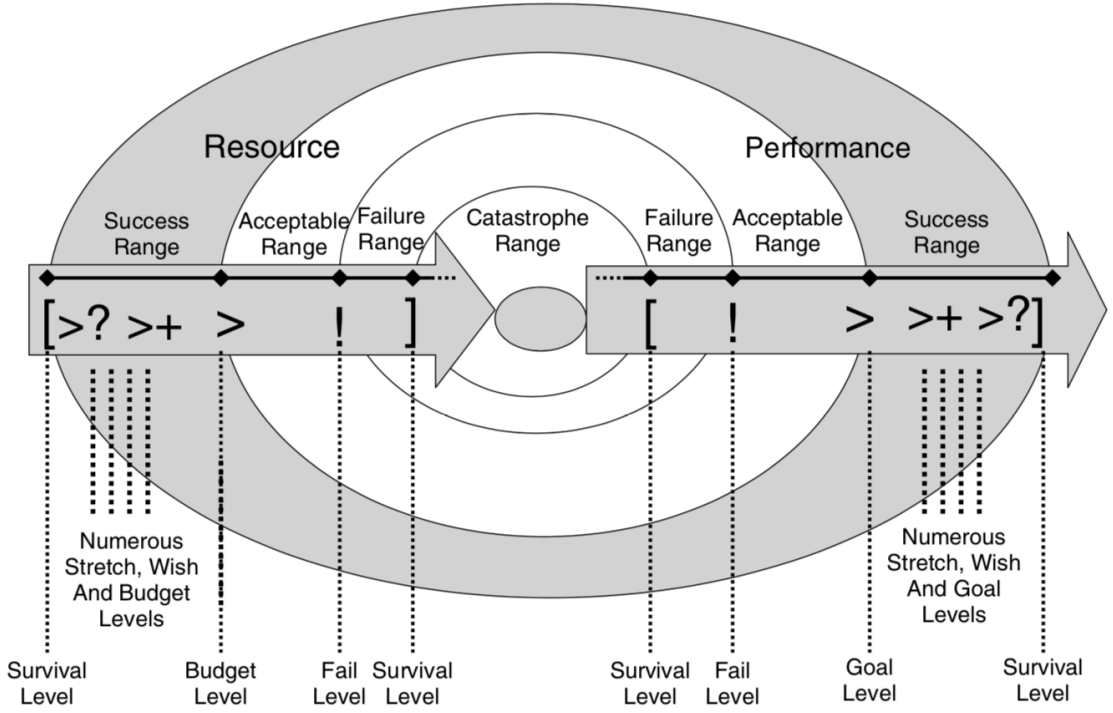
Record: Level: **1** Number Bad Stuff [Effects = { Death }, Stakeholders = { <All> }, Attack Types = { Vehicle Attack,Knife Attack,Gun Attack }, Conditions = { High Ale.

Due: **Planned (by end of):** ?

11. STATUS LEVEL. YOU NEED TO KNOW 'WHERE YOU ARE' BEFORE YOU CAN MAKE GOOD DECISIONS ABOUT 'WHERE YOU WANT TO GO'.

11. Status Level

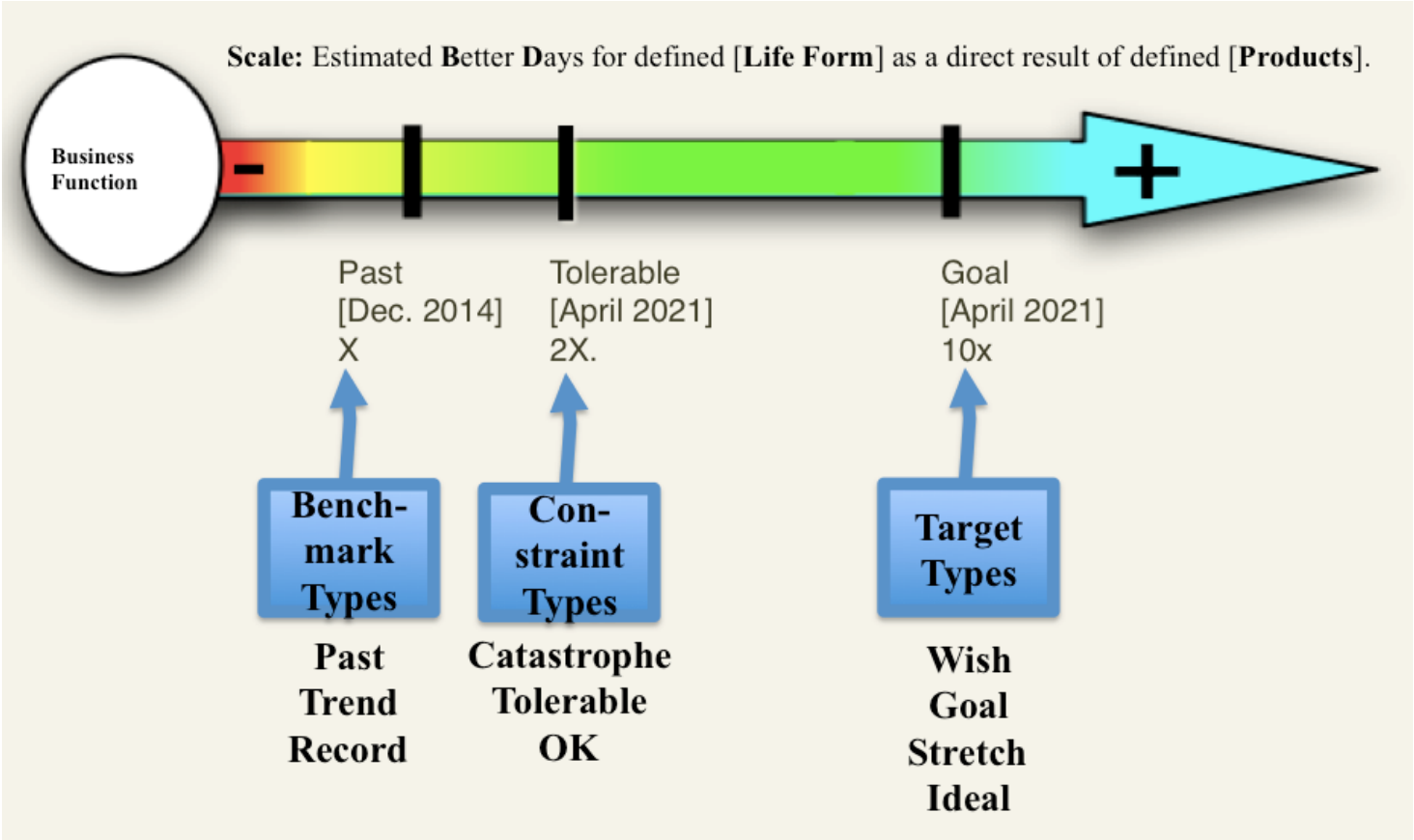
- ‘Status’ generally, is any registered set of conditions, for example the Quality status of a plan.
- Status here, is a short form for **‘Current Status Level’**. It represents a moving, changing, level of real-time report, right now.
- Once the Status moment is passed, then it becomes a Past level report.
- Status has very similar uses to what Past Level does.
- A good practice is to clarify exactly when and where the Status data was collected.
  - **Status**:[2018 September 18, Oslo, Adults, Calling In to Us] 42.
  - So that even when the Status moment has passed, we can still understand the analysis data, and realize that it is not a ‘current’ status.



12. DOUGHNUT DIAGRAM [CE BOOK G19], SHOWING HOW SCALAR CONSTRAINTS DEFINE MORE-OR-LESS ACCEPTABLE RANGES OF ATTRIBUTE LEVELS.

12. Scalar Constraints: levels to avoid.

- Scalar Constraints specify a requirement to **avoid certain performance levels**.
- They help us map *ranges of performance* that are unpleasant, unacceptable, illegal, or disastrous.
- They are a tool to help us *see risks* in our planned designs, and in our project execution
- They are a tool to help us *prioritize* using resources, so that we use resources to achieve the minimum constraint levels (not too cold), and also to *stop* using resources when we threaten to achieve more than necessary (too hot)



13. SETTING A LOWER BOUNDARY A 'TOLERABLE LEVEL';
A PRIORITY SIGNAL ABOUT THE 'WORST ACCEPTABLE' LEVELS

13. Tolerable Level: we can live with this level, but are not happy bunnies.

- From the Tolerable level, and better, we have a ‘acceptable’ range of performance.
- The Tolerable Level is just above the Fail Level.
- Fail-and-worse levels, down to the Survival Level, are a ‘Failure’ range of performance.
- *Failure* ranges might be tied to non-payment for systems or products.
- ‘*Acceptable*’ ranges might bear some penalty for less-than-desired performance (Wish, Goal, Stretch).
- We can specify a wide range of tailored scalar constraints using Scale Parameters, and other conditions in the Scalar constraint level statements (Tolerable, OK, Survival)
- This is a Technoscope helping us to distinguish between success and failure, and in-between levels of performance.

Sleep:
Scale: Hours deep sleep per night, average.
Tolerable: 0.2 .
OK: 4.
Goal: 7.

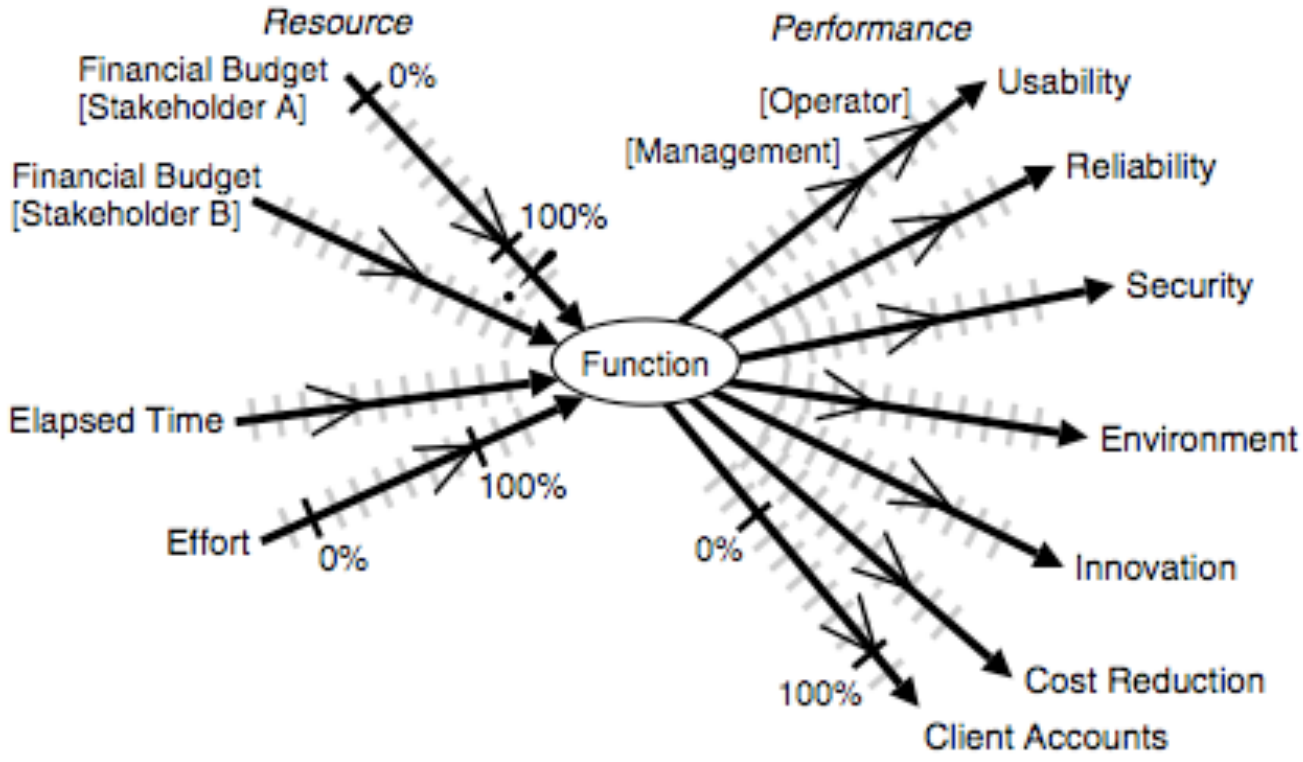
Hunger:
Scale: % of belly full.
Tolerable: 1%.
OK: 50%.
Goal: **100%.**

Air:
Scale: % of clean air needed per day.
Tolerable: 1%. (below this is intolerable)
OK: 50%. (at this level or better, things are satisfactory, but not good)
Goal: 100%. (at this level, we have 'success')
Stretch [For Pensioners and Athletes] 130%. (this level is valued, but marginal, luxurious)

14. THREE CONCURRENT HUMAN BODY OBJECTIVES. OUR PRIORITIES VARY DEPENDING ON SUPPLY AND NEED

14. OK: a range just above the tolerable range.

- Not intolerable.
- Not failing.
- Pretty ‘good’,
- but not yet at an ambitious and competitive ‘success’ level, (called the Goal). Not yet a 'Success'; see Targets, below.



15. <u>TARGETS</u> ARE LEVELS OF PERFORMANCE, ON MANY SIMULTANEOUS VALUE SCALES.
THESE TARGETS NEED TO COMPETE FOR MULTIPLE LIMITED RESOURCES, AS SPECIFIED BY A 'BUDGET' LEVEL

15. Targets: Performance levels we want achieve. Success.

- **Target Levels** is a class of requirement levels, that express **sufficient**, and **successful**
- Target Levels: include Wish, Goal, and Stretch (details below)
- Target levels can be tied to completion, meeting deadlines, and being within budget, if you achieve them.
- The main distinction is that:
  - **Wish** acknowledges a stakeholder *desire*
  - **Goal** expresses a project delivery *commitment*
  - **Stretch** articulates some *additional value*, if you can still afford the resources
- These concepts can help us *prioritize our use of resources,*
  - Like: Goals *before* Wishes. Goals *before* Stretch
- And help us understand and manage risks
  - ‘You might not get paid if you don’t reach the Goal level on time’.
- High target-levels will generally drive costs *up.*
- There needs to be a reasonable correlation, between the benefits from reaching the target levels, and the total costs of attaining them. Return on Investment.
- Target levels are used as the 100% level in Value Decision Tables (below). They help us make decisions about different strategies, and their related value attributes, compared to our plans.

W1: Wish [Deadline = New Years Day next year, Area = Canada, People = Teenagers, If Trade Agreement Valid] 30%.

Planning Status: [W1, December 24, 20XX]
PS1: Strategies: S1, S2, and S5 are estimated to get us to about 150% ±60% of the Goal <- IET ABC.

PS2: Capital Costs of the strategies are estimated to be 35% of total project budget

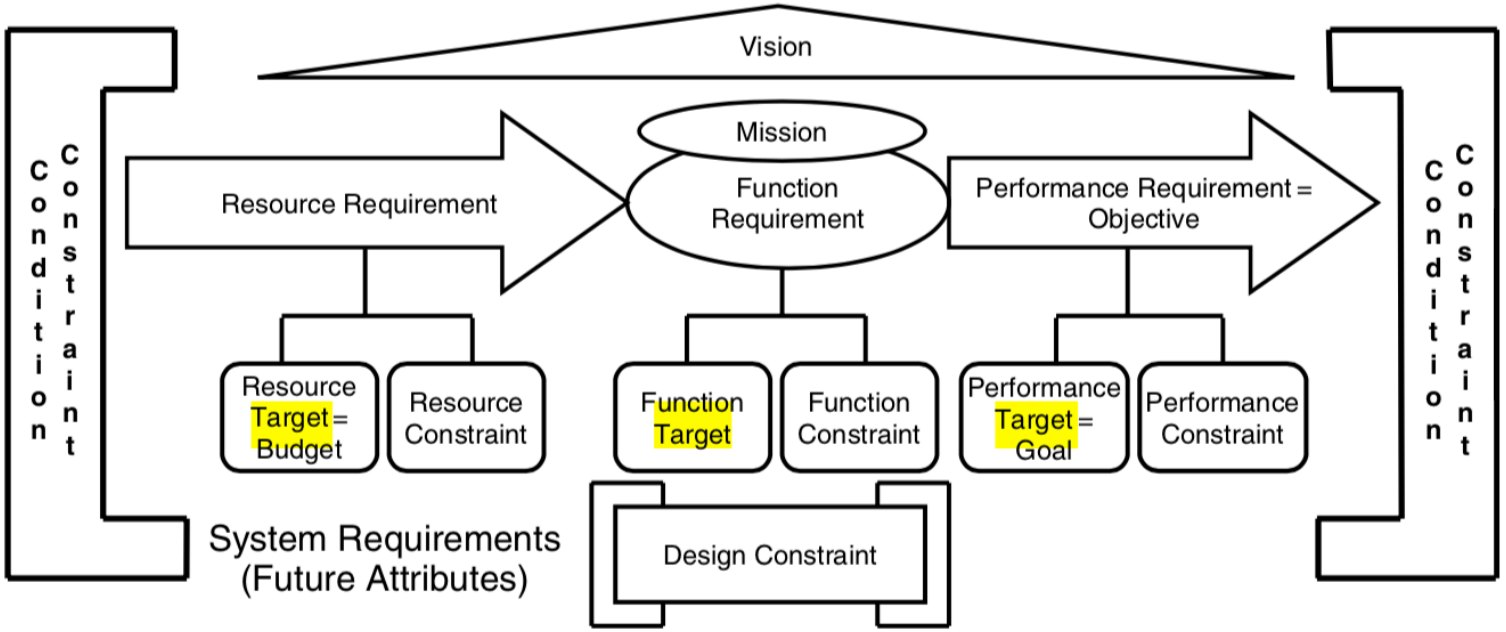
PS3: We do not know if we have qualified people available to carry out S5.

PS4: We do not know, at this stage of planning, if this Wish will have sufficient priority, ahead of all current commitments at that time; and there are still possibilities of additional commitments which could interfere.

16. DOCUMENTATION, INTEGRATED IN THE PLAN, EXPLAINING WHY THE WISH STATUS CANNOT BE CONVERTED TO A 'WE PROMISE' GOAL IN THE PLAN. [VP PLANGUAGE 6.10 C]

16. Wish Level: a stakeholder desire for a performance level with *some* value for them.

- A wish level is a level of performance that at least one stakeholder desires, because it has some value for them
- A wish level is *not* a project-committed requirement (like Goal, or Tolerable are)
- A wish level has to pass some tests: feasibility, economics, consistency with other commitments (see Goal below)
- A Wish level is a project's way of saying:
  - *we understand this level would be valuable to you. So we cannot promise anything. But we will look into it for you. If we convert it to a 'Goal', then that means, we are promising to deliver it.*
- The decision to re-classify a Wish level to a committed Goal level, requires systematic consideration of a large number of related factors (other objectives, other resources, specified constraints). See the map next to Goal Level below.
- This (Wish to Goal analysis) is a serious Technoscope tool for helping us manage complex sets of stakeholders, their wishes, and limited resources.



17. GOAL LEVEL REQUIREMENTS ARE ONE PART OF A LARGER PICTURE OF REQUIREMENTS.
ALL REQUIREMENTS MUST BE CONSIDERED,
FOR MULTIPLE SIMULTANEOUS OBJECTIVES,
BEFORE COMMITTING TO A SINGLE 'GOAL' LEVEL REQUIREMENT SERIOUSLY [CE FIG 2.2]
THIS EXCELLENT VISUALIZATION ENTIRELY DUE TO CE EDITOR, DR. L. BRODIE

17 Goal Level: one of several simultaneous (Target and Constraint) commitments to 'deliver levels of value to stakeholders'.

Here is the set of conditions that we think are logically necessary, for a project team to commit to a 'Goal level' value delivery.

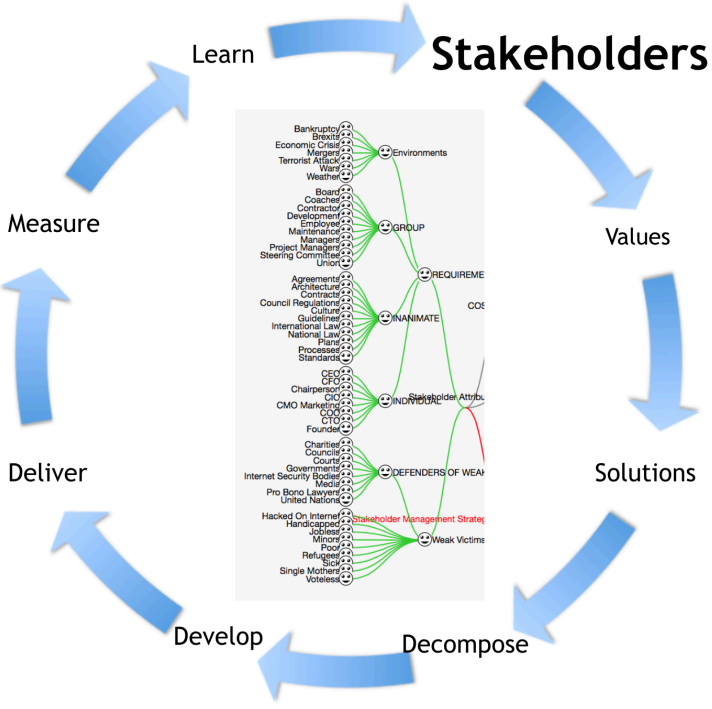
The Goal Level should:

1. be **technically** possible - within the state of art. There are known strategies to get there.
2. be **economically** possible - resources exist, or can be obtained.
3. have costs **consistent** with other requirements. *This* is **not** the *only* objective!
4. be **effective**, and it must deliver the effect necessary to satisfy real stakeholder needs.
5. be **profitable**: value exceeds costs.
6. be **prioritized**: by agreed rules of priority, such as Effectiveness, Profitability, or Politics.
7. have **all [Qualifier Conditions] i**n the Goal statement as '**true**'.

18. Stretch level, a Target, <u>if</u> we can prioritize any remaining resources.

18. Stretch Level: additional value if we have any resources left, after reaching all Goal levels.

- A Stretch requirement level, has a priority AFTER all currently required Goal levels are reached.
- But if we *have reached* all currently (value requirement *deadline* decides what is 'current') then we can use *any remaining resources* (people, time, money) to work towards reaching one-or-more specified Stretch Levels.
- The 'Stretch' implication is that there is some value (knock-on value at a higher level) to be attained by reaching, or at least *moving towards* the Stretch level.
- So, without further bureaucratic decisions, the project is authorized to make use of residual resources, to go beyond the Goal level, and up to, but NOT beyond, the Stretch level.
- This tool helps us to prioritize and differentiate, the **essential** (Tolerable, then Goal) from the '**nice to have**' if you can afford it.
- Meeting Stretch levels within resource constraints, and with all current Goal levels met, could be the subject of some motivation, like contractual bonus awards.
- The Stretch level is not necessarily for the same set of [Qualifiers] in the corresponding Goal statement.
- We can specify any narrow priority of conditions, for the Stretch target that interest the stakeholders
  - **Goal [By Next Year, People = Adults & Pensioners, Tasks = Economic Support] 60%**
  - **Stretch [Within 18 Months, People Schoolchildren, Tasks = Educational Support] 70%**



19. SOME STAKEHOLDER EXAMPLES, AND TYPES

19. Stakeholder: any entity with requirements for us to consider.

- Stakeholders can be individuals, groups of individuals, types of individuals, and non-human requirement sources such as laws, standards, plans, policies, rules, and principles.
- A critical stakeholder has at least one critical requirement: if you do not meet it, success is threatened, failure is a possibility.
- There are normally many stakeholders; dozens.
- New **stakeholders** can pop up, at any stage of a system lifecycle
- New and critical stakeholder **needs**, can pop up at any time.
- Systematic stakeholder analysis is a critical tool, for understanding non-trivial systems, over time.
- One objective (like ‘Reliability’) can be of interest to *several* stakeholders simultaneously
- But different stakeholders may require different Goal levels, for that one common objective.
- So when one stakeholder requests a change in the Goal, the other stakeholders should probably be consulted.
- That implies you need to keep real-time track of stakeholders during a project.

Software Redundancy:

Type: Technical Strategy.

Description: use triple redundant distinct software, with 2 of 3 voting corrections, and otherwise manual decisions after stop because of output differences. <- *Wendy Bartlett.*

Estimated Impact on [Availability] = 30%±10% <- *Linda Queen.*

Evidence: = Similar levels found at Our Corporate R&D **Labs.** Using Distinct Software with 3 Versions <- *Tommy Scharf.*

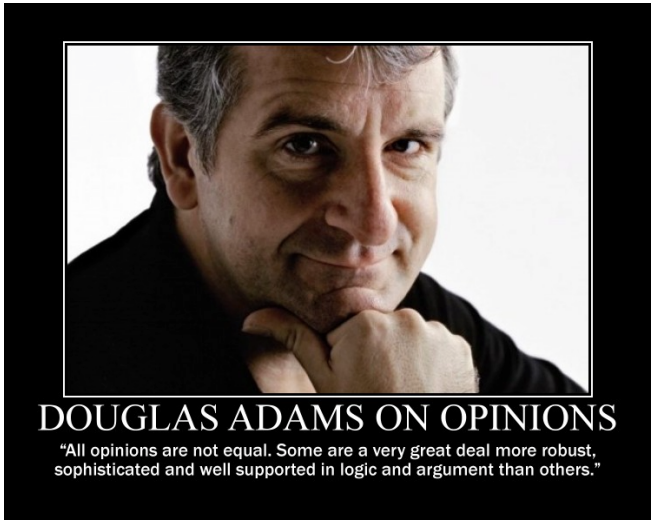
20. 3 SOURCES OF 3 DIFFERENT STATEMENTS

20. Source <- : The written or human source of any planning statement.

- It is very cost effective to annotate the exact source, for many specifications, in direct proximity to that specification.
- This is primarily so we can easily do quality control, by accessing that source, and checking that it is still a valid specification (things and people move on)
- It is secondarily, so people can see the 'status' of the source, and show 'due respect' to the decision
- It is also useful to ask people to take personal responsibility for being the source: they will then be motivated to make sure the specification is correct and complete. Take ownership
- The 'keyed icon' '**<-**' meaning, 'the source of this is to the right', has been found very convenient, if you do this as often as you should.
  - But you can also just write. 'XXXXX. **Source**: URL gilb.com ' for example
  - Or use other conventional source annotation, such as footnotes, or [Brackets]
  - You can also use Tags, which bring you to more detail via the tag, to both keep it short, and to reuse the source specification, which is useful if it is a long citation.
    - Tag: Statement <- Source Tag
    - **Source Tag**: lots of detail, here as you please.

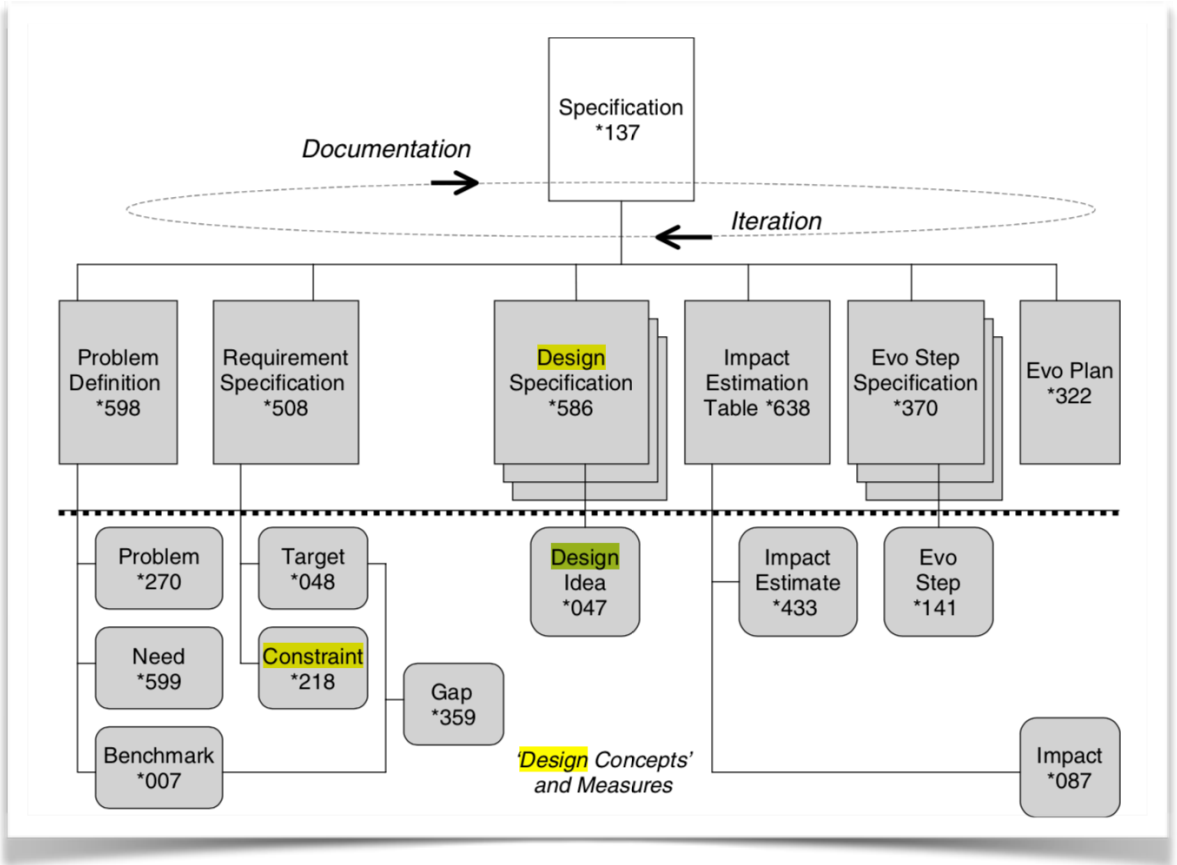
“A COMMON MISTAKE PEOPLE MAKE WHEN TRYING TO DESIGN SOMETHING COMPLETELY FOOL PROOF IS TO UNDERESTIMATE THE INGENUITY OF COMPLETE FOOLS.”

Douglas Adams (1952-2001) Mostly Harmless



Part 2: Designs, and Planguage

21 Design Constraint
22 Design Specification
23 Design Components
24 Independence of Implementation
25 Value Delivery Design
26 Concepts
27 Local Definitions
28 Global Definitions
29 Reusability
30 Risks
31 Issues
32 Dependencies
33 Supports
34 Supported By
35 Decomposition Principles
36 Design Principles
37 Templates (with hints)
38 Design Logic
39 Design to Objectives
40 Multiple Attributes

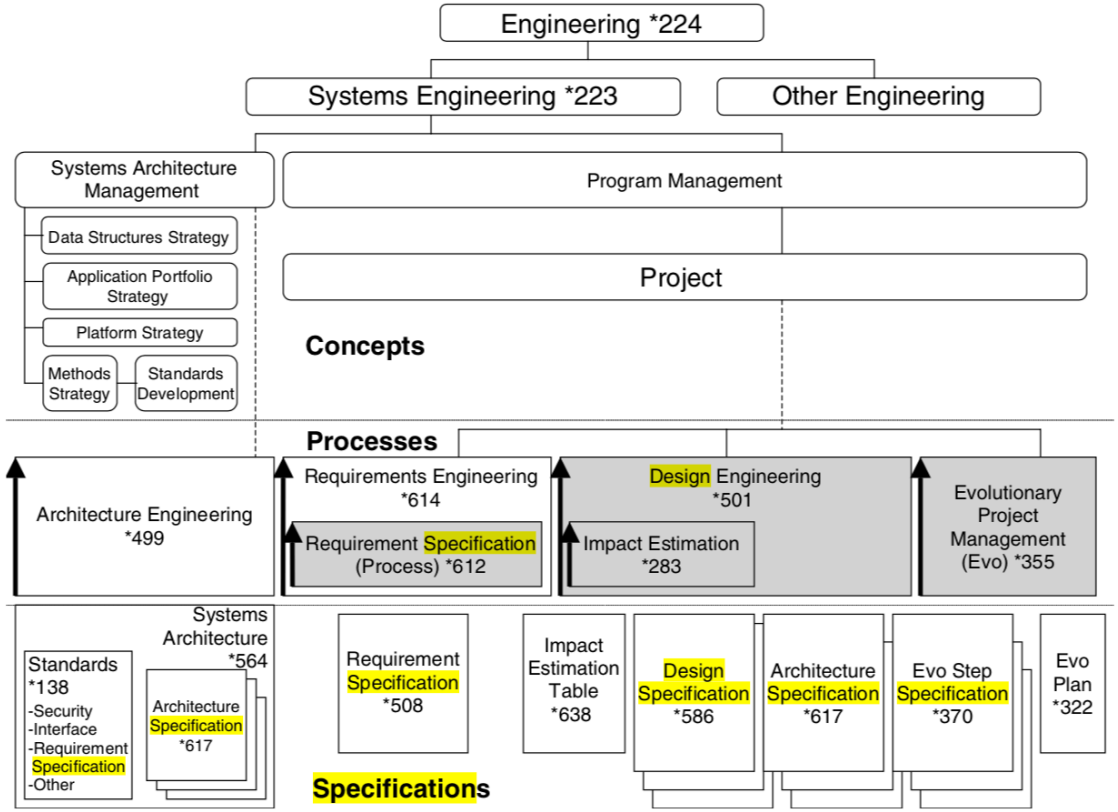


21. A 'DESIGN CONSTRAINT' (*218) IS A 'DESIGN IDEA' WHICH IS A 'REQUIREMENT', NOT A 'DESIGN *OPTION*'
SOURCE: 'COMPETITIVE ENGINEERING' [CE]

Source: 'Competitive engineering' book 2005, planguage concept glossary

21. Design Constraint: Solutions you must use, or must NOT use.

- A design constraint is an *explicit* and *direct* restriction regarding your *choice* of design ideas.
- It either declares a design idea to be *compulsory* (Mandatory Design), or to be *excluded* (Prohibited Design).
- Design constraints are dictated from an 'earlier system development stage' (a higher level or a more specialized level).
  - For example, the system architects, pass on a number of design constraints, within the architecture specifications, to the system engineers.
- A design constraint is a *binary* requirement. Do or Don't.
- It can be a *generic* constraint or involve specific design(s).
  - A *Generic Constrain*t example: "Use Only Designs from our Own Patents, where possible"
  - *Specific Design* example: "Use ISO Standard 123-4567 (April 2018)"
- This is a tool to help designers, architects, strategic planners to 'home in on' solutions early, and to avoid fear that some solutions, which violate the Design Constraint, will be rejected later, because *we were not aware* of the Design Constraint.
- A Design Constraint needs an *authority* behind it, and even that authority needs a **Justification** for the decision.
  - The Justification should be embedded in writing, together with the Design Constraint.
  - The justification might turn out to have *lower priority* than other requirements, and this can lead to rejecting the Design Constraint.
  - A Design Constraint, is not absolute, there are many competing priorities.
  - It is probably a good idea, to discuss rejection of the Design Constraint, with the source of it, or authority for it.



22. A *DESIGN* SPECIFICATION (*586) IS A 'TEMPORARY' IDEA, FOR SATISFYING *SOME* REQUIREMENTS. IT IS NOT 'HOLY' AND CAN AND MUST BE CHANGED, IF WE GET *BETTER* IDEAS. SOURCE: [CE]

22. Design Specification: how we ‘plan to satisfy' requirements

- A design specification is the *written specification* of a design idea.
- A set of many individual complementary design specifications is the *main output* of a *design engineering process.*
- A specific set of design specifications, when implemented, will, to some degree, *try* to meet the stated requirements.
- ‘Design’ is generic, and covers any useful variety of more-specialized design specification ideas, such as:
  - Architecture, Specialist Design (eg Security, Usability), Project Delivery Step Design, etc.
- Design is only ‘design’ *specifically in relation to a defined set of requirements*. Design is a *relative* point of view.
- At one extreme, a design could be expressed in terms of ‘Means Objectives' (R Keeney), a set of requirements,
  - which when satisfied, will result, we hope, in the delivery of a higher-level of requirements.
    - Simple example, A Technical Security Requirement, when achieved, will lead to lower economic hacker-losses at a higher level of concern.
    - But those ‘means objectives’ (like Security) might not yet have their own technical design specification, yet.
- The design spec can be articulated by any useful variety of words, numbers, tables, illustrations.
- Ideally, a design spec is *clear,* and ultimately *complete* (complete enough to meet all requirements).
  - *Clear* so that we can logically understand its claim to deliver the requirements,
  - and so that the implementors *cannot misunderstand* the designers’s intent.
- We know how to *quality control* design specifications, in relation to good design rules, and in relation to their purpose of helping us to meet requirements. See below in this booklet for several methods.
  - (Spec QC, VDT, Evo are specific Technoscopes for measuring the quality and suitability of a design spec)
- Design Specs answer the Technoscope question ‘how?’, by which means?, do we think we can reach our objectives?

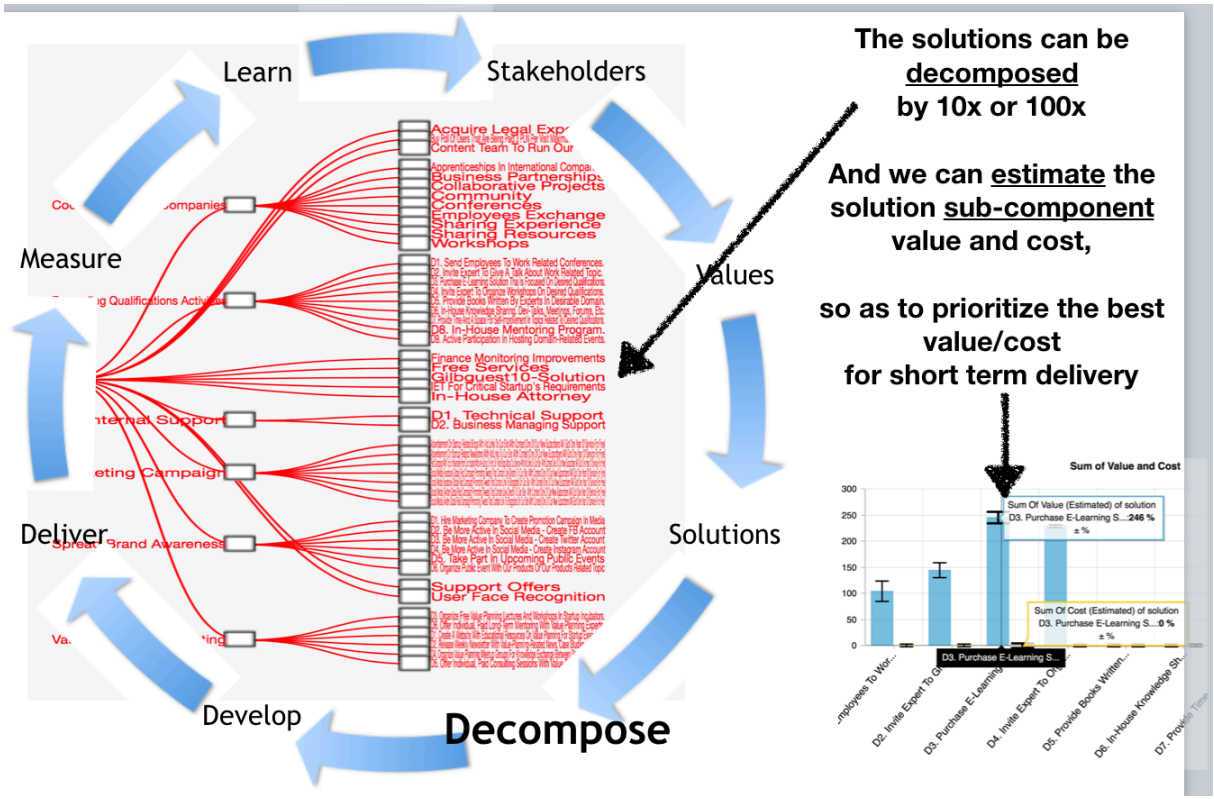
23. DIFFERENT TYPES OF DECOMPOSITION INTO COMPONENTS OR SUB-SETS

Source: Value Planing book, Chapter 5 Decomposition

Value Decomposition Methods	Details		
Design Decomposition	Independently Implementable	Measurable Value Delivery ability	Urgent subset
Value Decomposition	Multiple Values	Scale Dimensions	Scale Dimension Sets
Deadline Decomposition	Date Due Start Date	Event Synchronization like 'Law in Force', 'Official Holiday'	Condition Synchronization like 'Interest Rates over 5%','Heat Wave'
Priority Type Decomposition	Wish (Desired Level)	Tolerable (Minimum Level)	Goal (Committed Level)

23. Design Components: useful subsets of your solutions.

- Sometimes our design ideas need to be decomposed into a set of *smaller* ideas.
- There are several potential purposes for design decomposition:
  - to bring out more-specific detail. To define it better.
  - to understand costs better.
  - to understand different values, from each sub-design
  - to enable us to implement *high-priority* design components, *earlier*
- In all these cases, *decomposition* into a set of separate design components is a Technoscope: it helps us understand a complex system better.
- The *methods* of decomposition is a separate subject. But the end result, *decomposition*, is the *tool* for understanding the complex system, better.

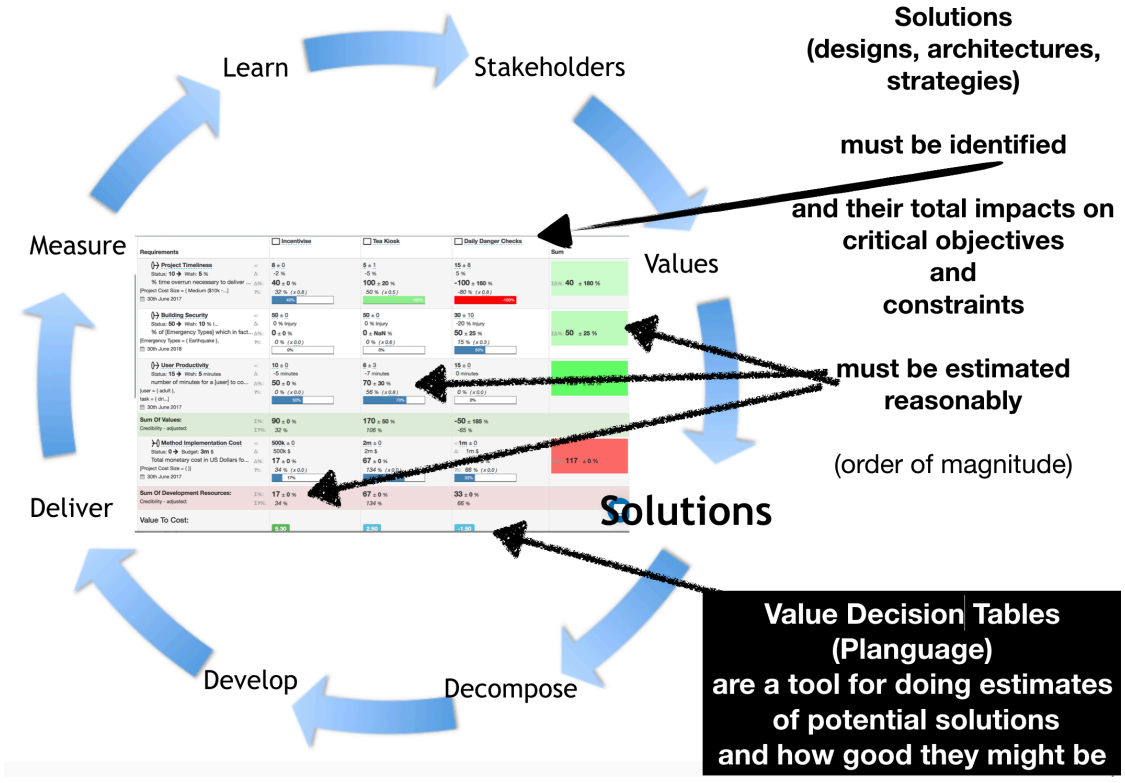


24. HERE IS A REAL EXAMPLE OF MY CLIENT'S DECOMPOSITION OF DESIGN SOLUTIONS INTO INDEPENDENTLY IMPLEMENTABLE COMPONENTS. ON THE RIGHT LOWER CORNER IS AN ESTIMATE OF THE EFFECTIVENESS OF EACH SUB-COMPONENT, SO THAT WE CAN CHOOSE TO GO IMPLEMENT THE MOST EFFECTIVE ONE EARLY

Source: ADVANCED AGILE SOFTWARE ENGINEERING Turkey, April 2018, Polish Case

24. Independence of Implementation: we can choose any design component, before all the others, and implement it, to get a result.

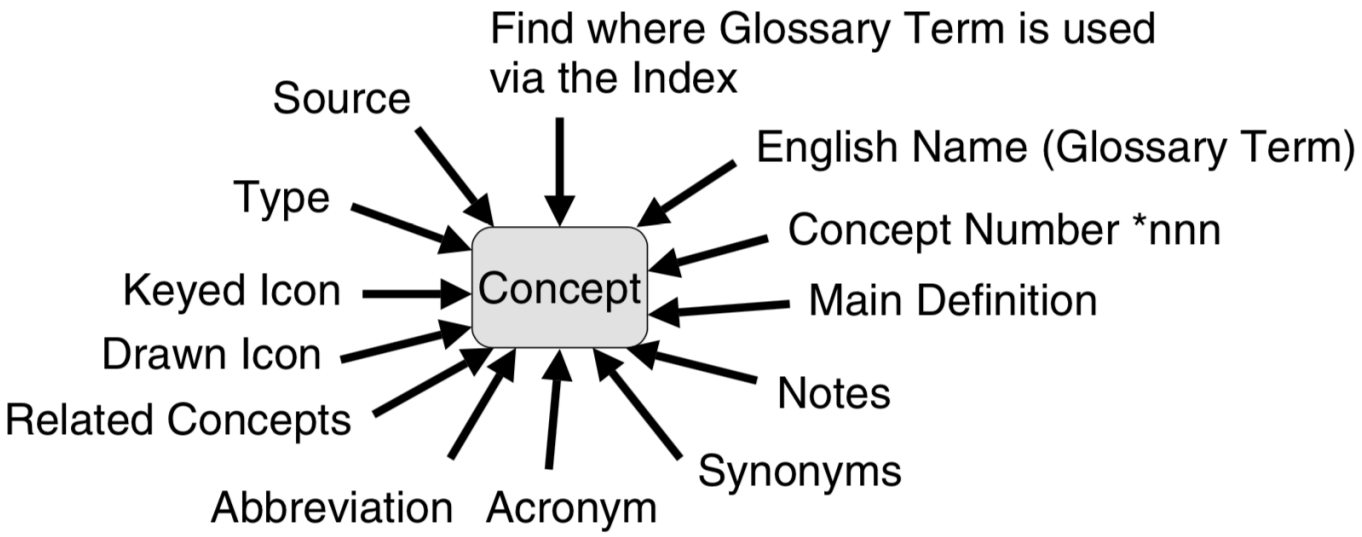
- When we decompose our designs, solutions, or architecture, there are several ways to do it.
- One very useful purpose, for a decomposition, is that *each sub-componen*t can be implemented, incrementally into a system you are improving, *without* us needing to implement *any other* sub-component before it, or at the same time.
  - We can implement any single sub-component *next*, without 'dependencies' on any other sub-components.
- A simple way to create independent sub-components, is simply to separate into sub-components, then ask, can I implement this one *before* any of the rest of it. If *yes*: I *have* an independently implementable sub-component. Continue the process. If not, go back and try again. We can discuss more direct and elegant ways to do this. But this will serve as a starter method.
- My experience is that people are very good at just doing this, when they are asked to '*make them independen*t': but if not asked, they unconsciously decompose without independence of implementation. You need to be conscious of it.
- Why is this decomposition tool a Technoscope? A tool for understanding a complex system.
  - Because
    - It gives us more realistic detail, than a non-decomposed top-level solution
    - We can use agile methods, to try out some of the best ideas early, and measure how well they work in practice, before committing to further extended implementation, of what might be a bad idea, or an idea needing re-design



25. THE PROCESS OF DESIGNING AND DELIVERING MULTIPLE VALUE-REQUIREMENTS: ENGINEERING VALUE. USING VALUE DECISION TABLES (VDT). SOURCE: ADVANCED AGILE SOFTWARE ENGINEERING TURKEY APRIL 2018

25. Value Delivery Design: Consciously designing for specific value levels, on time.

- Most projects do not consciously and *explicitly* design for 'value delivery': for any type of value, not just financial.
  - There are almost no *quantified values specified* in *your* requirements, to start with (check it out)
  - There is no explicit *design process* matching design ideas, to the required values, numerically (like VDT)
- Without these explicit design-for-values processes, there is no 'value transparency', at early decision-making stages
  - The consequence is that huge sums of investment is wasted, when the 'talked about' values never materialize.
  - Value Transparency requires the following set of tools, discussed in more detail, elsewhere in this booklet
    - *Quantified* value requirements.
    - *Decomposition* of big ideas, into smaller more finite, implementable, testable ideas
    - *Estimation* of the value impacts of the ideas, on the 'required values' (using Value Decision Tables)
    - *Prioritization* of ideas into high-effect, and low-risk, partial-implementation, delivery
    - *Measurement* of the real value-effectiveness, in small increments, to prove good ideas really work, or not
- *This* set of tools is necessary to *see* the values of design ideas, at all stages of consideration and implementation.
- You are probably missing these tools, so you are 'flying blind', and will fail.
  - But that is OK, if you are good at covering up, and blaming other causes, for failure.



26. CONCEPTS CAN BE REFERENCED BY A WIDE VARIETY OF TERMS AND SYMBOLS. THIS SOLVES THE 'DICTIONARY PROBLEM' OF PEOPLE DIFFERING ON 'WHAT THEY MEAN BY A WORD'. IS ALSO OPENS UP FOR OTHER LANGUAGES' TERMS, 'POINTING TO A CONCEPT DEFINITION'. AND IT ALLOWS GRAPHICAL SYMBOLS TO POINT DIRECTLY TO A CONCEPT DEFINITION, RATHER THAN TO A TERM, WHICH WILL HAVE 'VARIOUS' INTERPRETATIONS. SOURCE [CE]

26. Concepts: ideas and definitions, independent of actual words, or symbols, which we use, to refer to the concepts.

- Part of seeing what is going on in a complex system, is understanding the exact intended meaning of the terminology in use. Here are our Technoscopes for that problem.
- The problem is that very few ‘words’ actually have an exact standard definition
- The Planguage approach to this everyday problem is to
  - Assume that all undefined words are dangerous, and can mean *almost anything*
  - Make sure that all **critical** words are formally defined *somewhere*
  - Make sure that undefined, but **critical** words, are marked explicitly as <fuzzy>, i.e. ‘needing definition’
- This starts in the Planguage Concept Glossary (not called a ‘dictionary’). This contains under 800 concepts at present. Each concept is defined, and given a number like *123. Then I attach pretty good English words to it, as my personal choice of a useful term for the concept. But I explicitly recognize synonyms, translations, symbols, and acronyms. In addition I give any user, the freedom to attach any terms, they feel they want to use.
- These planguage terms are then use in specs, like *requirements* (Scale, Past etc) and *designs* (Impacts, Impacted By) giving a pretty good beginning, to understanding more-exactly what is specified.
- Then, in addition:
  - All Tags, and we use many, are in fact a reference to something which *is defined more fully* in the specs
  - Special definitions, can and should be created everywhere, where it seems important that all spec readers, have an accurate understanding of the terms, not just ‘their best guess’.
  - Our Planguage app (valplan.net) encourages *reuse of Tagged Terms*, and *building specialist glossaries*.
- Understanding of Terms can never be perfect, but we have measured (Using Spec QC, see Terzakis Intel, 2013) that we can make intelligibility at least 100X better using these methods

"When I use a word", Humpty Dumpty said, in a rather scornful tone, "it means just what I choose it to mean—neither more nor less." "The question is", said Alice, "whether you can make words mean so many different things." "The question is", said Humpty Dumpty, "which is to be master—that's all." Lewis Carroll, Through the Looking Glass, *Chapter VI (Humpty Dumpty), 1871.*

27. YOU NEED TO TAKE RESPONSIBILITY FOR YOUR WORDS, AND BE THEIR MASTER

Source: Gilb, Competitive Engineering. [CE]

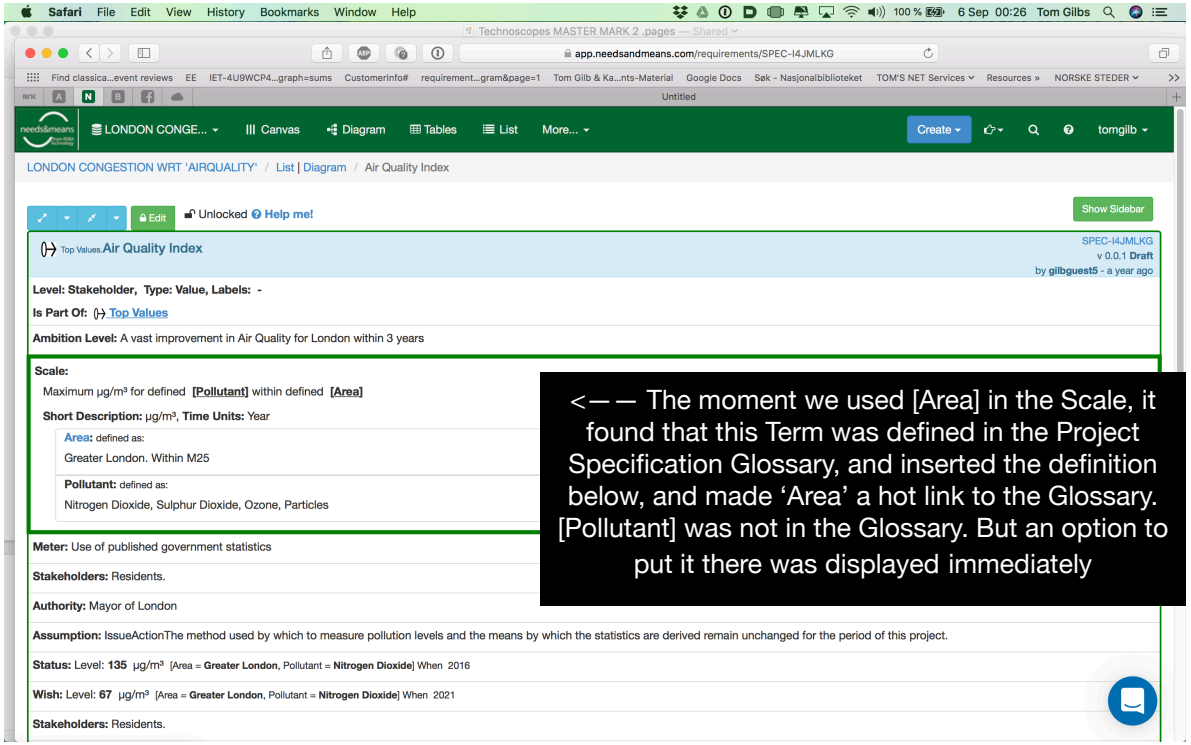
27. Local Definitions: making sure people understand your intent, on this page or screen.

- One Technoscope Tactic, is to allow, and encourage direct **Tagged** definitions of words, which are then used quite locally, in one design spec, or one requirement spec.
- I take no chances. It is much cheaper to *define in writing right now*, than to *hope* that everybody on the Planet, will understand you correctly, later. Too Late.
- Even when I have underestimated the need to define initially, I get a clear message when I hear someone either clearly misunderstand a term, or directly ask me or us what it means. I almost do not bother to answer them orally then and there. I immediately write the definition into the specifications. Done. Forever. Preventing the next misunderstanding, when I am probably not even there to take action. This also gives you an immediate chance to get feedback on your definition!
- As a simple example:

> **Motivation**
>
> **Type**: Design
>
> **Description**: <Teams>, not **I**ndividuals will be **M**otivated and **R**ewarded as <groups>
>
> **Motivated**: inspired explicitly or implicitly or internally by any applicable legal ethical means.
>
> **Rewarded**: acknowledged by any means that actually works in the specific situation.

Note: the words in <Fuzzy Brackets>, are *acknowledged* to need better local definition. The *Capital* '**I**' in Individual, indicates i**ntent to define**, even though it is not done, yet, here.

- If you think this is too much work, then you have too little experience of cleaning up, after the loss of time, money and credibility the happens, when people misunderstand (like your management, and suppliers). 1000x more + 're-work'.

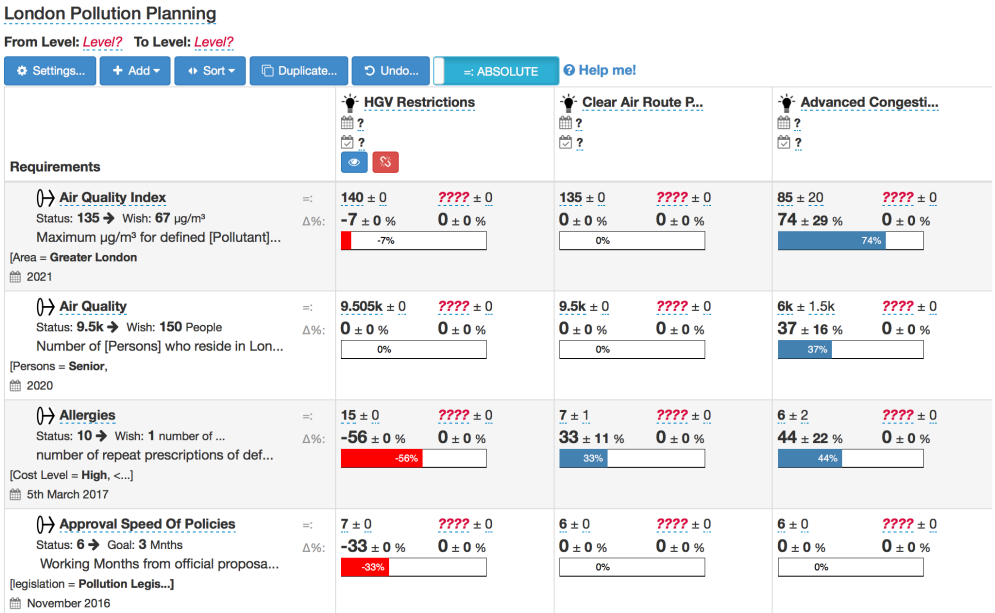


28. "AIR QUALITY INDEX" IS A 'GLOBAL' VALUE REQUIREMENT DEFINITION ('GLOBAL' FOR THIS PROJECT), IT CAN BE REUSED AND UPGRADED TO 'AVAILABILITY FOR A WHOLE ORGANIZATION'. "AREA" IS A HOT LINK TO A PROJECT-WIDE DEFINITION, AND CAN ALSO BE UPGRADED TO ORGANIZATION-WIDE USE. "POLLUTANT" IS, FOR THE MOMENT, LOCAL (TO THIS REQUIREMENT), BUT CAN BE UPGRADED, FOR REUSE IN OTHER SPECS.

March 9-10 2019 Edit

28. Global Definitions: Definitions with a project.

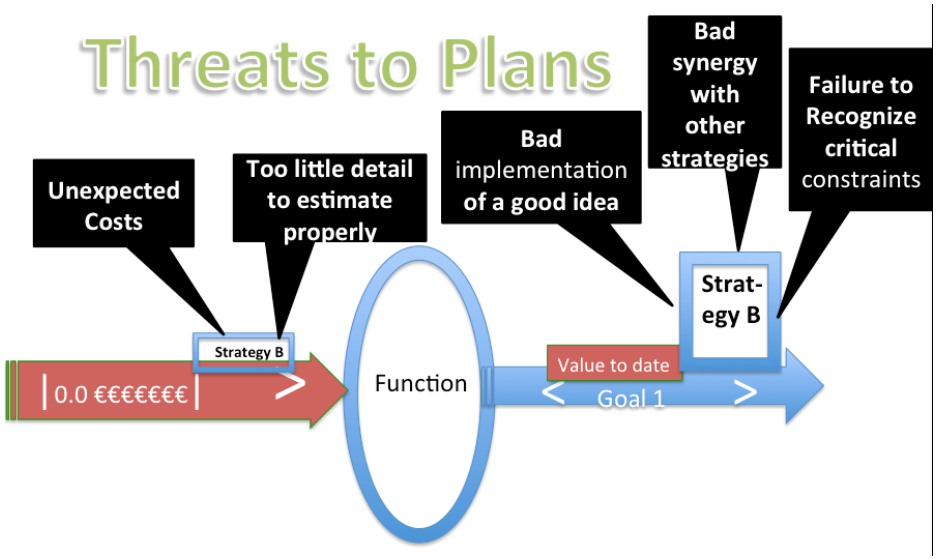
- One way to make large-and-complex specification clearer, is to promote global definitions: definitions which apply to either one project, or to one organization.
- Anybody can create and deploy global dictionaries or glossaries, and they do.
- This works even better, when you build it into a supporting app, as we normally do today.
- Here is a simple example of definitions, with the Term ‘Area’ (Capital A hints at formal definition): see the example in the illustration to the left.
- Of course these 2 Terms, function as Local Definitions, right *under* the Scale. But *one* of them is Global, as indicated by the colored hot link.



29. THE REQUIREMENTS TAGS, ENSURE REUSABILITY OF THE THE REQUIREMENTS, AND PARTIAL CAPTURE, IN THIS TABLE, OF CURRENT *STATUS* AND *WISH* LEVELS, AND SCALE.
THE 3 STRATEGY TAGS ENSURE REUSE OF THE STRATEGY DEFINITIONS.
THE NUMBERS IN THE CELLS, ARE REUSE OF *CONCEPT DEFINITIONS* OF 'SCALE IMPACT', '± IMPACT', 'UNCERTAINTY,' THE '0->' ICON IS REUSE OF THE 'VALUE SCALE ICON'. THE BAR CHART RECTANGLE IS REUSE OF THE 'DESIGN IMPACT EXTENT' RECTANGLE. THE LIGHTBULB ICON IS CONSISTENTLY REUSED IN THE APP.
BECAUSE IT SEEMS EASY TO GRASP AT A GLANCE.
THE UPDATED DETAILS OF OBJECTIVES AND STRATEGIES ARE ONE CLICK AWAY

29. Reusability: avoiding repetition, motivating quality definitions.

- If you **reuse definitions,** then there are some things that happen, which improve intelligibility
  - You can gradually become familiar with a Term, and get to know its specific definition from previous experience
  - A larger project group, or organization, can develop a higher degree of standardization, and common understanding
  - It will be worthwhile to create and maintain really high-quality definitions. Define once well, use good definition many times. The definition can be adjusted, as you get insight, that it needs more clarity or detail.
  - You can avoid the same Term having different interpretations in different parts of the plan.
  - Updating a common Term's definition, will upgrade the quality of *all current uses* of the Term's definition, and give better quality, when you *refer* to it, and *reuse* it, in the future.
  - You will avoid people claiming they had 'misunderstood' a term, when it is clearly '**M**arked' as a defined term (**C**apital **L**etters send a clear signal), with easy access to the definitions both locally, and via digital links. Plan writers are then 'ethically obliged' to know that they can, and must, check the current definition.
  - In Planguage, there are very many reusability devices: and the key to them all is the **Tag**, discussed above.
  - Designs and Requirements are 'reused' by reference to their Tags. There is no rewriting or paraphrasing of a definition, which could lead to misinterpretation. There is only one definition, it is always reused. And when it is edited in the future, all references to it are automatically upgraded, and in sync, because we use the same Tag.
  - *I once had a 26,000 employee client, in UK, building software for international corporate sale, with their hardware. The CEO of one of their largest customers called our CEO and complained about the bad application quality. I got the job of analyzing the root cause. Single ideas (like a 'requirement') were found in two dozen variations. Written, re-written, paraphrased, never quality controlled, for different purposes. Nobody understood anything the same way. The solution was reusability. One master definition for all. Simple, but nobody thought of it, until the shit hit the fan, and it got high-power attention.*
  - Re-usability is a Technoscope that makes complex plans easier to understand, easier to update and improve, and makes planners more motivated to invest, in writing more-intelligible plans, in the first place. **Write once, use many.**



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**

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

March 9-10 2019 Edit

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.

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 Behaviour**, 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.

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.

Risks: <*Name or refer to tags of any factors, which could threaten your estimated impacts*>.

R1. FCxx is delayed. Mitigation: continue to use Pxx<- tsg 2.12

R2: the technical **integration** of Px+ is not as easy as thought & we must redevelop Oribit

R3: the and or scalability and cost of **coherence** will not allow us to meet the delivery.

R4: **scalability** of Orbit team and infrastructure, first year especially <- BB. People, environments, etc.

R5: re Cross Desk reporting Requirement, major impact on technical design. **Solution not currently known**. Risk no solution allowing us to report all P/L

Issues: <*Unresolved concerns or problems in the specification or the system*>.

I1: Do we need to put the fact that we own Orbit into the objectives (Ownership). MA said, other agreed this is a huge differentiator. Dec 2.

I2: what are the time scales and scope now? Unclear now BB

I3: what will the success factors be? We don't know what we are actually being asked to do. BB 2 dec 20xx

I4: for the business other than flow options, there is still a lack of clarity as to what the requirements are and how they might differ from Extra and Flow Options. BB

I5: the degree to which this option will be seen to be useful without Intra Day. BB 2 dec

32. Dependencies: Making Sure we recognize things that have to be done.

- Often there are tasks, or results, or products, or services, that must be completed, before we can begin, or before we can complete something. These are ‘Dependencies’.
- It is useful to be very explicit about dependencies, so that we can double check in good time, whether they are ready for us to ‘depend on’, or to understand if there are any delays.
- It is not good enough in a large, fast changing, complex, distributed plan, that such dependencies are ‘taken for granted’ as ‘OK’. No problem! No! Problems!
- This dependency idea is of course fundamental to Critical Path Planning

<- 32. ILLUSTRATION. THIS IS A DOCTORED (XXX), BUT OTHERWISE REAL, SPECIFICATION FOR A DESIGN FOR A BANK. I INTERVIEWED THE DOMAIN EXPERT (BB) BY TELEPHONE AT HIS HOME. AND DELIVERED THIS TO THE TEAM, AT THE OFFICE, AFTERWARDS. YOU CAN SEE BY THE FIRST ROUND NUMBER OF ASSUMPTIONS, RISKS, ISSUES AND A DEPENDENCY THAT BB KNOWS A LOT OF INTERESTING STUFF ABOUT THE DESIGN IN QUESTION. THE DESIGN IS A MAJOR ARCHITECTURE ALTERNATIVE FOR MERGING 2 BANKING SYSTEMS.

Security Strategy:
Type: Top level security strategy.

Responsible Strategy Planner: Tom Gilb, Chief Planner, Sept 6 2018.
Security Architect: Steven Shad, of Yale Inc. Santa Monica.
Side Effects Analyst: Wendy Bartlett, Business Analyst.

Description: xxxxxxxx (the basis for analyzing impacts).

Supports: Security Objective.

Impacts: Usability, Training Costs, Terrorist Laws.

Risks: Unknown operational cost, Unknown Maintenance Cost.

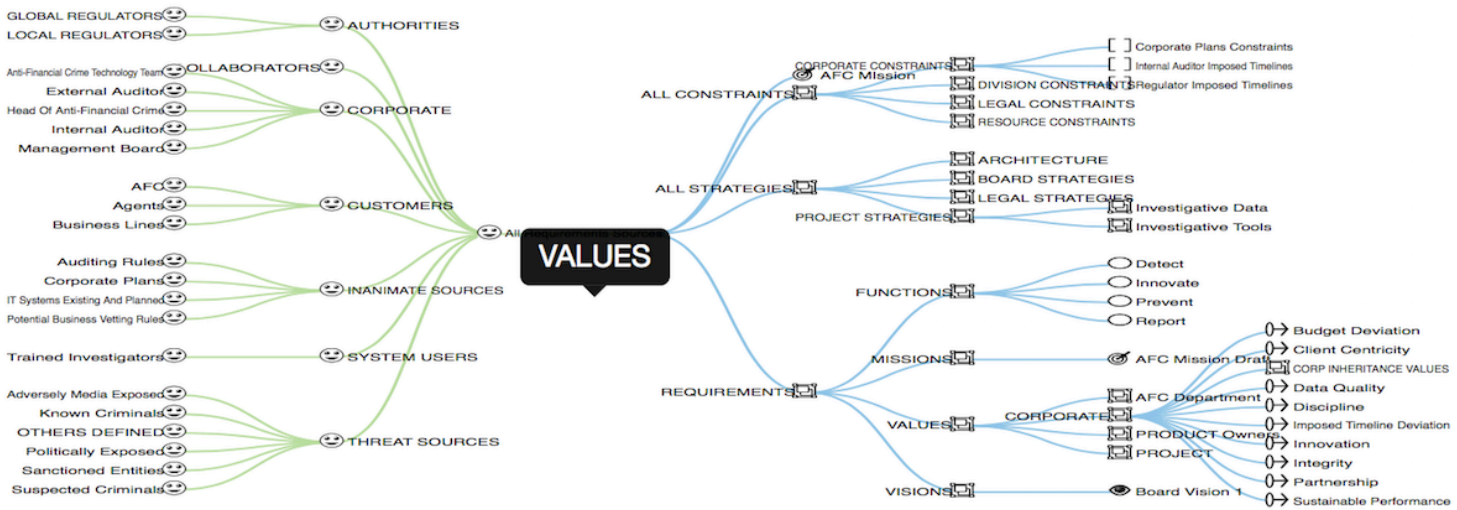
Issues: not thoroughly defined, no contract guarantees, effects not proven on similar projects.

33. EXPLICIT 'SUPPORTS' STATEMENT FOR A STRATEGY
BECAUSE THE TAGS AND PARAMETERS HAVE A CLEAR AND DEFINED MEANING,
WE CAN AUTOMATE 'DRAWING OF RELATION DIAGRAMS', 'SELECTING THE INTERESTING RELATIONS' ONLY.

Some ways of documenting various aspects of a strategy. Most everything is 'background' information about the strategy. Notice the 3 explicitly named responsibilities relations (line 3 to 5). Source: Value Planning, 2.2 B.

33. Supports: Making sure we recognize why things are there.

- ‘Supports’ is one of many relational parameters to describe a planning object.
- It is important to know how things are related and connected.
- If you are going to change one of them, the other might be affected, and you need to know explicitly where to look to find out the possible consequences.
- One frequently used method in planning notation is to draw arrows between boxes to indicate relationships.
- Our automated app (ValPlan.net) will to some degree automatically draw lines between related entities. Based on such parameters as these. But the relationships can get dense and messy, in complex plans. So the use of different parameters, and Tags means we can keep track of relations digitally, and draw maps and lines when we want to and *selectively*, to get rid of the line-relation noise. We think it is better to keep track of relationships this way, and let a computer draw diagrams as needed.
- Digital to Diagram is a smarter way to build Technoscope Overviews. Here is one example.



Contract Flexibility:
Type: Project level Critical Objective.
Owner: Project Manager.

Supports: CTO Objectives, especially Technical Adaptability.

Scale: The Speed which a **C**ontract can be **C**hanged at minimum cost of loss to reflect **C**ircumstances.

Goal: < 1 month.

Contract: All IT Services and IT Products.
Changed: Deleted or modified.
Circumstances: changed economics, or failure to live up to expectations.

Deadline: This Year.

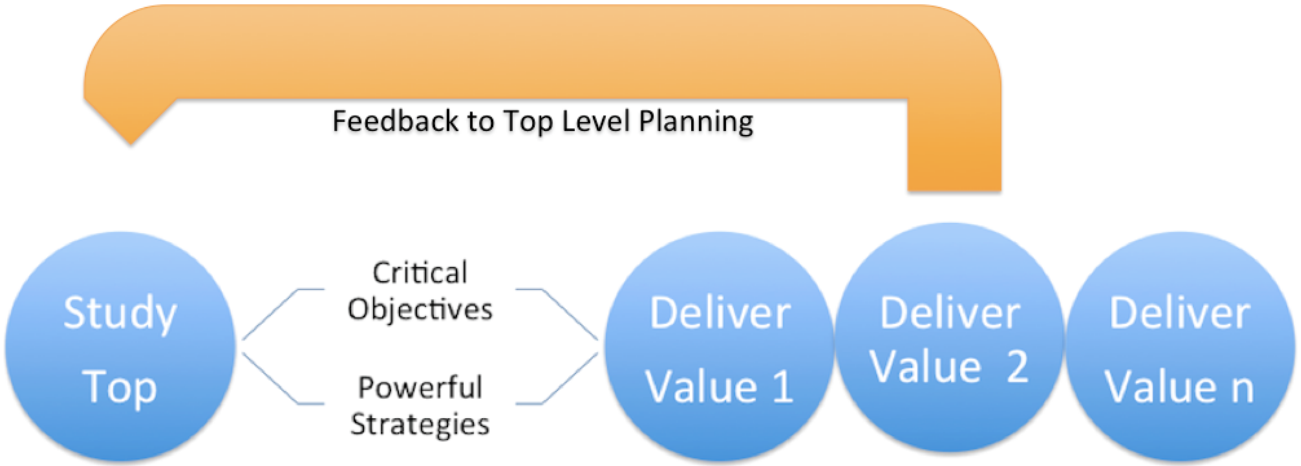
Supported By (Strategies):
FlexiCon: www.FlexibleContracts.com.

Supported By (Objectives):
Legal Dept: % of Flexible Contracts in Force.

34. TWO SIMPLE EXAMPLES OF 'SUPPORTED BY' STATEMENTS FOR AN OBJECTIVE
SOURCE: VALUE PLANNING Planguage Example 1.3 The mapping of many relationships for a single objective.

34. Supported By: Making sure we understand who our friends are.

- ‘Supported By’ is similar to ‘Supported’. Just the ‘other way’
- It documents anything we know is available, and responsible, in order to exclusively, or non-exclusively, **support** our specification object; our requirement, or our design
- However, it is just a *general* notion of support, and there are many other more-specialized parameters for ‘support’ in the examples to the left (Contract Flexibility) and above it (Security Strategy)
- For example the ‘**Owner**’ is the official ‘Specification Owner’ for *that particular one specification object* (Contract Flexibility). Not the entire plan set (all planning objects for a project).
- We can use this, *specialized* supporting relation (‘Specification Owner’), to, for example, make a diagram, or report, of all specification objects, in this project, or all projects, which are ‘Owned' by a particular ‘Spec Owner’ because:
  - They (one Owner) might like to get an overview of their personal current responsibilities
  - Maybe their responsibilities have to be transferred to others, as they have moved.
- It is worth noting that although we are documenting that there *is* a ‘relationship, between two entities: we do not yet know *how deep*, and *how rich* the relationship is. Something we are going to do below (VDTables)
  - We do not know the degree, or strength of the relationship (a point ‘diagrams’ alone, seem to miss)
  - We do not know *how many types* of simultaneous relationships there are, between the entities. For example, there may be a variety of qualities and costs in the relationship.
  - We are going to need even more-advanced Technoscopes to specify this type of information. Coming up!

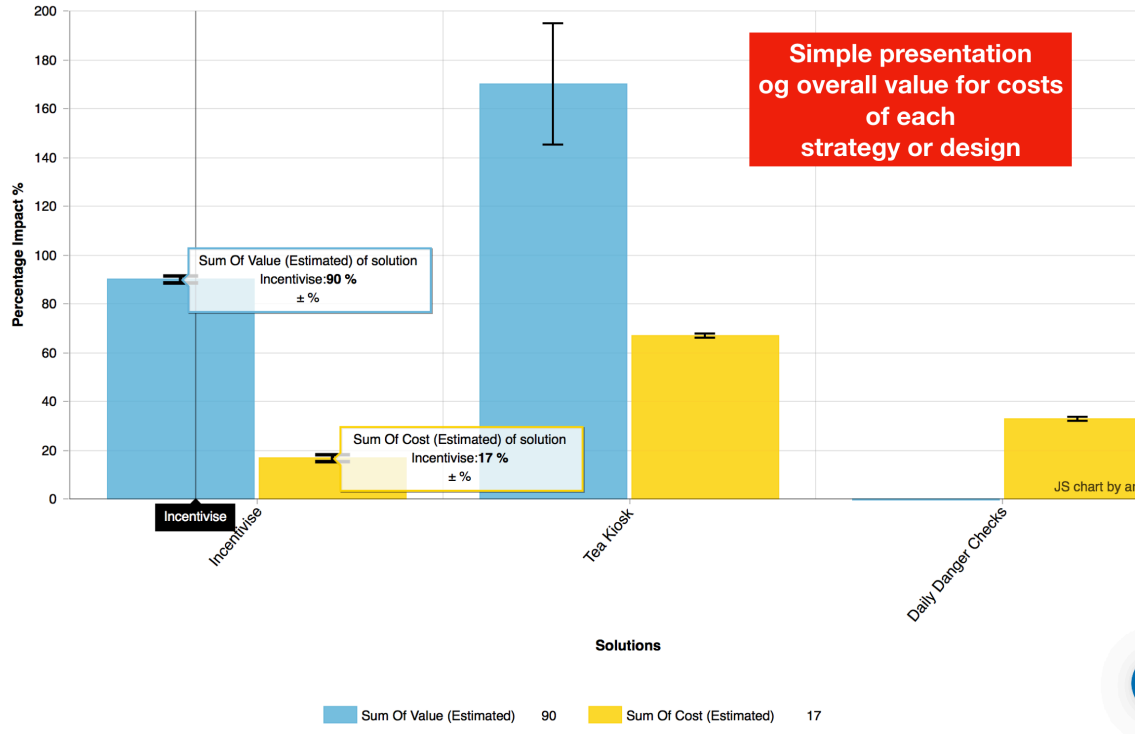


**35. AFTER A VERY BRIEF (1 WEEK) DETERMINATION OF TOP LEVEL OBJECTIVES AND STRATEGIES,
WE NEED TO START THE VALUE-DELIVERY CYCLES.
BUT WE CAN LEARN FROM THESE CYCLICAL EXPERIENCES,
AND FEED BACK FACTS, TO TOP LEVEL PLANNING.**

Source. Value Planning, 2.8.

35. Decomposition Principles: detailing strategies so you understand them better.

- We did a simple discussion of this subject above, in ‘Independence of Implementation’.
- There is more to the decomposition of strategies, designs, and architecture, that I’d like to add now.
- There are a large number of possible decomposition criteria, which might be interesting at any give time.
- For example; Decompose by *political sensitivity*, decompose by *security classes*, decompose by *privacy concerns*, decompose by *economics*, decompose by *return on investment*, and so on endlessly.
- But there is one ‘primary general decomposition rule’ that I would like to elaborate on here. It is a good set of rules for Agile project management, where we are going to deliver small selected prioritized increments of strategy, in an effort to build a rich and continuous stream of measurable results, for our prioritized stakeholders. That means to make people happy fast, for sure.
- We need to decompose strategy ideas, by the following rules, at the same time:
  - **Independence**. Ability of the sub-strategy to be implemented, before any of the other remaining, 'unimplemented yet’ strategies
    - As discussed in some detail above in *Independence of Implementation*
  - **Result Delivery.** This is the ability, when implemented in real life, incrementally to a system, or product, or service, to **deliver measurable**, and expected as estimated, **value** (quality improvements, cost and time savings etc.) either directly, or as knock-on consequences. If you cannot do this, then:
    - You are wasting your time
    - You do not know what you are doing
    - You cannot get any useful feedback, except that you are totally failing.
  - **Suitable Size Effort**. The ‘value delivery steps’ as we call them, need to be small enough to do, in a short time (a week or so), but large enough to make a real difference (non-trivial result). This delivery cycle size is roughly 2% of the entire project effort, or 50 weekly increments in a year’s project, or 48 monthly cycles in a 4 year project.
    - Some good reasons for the 2% size is that it is big enough to *make a difference in results*, and it is short enough to *keep people’s attention focussed*, and *small enough to ‘lose totally’* (2% of 100%) without destroying all credibility and resources.



Overall ‘Potential Values / Costs’
of 3 *options* or (if you need them all)
complimentary ‘benefit drivers’ = strategies = solutions = means’

36. HERE IS A TECHNOSCOPE FOR ENVISAGING THE COST-EFFECTIVENESS OF EACH SOLUTION THE BAR CHART IS COMPUTED BASED ON QUANTIFIED REQUIREMENTS AND COSTS, AND RELATED ESTIMATES THAT WE MADE, FOR EACH SOLUTION’S IMPACTS ON THE REQUIREMENTS

36. Design Principles: Technoscopes for seeing relevant design emerge.

Here are some Principles of 'Design Transparency': making sure people understand what is going on, and if it is useful.

1. All design must *directly relate* to all affected requirements: connections should be explicit, and well documented.
    1. Design must satisfy *some* value requirements, at least: if not, get rid of them
    2. Design cannot have 'too' destructive side-effects on other value requirements: if so, re-design
    3. Design must not, singly or in total, violate constraint requirements: constant Contraint awareness.
2. Design can, and should be, *estimated* before selection, at least with enough effort and accuracy, to avoid wasting time and effort, and getting locked into bad designs.
    - *Shall we say 1 to 8 hours of estimation for every week of design implementation*
3. Design risks and uncertainties, need to be *documented* and evaluated, before selecting design increments.
4. Designs shall be piloted on a small scale, measured 'for real', and scaled up only when successful.
5. Designs should be designed, so that we are **not** *locked into them* in any way, if they fail.

- **What happens if you 'haven't got time for this (it takes a day)?**
    - You will **fail** to get expected value for money.
    - You may **fail totally** and embarrassingly (you will find someone else to blame, of course).
    - You have wasted your time with this book, and these ideas.
    - I think you are wasting a great professional opportunity
        - But I won't be there to tell your boss. But maybe your boss reads books like this ?
    - Most projects have nowhere-like this level of 'engineering' discipline:
        - so you will have the *same failure rate* as all the others (horrendously high, Google it). Comforting?
    - You will have missed an opportunity to 'look like a genius', to 'further your career', get rich and be happy. :)
        - Most people are not interested in putting in the effort to be superior. By definition.
        - But I can hope that some readers, and leaders, and top managers are ambitious enough to be more disciplined about planning. Try out some of these ideas and see if they work for you! You do not have to do it all at once.
- **More detail on how we implement these Technoscopes Design Principles above, in the *following* section below.**

Design Specification Template <with Hints>

Tag: <Tag name for the design idea>.
Type: {Design Idea, Design Constraint}.

========== Basic Information ==========
Version: <Date or version number>.
Status: <{Draft, SQC Exited, Approved, Rejected}>.
Quality Level: <Maximum remaining major defects/page, sample size, date>.
Owner: < Role/e-mail/name of person responsible for changes and updates>.
Expert: < Name and contact information for a technical expert, in our organization or otherwise available to us, on this design idea>.
Authority: <Name and contact information for the leading authorities, in our organization or elsewhere, on this technology or strategy. This can include references to papers, books and websites>.
Source: <Source references for the information in this specification. Could include people>.
Gist: <Brief description>.
Description: <Describe the design idea in sufficient detail to support the estimated impacts and costs given below>.
<Term Tag here>: Definition: <Use this to define specific terms used anywhere in the specification>. "Repeat this for as many definitions as you need"
Stakeholders: <Prime stakeholders concerned with this design>.

========== Design Relationships ==========
Reuse of Other Design: <If a currently available component or design is specified, then give its tag or reference code here to indicate that a known component is being reused>.
Reuse of This Design: <If this design is used elsewhere in another system or used several times in this system, then capture the information here>.
Design Constraints: <If this design is a reflection of attempting to adhere to any known design constraints, then that should be noted here with reference one or more of the constraint tags or identities>.
Sub-Designs: <Name tags of any designs, which are subsets of this one, if any>.

========== Impacts Relationships ==========
Impacts [Functions]: <List of functions and subsystems which this design impacts attributes of>.
Impacts [Intended]: <Give a list of the performance requirements that this design idea will positively impact in a major way. The positive impacts are the main justification for the existence of the design idea!>.
Impacts [Side Effects]: <Give a list of the performance requirements that this design idea will impact in a more minor way, good or bad>.

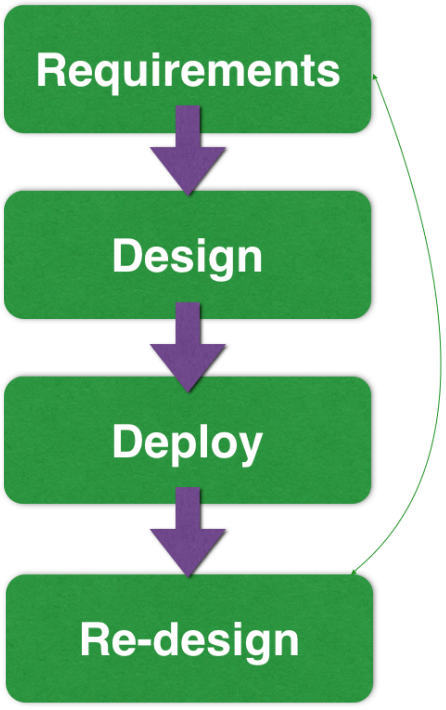
37. A TEMPLATE WITH <HINTS> (TOP HALF), SOURCE: [CE]

37. Templates (with hints): a simple Technoscope to instruct people 'on the spot'.

- At McDonnell Douglas (Now Boeing) we found that engineers made too many mistakes, when just filling in an Engineering Order form.
- The detailed instructions, for filing the form, were in a thick Engineering Handbook, and in forgotten training courses.
- So we 'moved the instruction directly into the form' (in the format of <Hints>), and it was immediately a success.
- What you need to know is there, when you need to know it.
- It is easy to improve the <hints text> when experience shows it is not good enough
- This works well in a simple paper form, but more likely in a PC text processor
- In our more sophisticated apps (valplan.net) we have applied the principle in the form of a greyed text.

The Basic Design Steps Logic: a summary

1. Environment Scope helps identify stakeholders.
2. Stakeholders have values and priorities
3. Values have many dimensions
4. Stakeholders determine value levels
5. Design hypotheses should be powerful and efficient ideas, for satisfying stakeholder needs
6. Design hypotheses can be evaluated quantitatively, with respect to all quantified objectives and resources
7. Designs can be decomposed, to find more efficient design subsets, that can be implemented early
8. Designs can be implemented sequentially, and their value-delivery, and resource costs, measured
9. Designs that unexpectedly threaten achievement of objectives, or excessive use of resources, can be removed or modified.
10. Designs that have the best set of effects on objectives, for the least consumption of limited resources, should generally be selected for early implementation.
11. A design increment can have unacceptable results, in combination with previous increments, and they, or it, might need removal or modification
12. When all objectives are reached, the process of design is complete: except for possible optimization of operational resources, by even-better design.
13. When deadlined and budgeted implementation-resources are used up, it might be reasonable to negotiate additional resources; especially if the incremental values are worth the additional resources.
14. When deadlined and budgeted implementation-resources are used up, it might be reasonable to negotiate additional resources; especially if the incremental values are worth the additional resources.



The Logic of Design: Design Process Principles.
Tom Gilb, 2016, Paper.
http://www.gilb.com/dl857

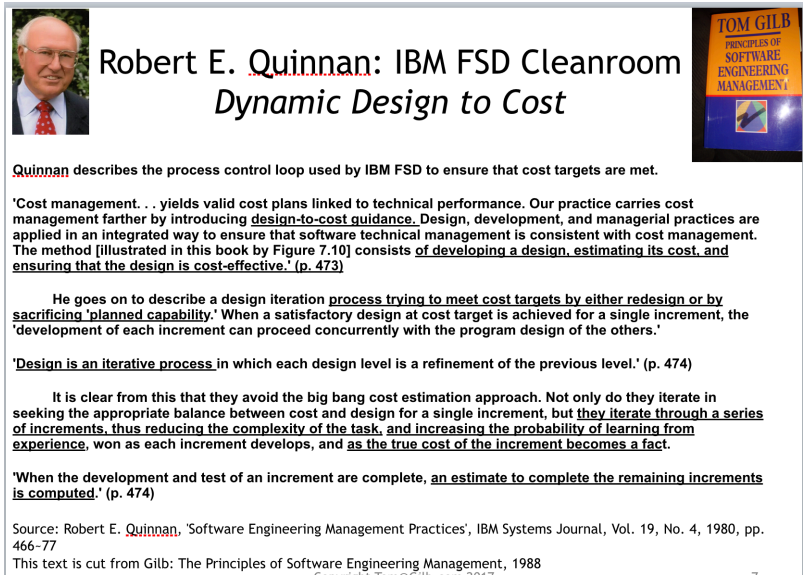
38. Design Logic Steps

38. Design Logic: Technoscopes for seeing what you have to do and when

- We all know that we need to do requirements, before design, and design before deployment of the design.
- But how many people are conscious of the fact that they need to do *stakeholder analysis* before understanding and finding *requirements*?
- How many people are very aware of all 14-steps in the 'Design Steps Logic' list to the left?
- How many people dive in, with their strategy or design, without really thinking about *all* of their many requirements, or without really clarifying them (like 'quantifying all qualities' for a start?
- OK, my point is that, most people are not as good as they *can* be in this area, and this checklist of 'logical design steps' might be a Technoscope, to help you and others, see that, 'some vital steps are missing here!'
- You might be able to use this checklist, helped by the corresponding paper[1], as a way to point out that your organization, or your project planning, *can* be improved.

[1] gilb.com/DL857 TheLogic Of Design

See [2]



Robert E. Quinnan: 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 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)

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

What critical numeric improvements do stakeholders need?

We can, and must always, express *their* values with *well-defined* numbers

Define both failure and success numerically

and

keep learning what those critical numbers are continuously

39. Design to Objectives

[2] Mills, H. 1980. The management of software engineering: part 1: principles of software engineering. IBM Systems Journal 19, issue 4 (Dec.):414-420. http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan

39. Design to Objectives: constant iterative refocus on Values, and changing Values, when developing designs and implementing designs

- Design, or ‘strategy planning’, or ‘architecture, or ‘problem solving’ (same things essentially) must keep a continuous, iterative focus on a ‘fluid (changing) set of value-objectives’, and changing constraints.
- We need to continuously ask: are our design ideas *really* moving us, in *practice*, towards the *current* updated requirements (objectives, constraints).
- Adopting a particular design or strategy, *once* - is not enough.
  - The real-and-valid requirements will *inevitably* change, even *during* your project:
    - is the design still good enough then?
  - New technologies, economics, and markets, will change the *possible* selection of designs, *during* a project
  - **Feedback,** from *agile incremental delivery* of sub-designs will tell you that:
    - The design is not as good as you hoped, for primary requirement values.
    - The design has some unforeseen side-effects, that are not tolerable.
    - The design has costs, that are unacceptable.
    - The design violates some constraints, and some new unknown stakeholders are making a fuss about it
- You have to learn what is true, as you go along. Nobody knows these things for sure in advance.
- Keeping this focus has to be done using *quantification*, *feedback*, and *measurement*: a formal continuous process.
  - This ‘**Design To Objectives**’ is the Technoscope: the tool for dynamically keeping track of success and failure.
- Most people do not have that disciplined process. Most people fail too much, too often.

Oslo Opera House requirements (imagined, for example)

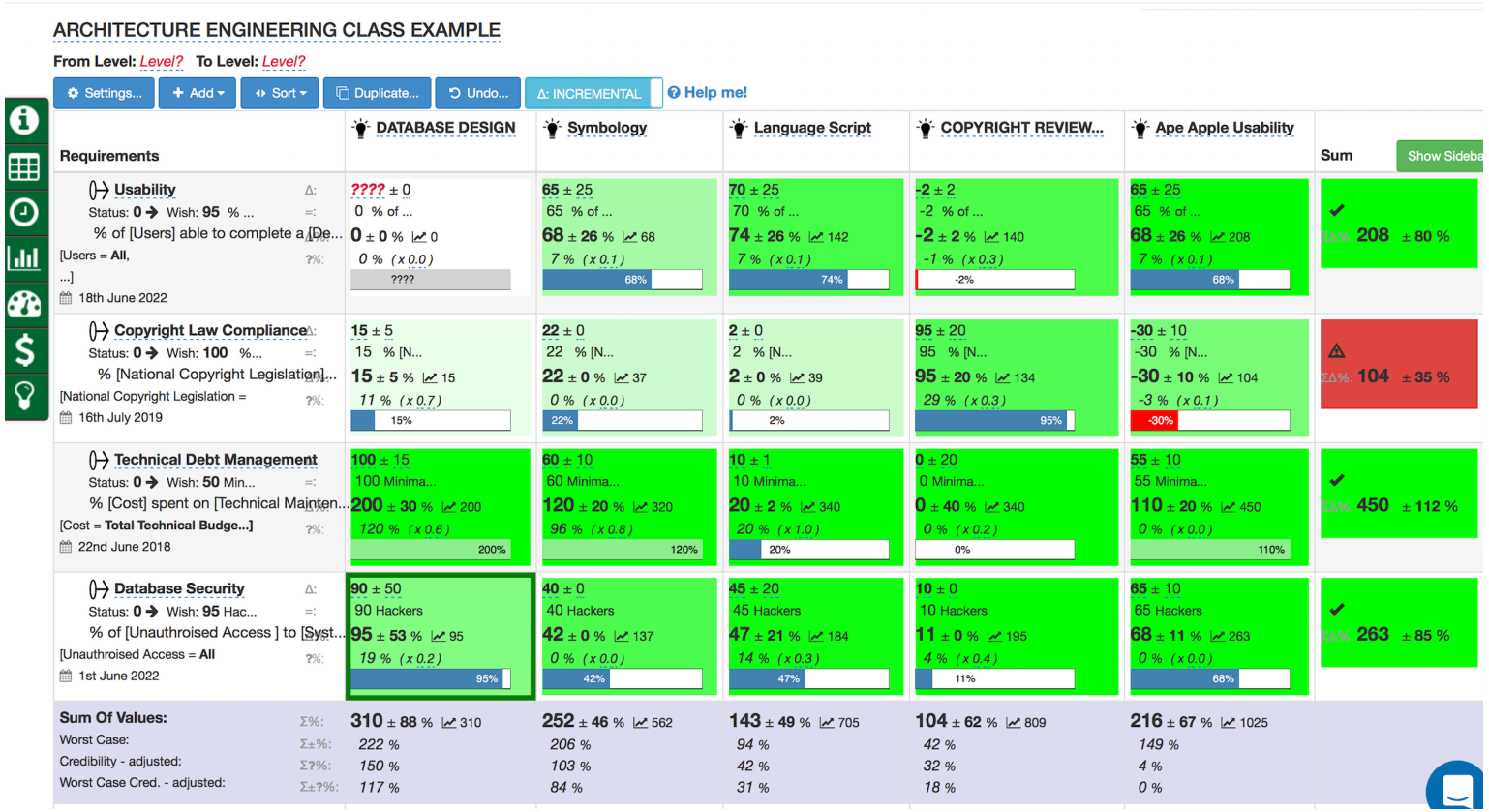
- Qualities
  - Impressive
  - Acoustics
  - Flexibility
  - Extendibility
  - Integratedness
  - Performance Visibility
  - National Symbol
  - Access to Fjord View
  - Comfort
- Costs
  - Building
  - Maintenance
  - Operational manpower
- Constraints
  - Legal Building
  - National Architecture
  - Archeological Site
  - Local Materials
  - Local Labour

40. SOME OF THE OBVIOUS MULTIPLE ATTRIBUTES OF AN OPERA HOUSE.

40. Multiple Attributes: a more-complete picture of reality

- **Nothing** can be managed or evaluated successfully on any *single* dimension of value, quality, function, or costs, *alone*.
- You *have* to learn how to juggle a **large number of factors**, at least 10, at the same time. Sorry!
- Juggling many balls in the air, at the same time is difficult, takes practice, and needs suitable methods.
- We humans resist this juggling, because we *feel that* we cannot cope with it successfully.
- The *real* problem is that you don't know 'the juggling methods', and have not *practiced* like all jugglers obviously have to do. Does 10 thousand hours of practice to *master* something scare you? Lower your ambition level.
- The inherent problem is that if you are *always* dealing with a system, with about 10 critical objectives, and if 1 single one of these 'balls' falls down, then you probably have a *total system failure.*
  - If your brain stops working, it does not much matter that your heart and lungs are working fine.
- To succeed in the real world, in almost anything, you have to be good at keeping you eyes on many balls in the air.
- You need Technoscopes to help you. Your eyes, ears, and brain are a good start: but ways to 'model' the balls moving on the air need to be added to your Technoscope tools.
- More detail follows, and a pretty good set of Technoscopes has already been discussed above.
- You do not have to spend 10 thousand hours to get practical use from Technoscopes. Not everyone needs to be a master.
  - You may have noticed that many of these Technoscopes can be of practical use: 'the same day'.
    - But being good at *all* these Technoscopes might take some more effort, weeks, months, years: but it feels *so good* to pick up these practical tools, and *use them on an everyday basis*. They are like 'superpowers'.

PART 3. A VALUE DECISION TABLE (VDT)

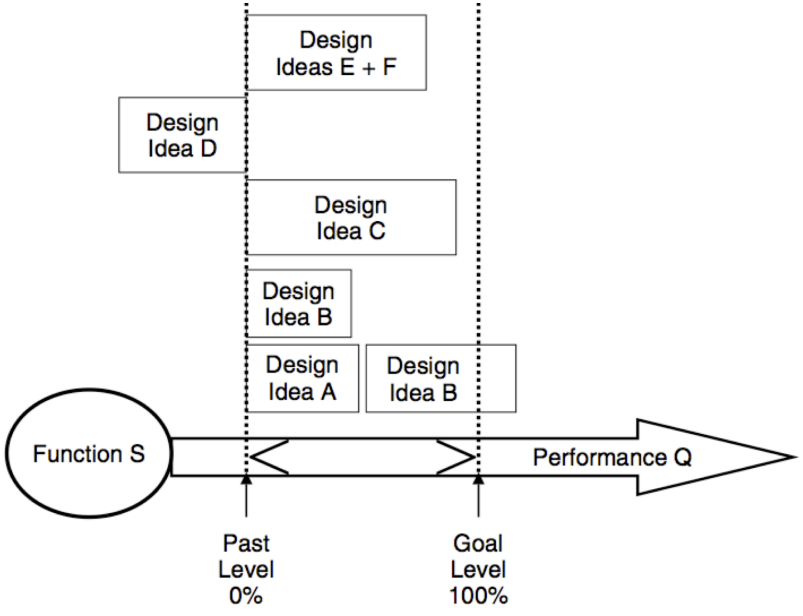


PART 3: A 'VALUE DECISION TABLE' CAPTURES ESTIMATES: AND LATER IT CAPTURES 'MEASURES' OF THE IMPACT, WHICH STRATEGIES HAVE, ON REQUIREMENTS (VALUES AND RESOURCES)

March 9-10 2019 Edit

Part 3: Value Tables (Value Decision Process)

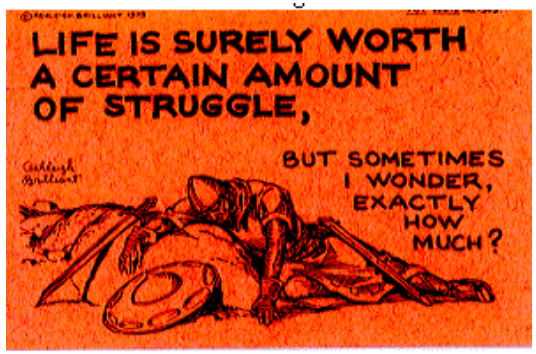
41 Impact Estimate
42 Impact Measurement
43 % Impact
44 ± uncertainty
45 Resource Estimates
46 Value Estimate Sum
47 Resource Sum
48 Values to Resource Ratio
49 Prioritization Options (Static, Next Step)
50 Feedback
51 Deviation Analysis
52 Safety Factor
53 App: Needs And Means = ValPlan
54 Bar Charts
55 Modeling versus Measurement
56 Time, Micro Deadlines
57 Long Term Short Term
58 Side effect Analysis
59 Risk Prevention
60 Evidence and Credibility



Source [CE]

41. WE CAN ESTIMATE THE DEGREE TO WHICH A DESIGN WILL HELP US PROGRESS TOWARDS OUR GOALS NEVER EXACTLY,

BUT WELL ENOUGH TO DECIDE WHICH DESIGNS ARE WORTH DOING FIRST[3]



[3] All PotShots in this book are used by paid contractual permission from the © author. Ashleigh Brilliant.

41. Impact Estimate (IE): how good is your idea, exactly?

- We always expect *some* value from our investment in designs or strategies
- It does matter a great deal to us, whether the value delivered is trivial, enough, or overwhelmingly great
- We can of course 'just do it', and see what happens.
  - But of course, we might well waste time and money, and our credibility
- So the smart thing is, to get a *reasonable idea,* of what we can expect, before we 'take the plunge'.

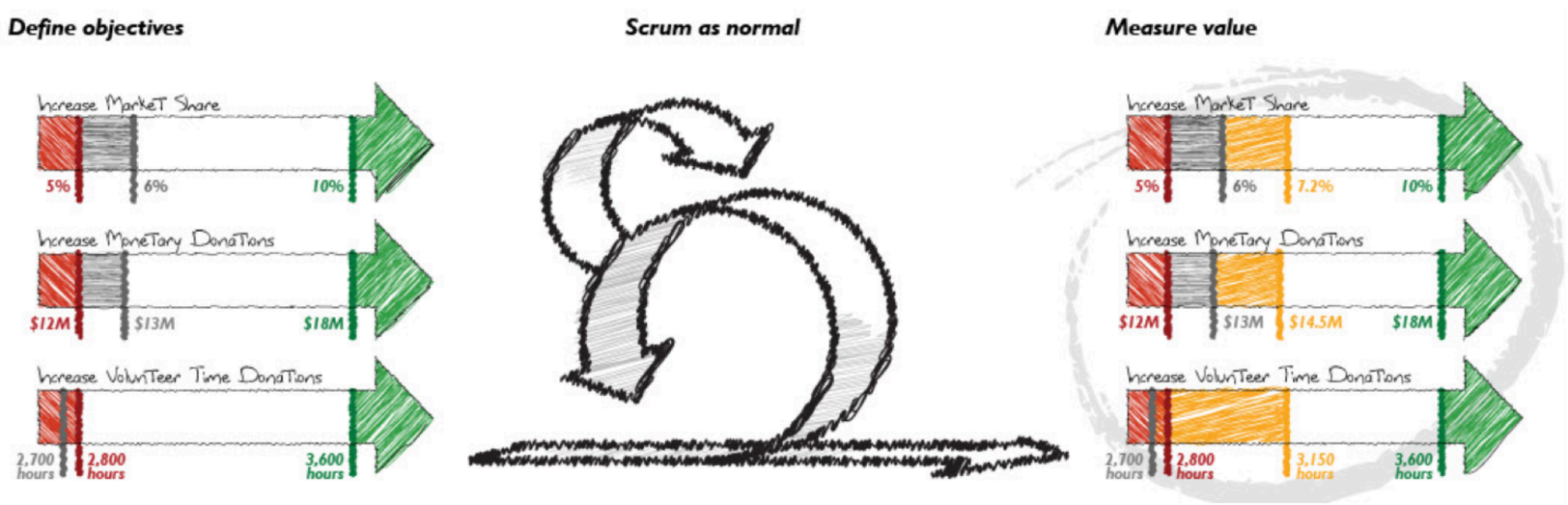
- I call this 'Impact Estimation'

- This means putting a number on how great your ideas going to be.
- Not just comforting words, like *'It will be fantastic. Everybody just loves it'*
- Estimating, especially about the future, is difficult, as our weather-people are well aware.
- But, they try every day, every hour, every cyber second, to 'predict'.

- If we are serious about our work and our projects, we have to try too.
- We have to put our opinions and experiences into numbers: *words are not good enough.*

- We do *not* have to get it right, not *exactly* right.
-
- But:
  - We should mostly get the right 'order of magnitude' (factor of 10)
  - If we cannot do *that*, we do not deserve to do this difficult job.
  - Decisions can just as well be made by random lottery, or by innocent children

- The IE idea is pretty simple. But it does require you to have quantified your *value objectives*, first.
- Then you can Estimate the Impact (degree of change you will get, from deploying your idea) in practice.
- You can give a real number on your scale of measure (like 'improvement 3 seconds')
- And sometimes we express progress as a %. 100% means 'all the way tour goal level, *on time*.



42. Source: Ryan Shriver, 'Measurable Value with Agile. in Overload Journal February 2009. https://accu.org/index.php/journals/1534

42. Impact Measurement.

- Guessing ‘what impact’ a design will have, is useful. But what *really* counts is the *real* result.
- So when the idea, or a trial-size part of it, is implemented, we need to measure, as best we can, the real effect.
- The Real measure tells us if we are on the right road, and can ‘scale up’, or if we need to dump, or adjust the delivery idea that disappointed us.

Requirements	ProductDesign	Financials	MarketingStrategy	DistributionMethod	Sum
Demographic Past: **0** → Wish: **50** %	20 ± 5 % 0 Δ%: **40** ± **10** % 40	27 ± 5 % 0 Δ%: **54** ± **10** % 94	23 ± 3 % 0 Δ%: **46** ± **6** % 140	10 ± 0 % 0 Δ%: **20** ± **0** % 160	Σ Δ%: **160** ± **26** %
Millionaire Past: **1** → Wish: **1000000** $	450000 ± 150000 $ 0 Δ%: **45** ± **15** % 45	400000 ± 100000 $ 0 Δ%: **40** ± **10** % 85	100000 ± 50000 $ 0 Δ%: **10** ± **5** % 95	200000 ± 100000 $ 0 Δ%: **20** ± **10** % 115	Σ Δ%: **115** ± **40** %
MarketSegment Past: **4** → Wish: **1** Market Rank	1 ± 1 Market... 0 Δ%: **100** ± **33** % 100	4 ± 1 Market... 0 Δ%: **0** ± **33** % 100	2 ± 1 Market... 0 Δ%: **67** ± **33** % 167	3 ± 1 Market... 0 Δ%: **33** ± **33** % 200	Σ Δ%: **200** ± **132** %
Geography Past: **0** → Wish: **100** %	5 ± 5 % 0 Δ%: **5** ± **5** % 5	10 ± 4 % 0 Δ%: **10** ± **4** % 15	40 ± 5 % 0 Δ%: **40** ± **5** % 55	30 ± 5 % 0 Δ%: **30** ± **5** % 85	Σ Δ%: **85** ± **19** %
Market Past: **0** → Wish: **100** %	40 ± 10 % 0 Δ%: **40** ± **10** % 40	5 ± 3 % 0 Δ%: **5** ± **3** % 45	40 ± 10 % 0 Δ%: **40** ± **10** % 85	20 ± 5 % 1 Δ%: **20** ± **5** % 105	Σ Δ%: **105** ± **28** %
Sum Of Performance:	Σ%: **230** ± **73** % 230	Σ%: **109** ± **60** % 339	Σ%: **203** ± **59** % 542	Σ%: **123** ± **53** % 665	
TimeToMarket Past: **1** → Wish: **8** Weeks	2 ± 0.5 Weeks 0 Δ%: **14** ± **7** % 14	2 ± 0.5 Weeks 0 Δ%: **14** ± **7** % 28	3 ± 0.75 Weeks 0 Δ%: **29** ± **11** % 57	4 ± 1 Weeks 0 Δ%: **43** ± **14** % 100	Σ Δ%: **100** ± **39** %
ShowMeTheMoney Past: **0** → Wish: **5005** £	1200 ± 200 £ 0 Δ%: **24** ± **4** % 24	205 ± 200 £ 0 Δ%: **4** ± **4** % 28	2100 ± 500 £ 0 Δ%: **42** ± **10** % 70	1500 ± 0 £ 0 Δ%: **30** ± **0** % 100	Σ Δ%: **100** ± **18** %
Sum Of Resources:	Σ%: **38** ± **11** % 38	Σ%: **18** ± **11** % 56	Σ%: **71** ± **21** % 127	Σ%: **73** ± **14** % 200	
Performance To Cost:	6.05	6.06	2.86	1.68	
Ratio (Worst Case)	3.20	1.69	1.57	0.80	

43. IN EACH VALUE DECISION TABLE CELL, THE UPPER NUMBER IS THE 'REAL ONE' (20±5) ON THE DEFINED SCALE. THE LOWER NUMBER IS A COMPUTED %. (40±10%)
NOTICE THERE ARE 2 TYPES OF 'SUMS' (PERF., RESOURCES) AND ALL ARE IN %. MORE BELOW

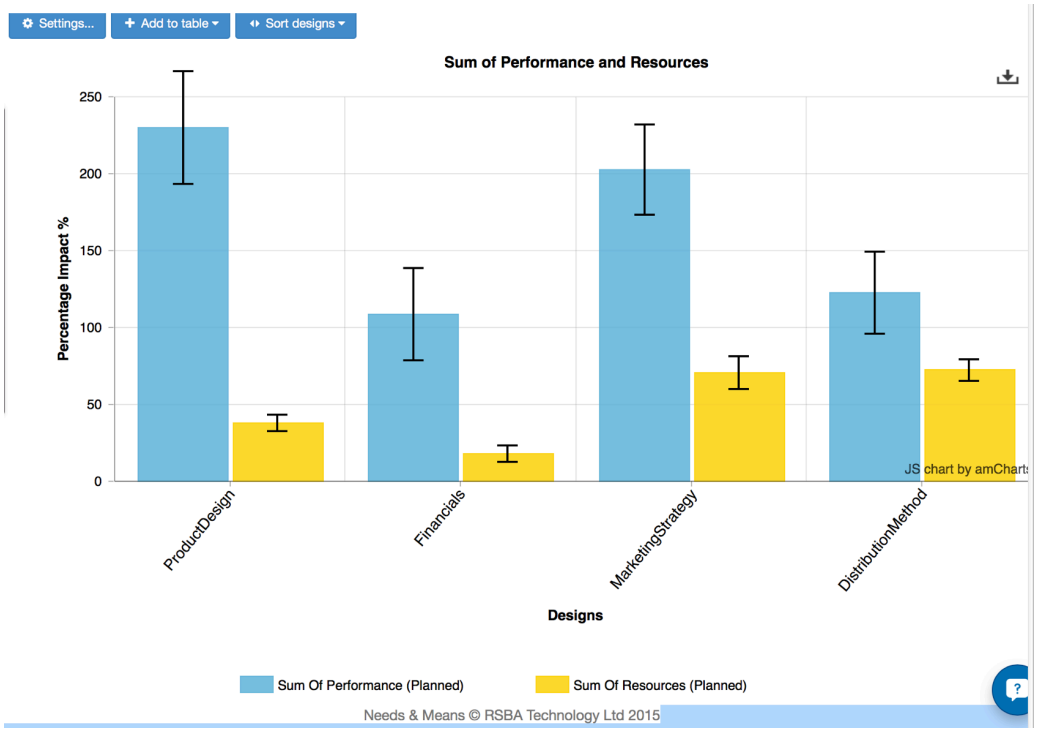
Source Geoff Cooper, UK, Startup Plan, 2015

43. % Impact: A relative-to-goal measure.

- There are two useful *alternative* ways to express the idea of ‘how much effect’ a design has.
- The ‘real’ number is a number on our defined Scale of measure. For example, in ‘seconds’, or ‘number of people’.
- A real number allows us to present results in a language that domain specialists understand easily.
- But we have found that it is very useful to have a ‘scale-neutral’ way of expressing the impact idea.
  - ‘% Impact’ is simple.
    - 0% means ‘no change’ from a defined benchmark.
    - 100% means ‘all the way’ to a selected Target (usually a Goal), on time.
    - All other numbers -50%, +50%, 150% are relative to these two points.
    - The % is computable, automatically, from the ‘real’ numbers. Like Celsius to Fahrenheit

You might well say ’why bother’ with the %?

- it turns out it is a Technoscope tool.
- It allows us to ‘add up impacts’ (in %) from quite different ‘Scales of Measure’.
- And that allows us to get an *overview* of the ‘bigger picture’. The ‘Overall view.
- And that is what we expect from a good Technoscope
- More below see: Value Estimate Sum, Resource Sum, Values to Resource Ratio.

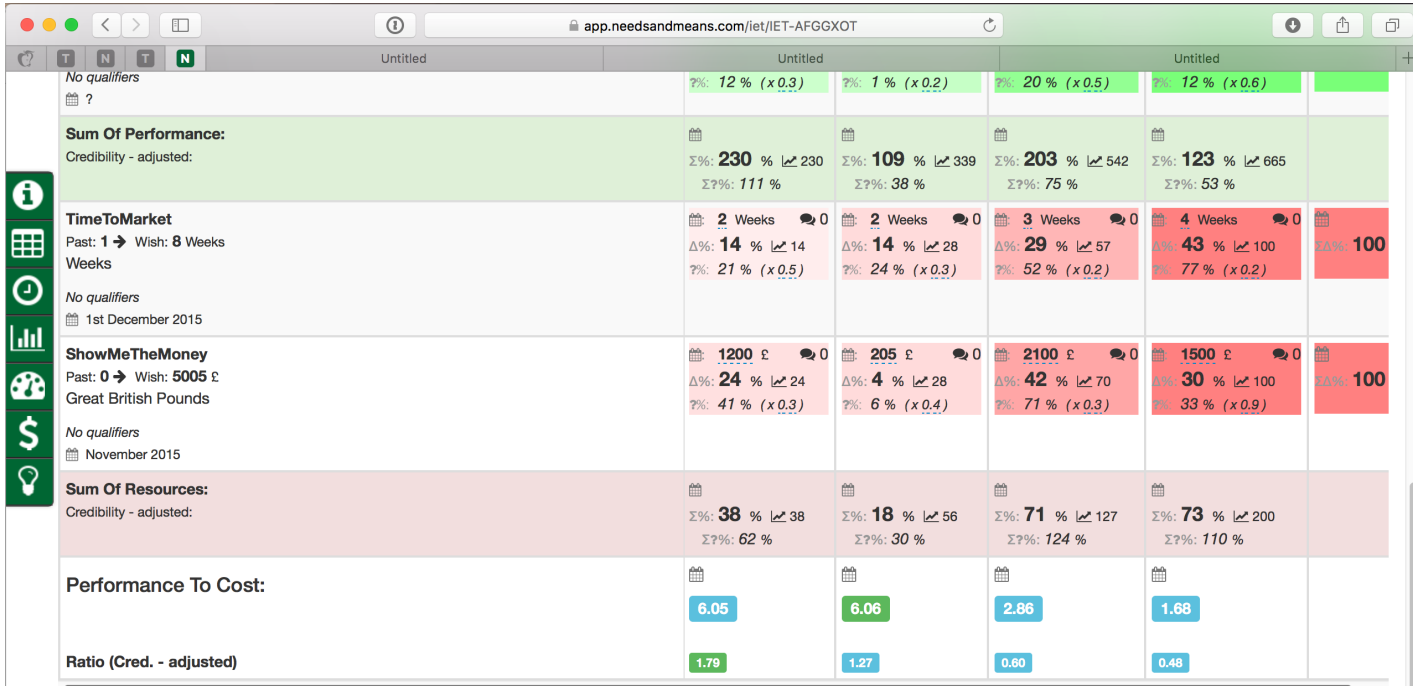


44. A TECHNOSCOPE FOR UNDERSTANDING THE 'SUM OF WORST-CASE POSSIBILITIES' FOR VALUES AND COSTS (THE I BAR) INDICATES WORST LEVEL.
THIS CHART IS GENERATED FROM THE SPREADSHEET ABOVE, BY COOPER.

Source Geoff Cooper, UK, Startup Plan, 2015

44. ± Uncertainty: a Technoscope for seeing uncertainty, and the worst case.

- We need Technoscopes to help us deal with risk. And there are many different tools for that, here.
- I am a strong believer that nobody can estimate 'exactly'. So the thing I ask everyone to do, is 'to add a ± number' to their estimate, to indicate 'how bad it might be', and 'how good'.
- Denying that this is needed, is like saying '±0' - wanna make a bet on that?
- This '±' Technoscope gives us a window, on uncertainty, on possible disappointment and risk.
- The ± number should reflect the range of *experience*, for that value or cost, with that Design.
  - What is the lowest number observed?
  - What is the highest number observed?
- Another term for this is Evidence-Based Decision Making. Or would you prefer 'wild guesses' for *your* career?
- You might object, I can hear you as I write this 'But I don't have such data!'
  - Probably, you do not have ideal facts, not for most things
  - But it is amazing what you can dig up, by an internet search, or by asking a real expert, or even asking the Design Product/Service Supplier, who has often done this, all over the world, for years. Surely they have some facts?
  - And if you cannot dig up facts and numbers?
    - Write it down as an Issue or a Risk.
    - Estimate a very large ± like '99±90', which will
      - Make proponents of the design angry enough to dig up some facts.
      - Allow us to compute a very bad worst case (99 - 90 = 9) and radically lower the priority that this design will be awarded, when we do a 'worst case' analysis (quite automated in spreadsheets and our apps).



45. AN EXAMPLE OF CONSIDERING 2 LIMITED RESOURCES, AND ADDING THEIR EFFECTS AS A % OF BUDGET SUM. THE VALUE DECISION TABLE IS TRYING TO LOOK AT ALL VALUE IMPACTS OF ALL STRATEGIES, AND THE SUM, AT THE BOTTOM, IS A TECHNOSCOPE FOR GETTING *SOME* IDEA OF THE COMBINED EFFECT (VALUES FOR RESOURCES).

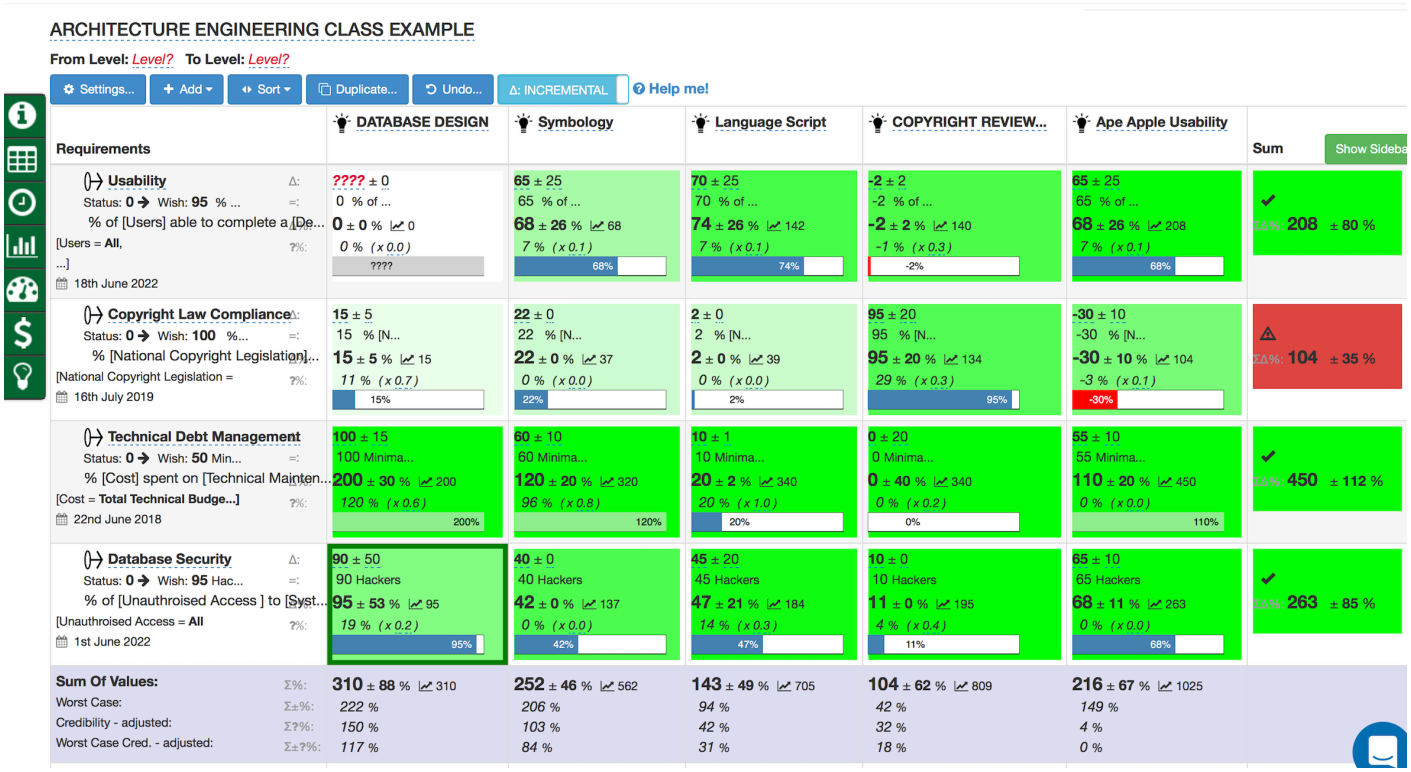
45. Resource Estimates: how to understand the possible costs.

- I find it amazing how often people planning something, do not *really* ask about the costs of their plans.
  - I guess they are spending *someone else's money* (like your taxpayer money, or their employer's or client's)
- But, it is not as simple as that, 'the up-front financial cost' or 'CapEx'
- There are *many* cost concepts we need to plan for.
  - Money, time, effort, people, space
  - Up-front costs, annual maintenance costs
- We need to plan for costs for several reasons:
  - Avoid overrunning budgets and deadlines.
  - Avoid reducing the planned return-on-investment of your project.
  - Make sure you are *prioritizing* (your agile stream of value delivery) the *most cost-effective* things, *early*.

So each individual design (solution, strategy, architecture), needs to be estimated, for at least more than one, critical resource, it consumes.

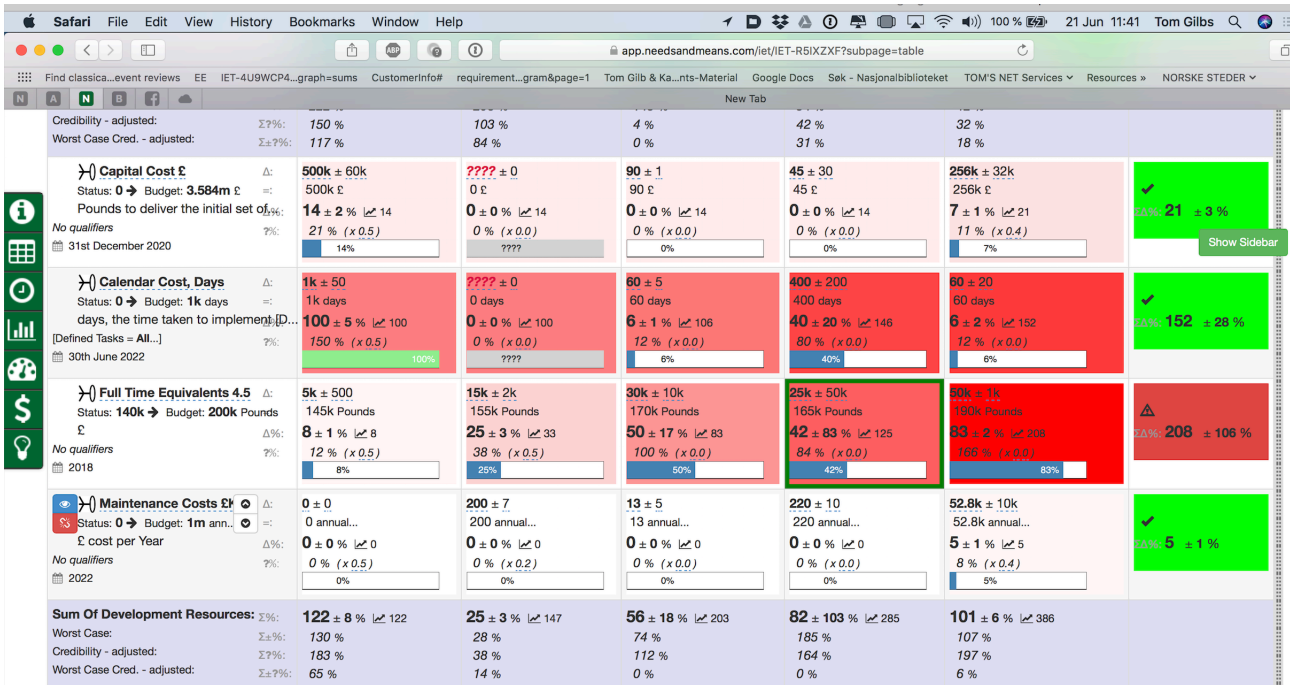
It is not just a matter of ***estimation*** of the resources. We need to followup *actual* costs, at least time and money, in relation to the corresponding estimates, as we gradually release agile value delivery cycles. We need to spot 'threateningly bad' cost deviations early, and do something about them immediately. Before we 'scale up' bad costs.

46. THE COMBINED POWER, TOWARDS ALL VALUES, OF A SINGLE STRATEGY



46. Value-Estimate Sum: A Technoscope for getting the overview of all values delivered, by a design or strategy.

- We have a tendency to focus on a *single value* of a strategy or design. Just human simplification I guess.
- But, like it or not, all designs and strategies, will have substantial effects on many, or most, of your critical value objectives, and on your many cost aspects.
- It is very much like a chess move, which like it or not, has *many* value and cost aspects.
  - And you lose the game, if you do not think about them all, before you move a piece.
- Life would be nice and simple, if a design had one-single effect, on *one* value. But that is unrealistic.
  - Designs have useful positive effects, on several other value requirements
  - They can have *negative* effects, on *other* requirements. Enough to kill the project.
- When evaluating, and prioritizing designs, we could 'just look at one single value that interests us at the moment primarily', for example 'security'. But nice and convenient as that seems,
  - we might miss the opportunity to chose *another* design that makes a large positive contribution ('for free') to many other critical objectives.
  - Or, *worse*, we might miss the fact that we are totally wasting our time with that design, because it has an unacceptable negative 'side effect' on another critical objective.
  - You choice is to use an hour of evaluation, to save a month of regret (for doing a bad design).
- Most planners I see, 'happily ignore' side effects, until 'bad things happen'. But you are reading this book because you want to be better than those failures. Right?
- Planners will do what they do. The changes come when a leader or manager decides to insist that their team does things much better. I am hoping some change leaders are reading this, and will act on it.



47. WE ADD UP THE 4 DIFFERENT COST CONSTRAINTS, TO GET A SUM.
THIS SUM IS FURTHER ANALYZED WITH REGARD TO ± UNCERTAINTY AND CREDIBILITY OF THE ESTIMATE
RED(ISH) CELLS INDICATE BUDGET EXCEEDED. GREEN MEANS WITHIN BUDGET.
Source: London 21 June 2018 BCS Architecture Engineering Class

47. Resource Sum: understanding the economic footprint, big picture.

- We presented ‘Resource Estimates’ just above. Now we will look at the **%-impact sum** of the individual resource estimates for all the 'concurrent strategies’, the ones we might all use sometime. Not exclusive alternatives.
- Since the various types of resources, will usually have *quite different Scales of measure*, we cannot simply ‘add them up’. But we can however add up the **% of the budget** or deadline, which each design is estimated to consume.
- This allows us to get a rough idea, of the resource consumption total, for each design.
- We could complicate this model by ‘weighting’ each resource. But I think it is probably not worth the effort. It could be done, if you think it is worth it for you.
- We do end up with a Resource Sum number, that primarily is used to develop a **Values Sum / Resources Sum;** an overall *cost effectiveness* number.
- This can be used to identify alternative designs with better cost-effectiveness, than others.
- This in turn is primarily used in **Value Decision** mode: (VD = analyzing the impact data on the Table),
  - to chose one design option, *instead* of others
  - to choose one design option as the probably best, and anyway very cost-effective design: and it should be the one we prioritize *to do next* (and of course *then* measure what *really* happened, with costs and values).

Technoscopes

Requirements	ProductDesign	Financials	MarketingStrategy	DistributionMethod	Sum
Demographic Past: **0** → Wish: **50** %	20 ± 5 % 0 Δ%: 40 ± 10 % 40	27 ± 5 % 0 Δ%: 54 ± 10 % 94	23 ± 3 % 0 Δ%: 46 ± 6 % 140	10 ± 0 % 0 Δ%: 20 ± 0 % 160	Σ Δ%: 160 ± 26 %
Millionaire Past: **1** → Wish: **1000000** $	450000 ± 150000 $ 0 Δ%: 45 ± 15 % 45	400000 ± 100000 $ 0 Δ%: 40 ± 10 % 85	100000 ± 50000 $ 0 Δ%: 10 ± 5 % 95	200000 ± 100000 $ 0 Δ%: 20 ± 10 % 115	Σ Δ%: 115 ± 40 %
MarketSegment Past: **4** → Wish: **1** Market Rank	1 ± 1 Market... 0 Δ%: 100 ± 33 % 100	4 ± 1 Market... 0 Δ%: 0 ± 33 % 100	2 ± 1 Market... 0 Δ%: 67 ± 33 % 167	3 ± 1 Market... 0 Δ%: 33 ± 33 % 200	Σ Δ%: 200 ± 132 %
Geography Past: **0** → Wish: **100** %	5 ± 5 % 0 Δ%: 5 ± 5 % 5	10 ± 4 % 0 Δ%: 10 ± 4 % 15	40 ± 5 % 0 Δ%: 40 ± 5 % 55	30 ± 5 % 0 Δ%: 30 ± 5 % 85	Σ Δ%: 85 ± 19 %
Market Past: **0** → Wish: **100** %	40 ± 10 % 0 Δ%: 40 ± 10 % 40	5 ± 3 % 0 Δ%: 5 ± 3 % 45	40 ± 10 % 0 Δ%: 40 ± 10 % 85	20 ± 5 % 1 Δ%: 20 ± 5 % 105	Σ Δ%: 105 ± 28 %
Sum Of Performance:	Σ%: 230 ± 73 % 230	Σ%: 109 ± 60 % 339	Σ%: 203 ± 59 % 542	Σ%: 123 ± 53 % 665	
TimeToMarket Past: **1** → Wish: **8** Weeks	2 ± [illegible] Weeks 0 Δ%: 14 ± [illegible] % 14	2 ± 0.5 Weeks 0 Δ%: 14 ± 7 % 28	3 ± 0.75 Weeks 0 Δ%: 29 ± 11 % 57	4 ± 1 Weeks 0 Δ%: 43 ± 14 % 100	Σ Δ%: 100 ± 39 %
ShowMeTheMoney Past: **0** → Wish: **5005** £	1200 ± [illegible] £ 0 Δ%: 24 ± 4 % 24	205 ± 200 £ 0 Δ%: 4 ± 4 % 28	2100 ± 500 £ 0 Δ%: 42 ± 10 % 70	1500 ± 0 £ 0 Δ%: 30 ± 0 % 100	Σ Δ%: 100 ± 18 %
Sum Of Resources:	Σ%: 38 ± 11 % 38	Σ%: 18 ± 11 % 56	Σ%: 71 ± 21 % 127	Σ%: 73 ± 14 % 200	
Performance To Cost:	6.05	6.06	2.86	1.68	
Ratio (Worst Case)	3.20	1.69	1.57	0.80	

Source Geoff Cooper, UK, Startup Plan, 2015

48. FINALLY USING THE PERFORMANCE SUM FOR EACH DESIGN, AND THE DEVELOPMENT RESOURCE SUM,
WE CAN COMPUTE AN OVERALL VALUE/COST RATIO.
THIS EFFECTIVENESS/COST RATIO IS ALSO ANALYSED WITH REGARD TO UNCERTAINTY AND CREDIBILITY

48. Values-to-Resources Ratio: a Technoscope for Cost-Effectiveness.

- It is important to somehow capture an idea of the ***effectiveness/cost ratio*** for design ideas.

- This can help us make decisions about **which designs are 'best'** in that (cost-effective) sense, and which ones we should prioritized early delivery of.

- We can even program spreadsheets and apps, to highlight the 'best options'.

The situation for the 4 person team, at week 9 of 12 before International Product Update Release, is shown in the column D (Improvements %).

Two of the goals are already reached (100%, 200%). Priority color Green.

One Goal is halfway there (50%), and has reached its Tolerable level (F7). Priority color Yellow. Tolerable.

Three goals are not at all improved from their Past Level (0%).They are priority color Red.

We have 4 choices to work on: one (or more) of the 0.0 % , or the 50%.

They chose 'Usability.Productivity', currently at 65 minutes, for 'time to setup a marketing report' (The Scale).

And decided to see if they could improve that, to at least a tolerable 35 minutes (F15), and perhaps to the Goal of 25 minutes (G15).

They did in fact achieve, that week, getting it down to 20 minutes: a 112.5% impact.

So, this is a decomposition based on dynamically available choices for 'filling the gap' to the Goal levels.

49. *NOTICE HOW THE SPREADSHEET COMPUTED THE PRIORITY AND MARKS PRIORITY REQUIREMENTS WITH TRAFFIC LIGHT COLORS.*

EVO Plan Confirmit 8.5
4 more product areas were attacked concurrently

Impact Estimation Table: Reportal codename "Hyggen"

Current Status	Improvements		Reportal - E-SAT features			
Units	Units	%	Past	Tolerable	Goal	
			Usability.Intuitivness (%)			
75,0	25,0	62,5	50	75	90	
			Usability.Consistency.Visual (Elements)			
14,0	14,0	100,0	0	11	14	
			Usability.Consistency.Interaction (Components)			
15,0	15,0	107,1	0	11	14	
			Usability.Productivity (minutes)			
5,0	75,0	96,2	80	5	2	
5,0	45,0	95,7	50	5	1	
			Usability.Flexibility.OfflineReport.ExportFormats			
3,0	2,0	66,7	1	3	4	
			Usability.Robustness (errors)			
1,0	22,0	95,7	7	1	0	
			Usability.Replacability (nr of features)			
4,0	5,0	100,0	8	5	3	
			Usability.ResponseTime.ExportReport (minutes			
1,0	12,0	150,0	13	13	5	
			Usability.ResponseTime.ViewReport (seconds)			
1,0	14,0	100,0	15	3	1	
			Development resources			
203,0			0		191	

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	1281	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

Current Status	Improvements		Reportal - MR Features		
Units	Units	%	Past	Tolerable	Goal
			Usability.Replacability (feature count)		
1,0	1,0	50,0	14	13	12
			Usability.Productivity (minutes)		
20,0	45,0	112,5	65	35	25
			Usability.ClientAcceptance (features count)		
4,4	4,4	36,7	0	4	12
			Development resources		
101,0			0		86

Current Status	Improvements		XML Web Services		
Units	Units	%	Past	Tolerable	Goal
			TransferDefinition.Usability.Efficiency		
7,0	9,0	81,8	16	10	5
17,0	8,0	53,3	25	15	10
			TransferDefinition.Usability.Response		
943,0	-186,0	######	170	60	30
			TransferDefinition.Usability.Intuitiveness		
5,0	10,0	95,2	15	7,5	4,5
			Development resources		
2,0			0		48

Source. The Confirmit (Norway, confirmit.com) case, in Value Planning book 5.10. 2003.

49. Prioritization Options: Estimated Static, Next Step Measured Computed.

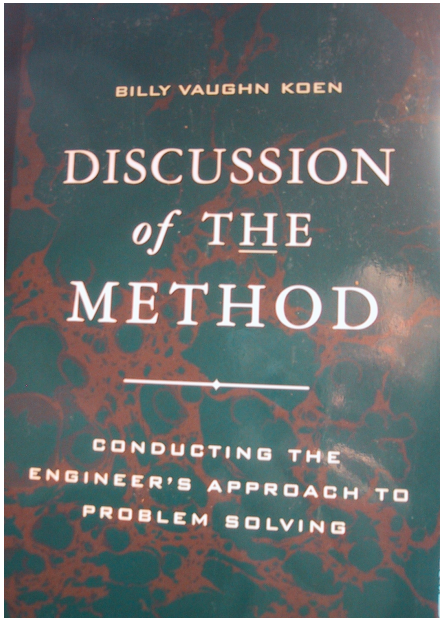
- **Prioritization** is an evaluation process, which helps us make our projects more cost-effective. Cost-effective refers to the ratio of **value-effectiveness /costs**. Another good term for cost-effectiveness is 'efficiency' (value/cost)
- Our ability to prioritize, depends on the data we have, about the designs, or other planning objects (stakeholders, objectives).
- The purpose of Planguage is to provide us with a language, and an opportunity to provide prioritization information.
- There are many different artifacts, that can help us to understand the relative priority of a planning object.
  - There is nothing simple like a 'weight' or a 'priority declaration' to prioritize. *Those* are primitive and static tools.
  - Priority can be partly determined by for example, power of a stakeholder, geographical location, constraints, such as laws, and dozens of other factors. People will subjectively decide on the basis of this data what they 'prioritize'
- But, there is one method of prioritization I want to spell out here.
  - It is one that we people use intuitively all the time, for example to decide how to prioritize our body actions: breathe, sleep, eat, drink.
  - It is so natural we don't initially think much about it.
  - I call it nature's own prioritization method.
  - It is based one the following concepts: **multiple critical factors** (breath, heart beat, energy), and of **variable prioritization signals** (starving, hungry, satisfied, over-satisfied)
- I think that there is a lot of planning power in the Technoscope, of making this dynamic and multi-attribute prioritization process *practical* and *operative,* in everyday planning.
- All of the above factors of multi-dimensional 'value', and levels of the values (Tolerable, Goal) are in Planguage, and are discussed above.
- All we have to do is 'put it all together'. Do the maths. We can compute our priorities, and determine them with agreed logic (EXAMPLE: a factor that is not life-threatening (Tolerable or Goal level), has *lower* priority than one that is Intolerable)
- There are two major types of priority determination: estimated, and measured:
  - in 'Estimated Prioritization' we determine priority based on impact estimates, in a Value Decision Table.
    - In 'Measured Prioritization' we have deployed an agile design-increment and measured the resulting values and costs. We compute our priorities just like our body does, on the run: **we prioritize by 'satisfying the weakest satisfied value up to now', using the 'most plentiful *remaining* resources'.**

"Make small changes in the SOTA"

50. 'SOTA' = ENGINEERING STATE OF THE ART HEURISTICS

Source: Prof. Billy Koen, 'Discussion of the Method', p. 48, quoted in 'Value Planning'

"Always give yourself a chance to retreat; and Use feedback to stabilize the design process"



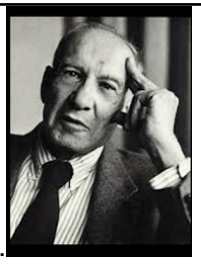
March 9-10 2019 Edit

50. Feedback: the most powerful Technoscope of all.

- Feedback means ‘data from a real world action’
- Feedback tells us what is true and real, well at least compared to our overoptimistic estimates.
- Feedback, if we get it early enough, gives us a chance to correct a bad situation, or to capitalize on a surprising opportunity
- Feedback, when negative or disappointing, can allow us to cut or reduce our losses quickly: **IF**
  - We have made a small investment (2% !), and/or IF
  - we can reverse our commitments (return the hardware, opt out of the contract, not pay the supplier for their bad results)
- Decades ago I held a talk in Japan[4] about the most powerful principles I knew. And *feedback* was top of this list. I did this by observing my clients’ results, for decades internationally, a simple fact. Methods with good feedback mechanisms work surprisingly well. Methods with bad-or-no feedback mechanisms work surprisingly badly, and fail.
- In Planguage, the most important feedback, is about the level or increment of value delivery. Secondarily, the feedback about the consumption of resources is important feedback.

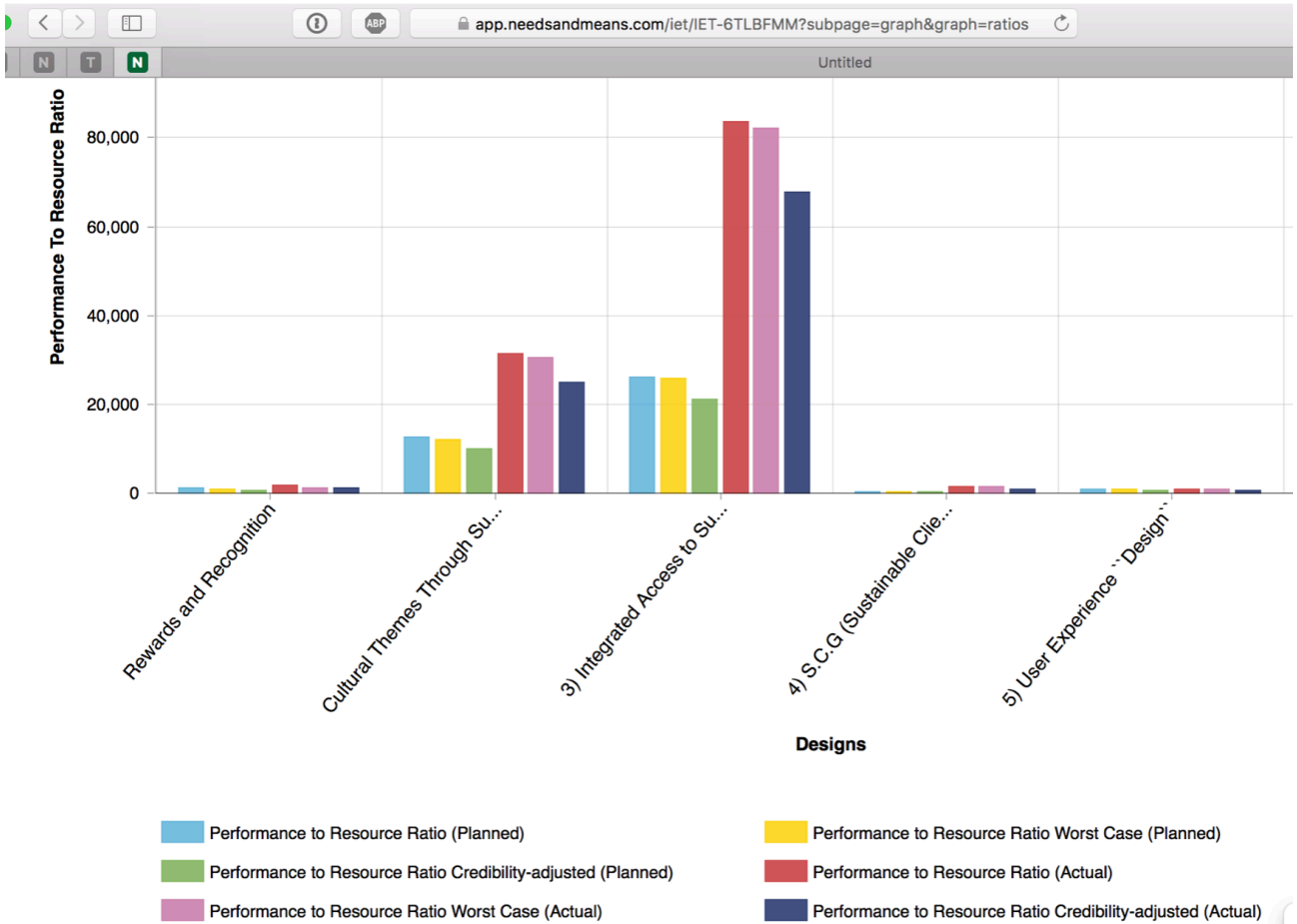
“Work implies not only that somebody is supposed to do the job, but also accountability, a deadline and, finally, the measurement of results —that is, feedback from results on the work and on the planning process itself,”

Drucker wrote this in ‘Management: Tasks, Responsibilities, Practices’. Quoted from Value Planning book.



Peter Drucker,: Business Author (1909 Vienna - 2005 Claremont, California).

[4] https://pdfs.semanticscholar.org/5402/6ac180c0ac0ae6a83c5978b1f0afed1307ee.pdf



51. 3 TYPES OF *ESTIMATES* FOR EACH DESIGN, ('PLANNED')
TOGETHER WITH CORRESPONDING *MEASUREMENT* FEEDBACK (ANOTHER 3 MEASURES) 'ACTUAL'
IT LOOKS LIKE ACTUAL RESULTS ARE FAR BETTER THAN THE ESTIMATES
OPPORTUNITY KNOCKS!

51. Deviation Analysis: what to look for and how to analyze deviations.

- There will usually be *some deviation* from your estimates, or expectations. When you get feedback from an incremental delivery. Using a step size of about 2% of total budget.
- Here are some deviations you should be interested in
  - When the result is *well outside* the ± uncertainty
  - When the incremental total (value progress to date) is *far below* expectations, at this stage
  - When the costs are way outside of estimates

These are all Technoscopes, for seeing feedback, and inviting you to action

"True wisdom is knowing what you don't know"

"By three methods we may learn wisdom:
First, by reflection, which is noblest;
Second, by imitation, which is easiest; and
third by experience, which is the bitterest."

Confucius, Died 479 BC (aged 71–72) Sorry, no photo ☺ <- VP 9.6

Source VP Book Quote 9.9 C

It's a very sobering feeling to be up in space and realize that one's safety factor was determined by the lowest bidder on a government contract.

Alan Bartlett Shepard (1923-1998) Astronaut.

***52.* THE SAFETY FACTOR IS STIPULATED TO 2 (200% OVER-DESIGN IS NEEDED) THE RIGHT HAND RED CELL IS A WARNING THAT WE DO NOT HAVE ENOUGH DESIGN YET: GREEN=OK**

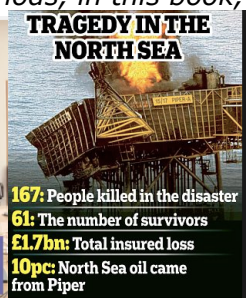
Source: Architecture Engineering Course, London 21 June 2018, BCS

52. Safety Factor: making sure it will work, when expectations don't work out they way you hoped.

"Safety is not an intellectual exercise to keep us in work. It is a matter of life and death. It is the sum of our contributions to safety management that determines whether the people we work with live or die" –

Sir Brian Appleton after Piper Alpha (Oil Platform Disaster, 1988).

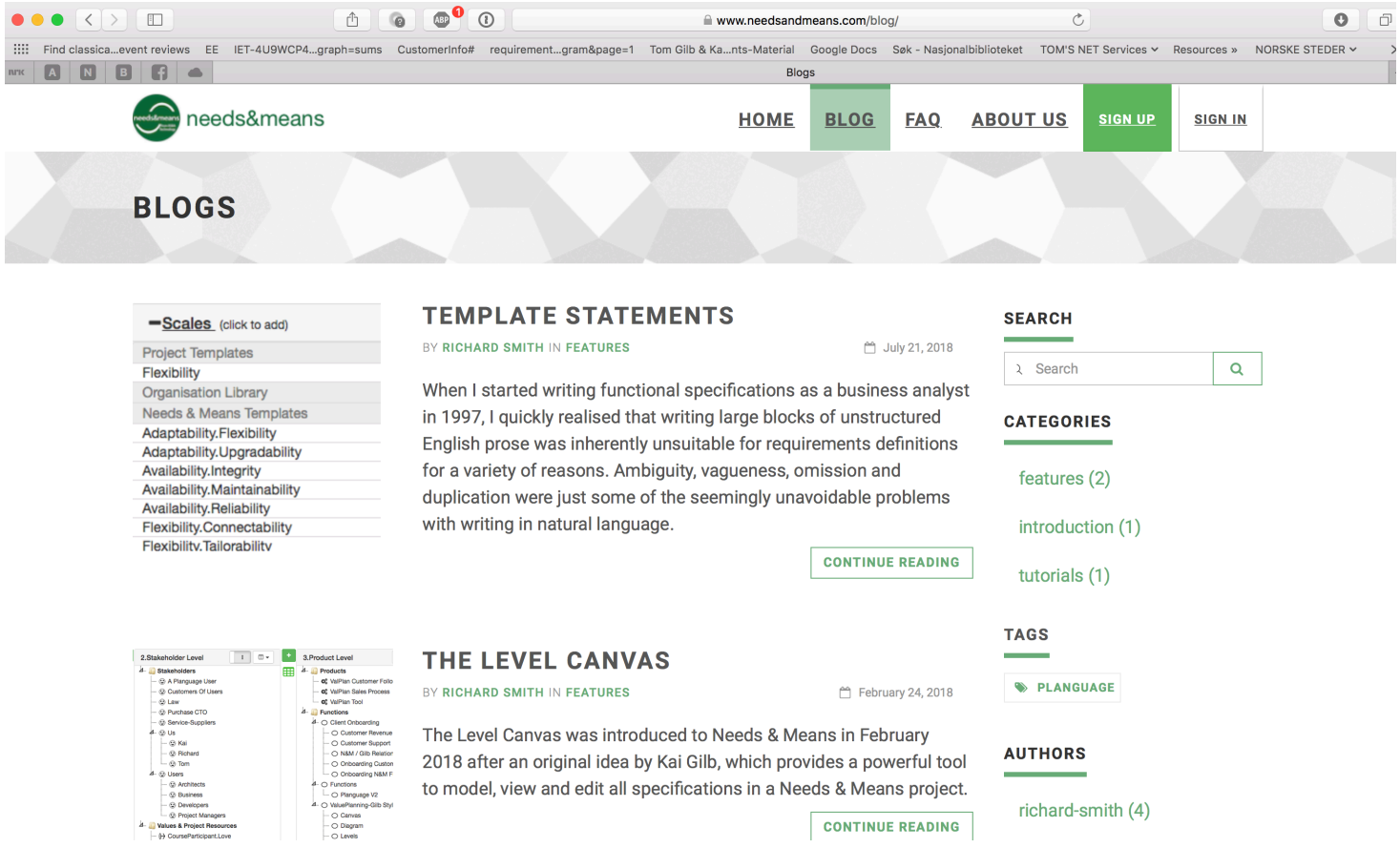
(Kai and I used the methods, in this book, to help plan Oil Platforms.)



- 'Safety Factor' is an engineering tradition. See the 2 quotations here.
- It is also a Technoscope that can help you understand if you have enough 'design power' to be on the safe side.
- Access to this safety factor, is through a set of quantified attributes: 1. quantifying the Goal level, 2. Estimating the impact of a Design, 3. Adding up the % impacts of a set of designs.
- If they total 200% we accept we have a safety factor (of 2x), good enough to dive in to the agile value delivery process. And then see what *really* happens.
  - We can set *our own* safety factor, based on experience, and risk willingness.
- If your agile increments, in *fact,* get you to your 100% Goal level, long before you have run out of design ideas, then you do not strictly need the other design ideas. Unless you want to improve upon the Goal level (go for a Stretch level).
- Obviously we need to prioritize doing the most cost-effective estimated increments of design FIRST!

March 9-10 2019 Edit

53. APP: FOR PLANGUAGE



March 9-10 2019 Edit

53. App: Needs And Means = ValPlan.net

Since about 2014-2018 we have had the pleasure of Beta Testing an app to 'do Planguage'. 'NeedsAndMeans', now ValPlan.net
It is developed by Richard Smith, of London. He learned our methods at Citigroup about 2003, and has been using them industrially, and successfully since then. His basis for design of the Technoscope tool is the definition of Planguage in 'Competitive Engineering' (2005).

Kai Gilb and I, along with our clients and students, have been giving him feedback for years, and Richard has rapidly continued improving the app considerably. Kai will do the marketing from 2018 via ValPlan.net

Kai himself spent about 15 years building a Planguage tool in Excel. But that language has its limitations. Most of our clients have built tools in Word and Excel. But there is a limit to how much one user can invest.

Bechtel (major Construction, USA, Frederick Gibson) used Competitive Engineering to build a Planguage based tool for Bechtel and the Building Industry. In 2018 he has a new startup, to go public with a similar offering. https://graphmetrix.com
The 3 D Graphics he showed me reminded me of Tom Cruise in Minority Report. The surprise for me is that Fredrick, who is a building Architect, felt that Planguage was such a good basis for organizing tools for the construction industry ! We are on his advisory board.

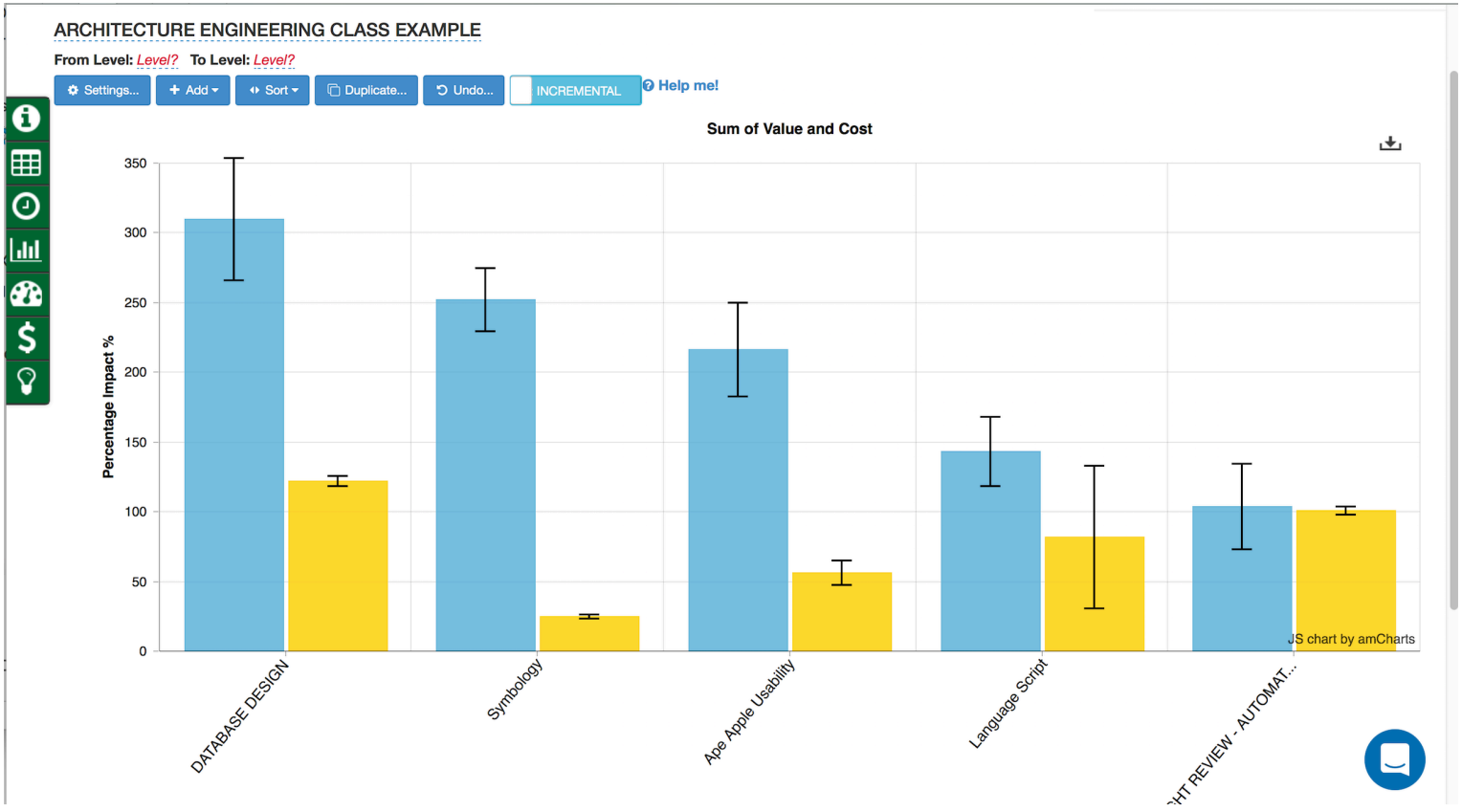
Apps are not a pre-condition for use of these Technoscopes. Pen and paper, or whiteboards work well too.
Shakespeare, Wren, and Ibsen used a quill pen.

But automation has some advantages: especially for large and complex plans.
- Less work to build a plan, and to edit it
- More-automated calculations (Like 'sum value' and 'safety factor')
- Ability to derive selective reports and graphical presentations
- Ability to standardize the definitions.
- Ability to train the app to do quality control analysis
- Ability to build standard libraries, and local libraries of terms, processes, scales, and much more.

Some people are initially skeptical about learning a new tool. I am too! I stick to the old stuff I know!

But our practical experience is that in two days of training, a whole classroom of people, happily learn to use the tool, while learning Planguage and discovering new Technoscopes. We are also building super fool-proof interfaces for inputting stuff. So you will not really have to be conscious of the Planguage rules themselves. Exciting times. Follow progress at Gilb.com

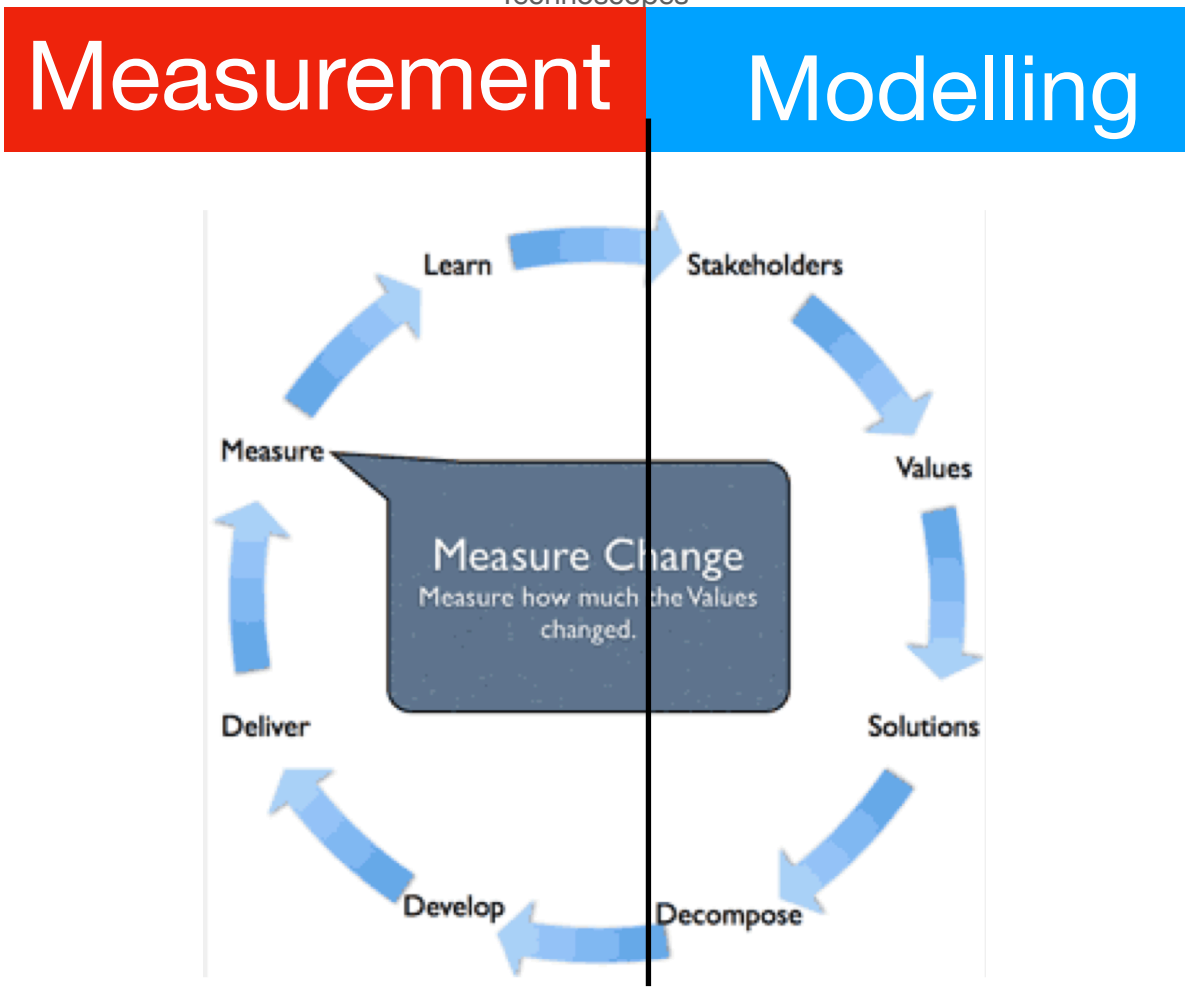
March 9-10 2019 Edit



54. THE COMPLICATED, BUT NECESSARY DETAIL, CAN EASILY BE CONVERTED TO BAR CHARTS IN THIS CASE THEY ARE SORTED LEFT TO RIGHT IN PRIORITY SEQUENCE THE I BAR GIVING SOME IDEA OF THE WORST CASE TOTAL VALUES ON LEFT, AND TOTAL COSTS TO THEIR RIGHT FOR EACH STRATEGY

54. Bar Charts: A simplified presentation of complex evaluations.

- Graphical visualization of complex sets of detail is a useful Technoscope
- Some people do not need to know all the scales and numbers, just the ‘bottom line’
  - They or others, can ‘drill down’ to the details, if necessary
- And we can automatically sort the choices into priority order, so we easily see the winner, and why
- Let me point out that this is a sequencing and choice of agile result delivery options
- When the feedback changes things, and other circumstances change, we can easily update and calculate new priorities.
  - Or look at ‘what if’ optional scenarios.



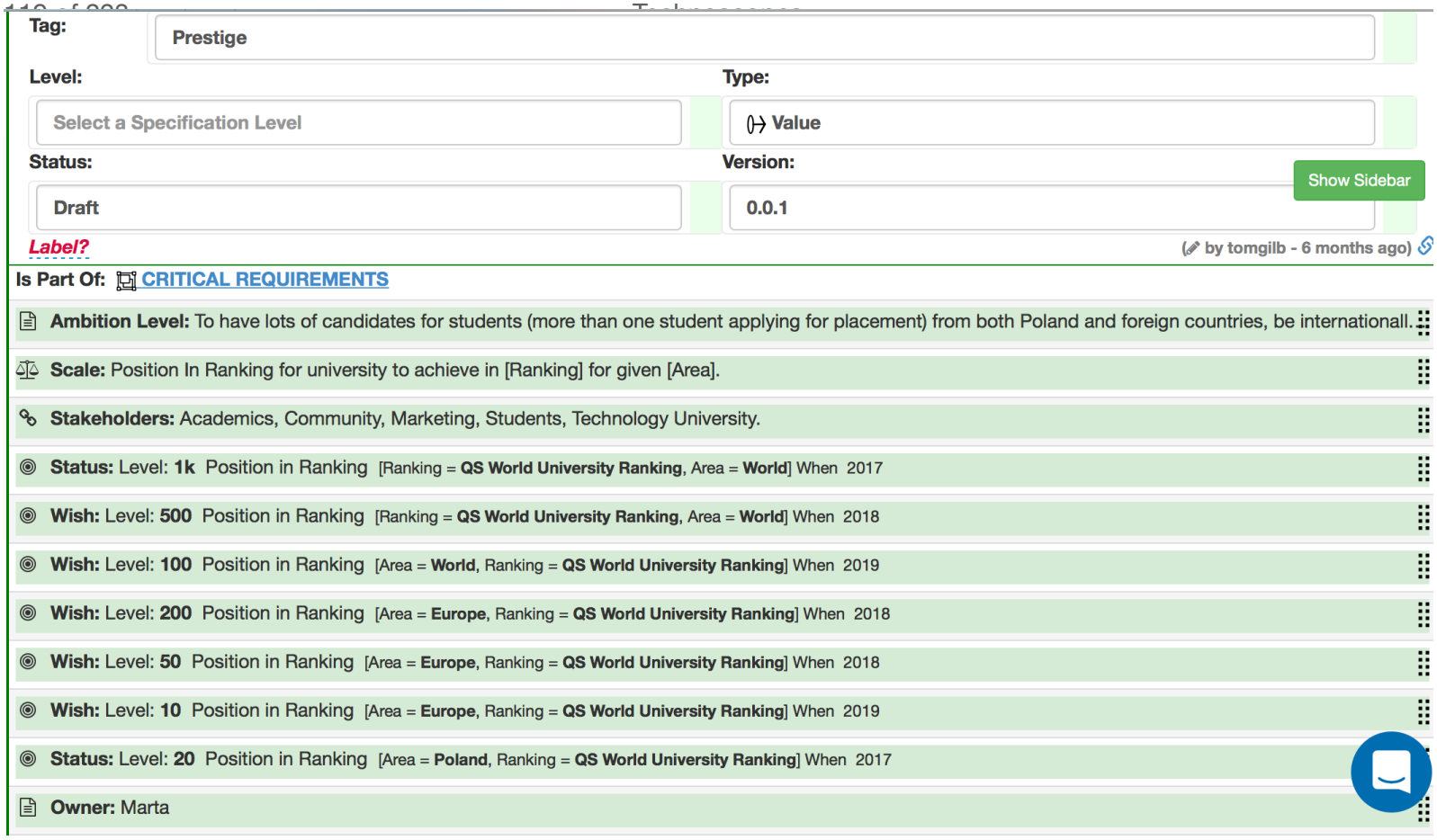
55. THE GILB VALUE DELIVERY CYCLE: SPLIT BETWEEN MEASUREMENT AND MODELLING

55. Modelling versus Measurement.

- Planguage is my Technoscope for modelling: especially for modelling ‘quantified values’ and costs.
- Modelling and estimations have their limitations[5]: we hope they guide us, but cannot be sure they are true.
- So, as much as I enjoy better modelling, and also detailed modelling of complex systems,
  - I believe we need to **limit** the modelling, keep the effort short and sweet, or quick and dirty
- Then, using the rough prioritization results, we need to dive in there really early (2nd week of a project) and JUST DO IT.
- We need to get a confrontation with real people, and real systems ASAP
- And, feed the measurements and stakeholder feedback, into the model.
- One key to this, was discussed earlier: decomposition of strategies and designs, so that we in fact have something specific we can do, in the *second week* of a project; to deliver some value, to get *some* feedback, and to help you check your modelling estimates, before your over-optimism gets in the way of disappointing reality.
- Model, to set the stage for fast results, and fast learning in practice.

---

[5] "What is drastically wrong with most software engineering modeling languages and approaches, and 10 necessary principles for a really good modeling language” http://concepts.gilb.com/dl795



56. ALL TARGETS (LIKE 'WISH') AND CONSTRAINTS (LIKE 'TOLERABLE') CAN HAVE A DIFFERENT DEADLINE FOR DELIVERY OF THAT DEFINED VALUE

Source: Polish Export Plan, Class Case 2017, Warsaw

56. Time, Micro Deadlines; continuous and early deadlines.

- In conventional project culture, there is ‘one big deadline at the end' (‘project done’), with perhaps some large phases of partial delivery before the end. Interspersed with a lot of tasks, to build something, like the phases.
- In our Planguage planning, method, which is anchored in ‘agile incremental value delivery', small steps, we have a tradition of much smaller ‘phases’, a week to a smonth. And the ‘deadline’ concept is entirely different.
- We want the earliest possible delivery of the *highest priority* values, to the *most critical* stakeholder, conditions, and places.
- In particular, we are rarely bound to the idea, that we are going to build something big, from scratch; so that any useful result, of that building effort, will be years in the making.
- We prefer the idea that if at all possible, we will NOT focus on building a new system.
  - We will focus on delivery of improved values,
  - And we will do so in increments to the **existing systems** (even though these might be replaced in time)

The consequence of all this, is that we can choose to prioritize increments of value, early, continuously, without waiting and hoping for the big-bang system, after some years (which seems usually to be a big disappointment).

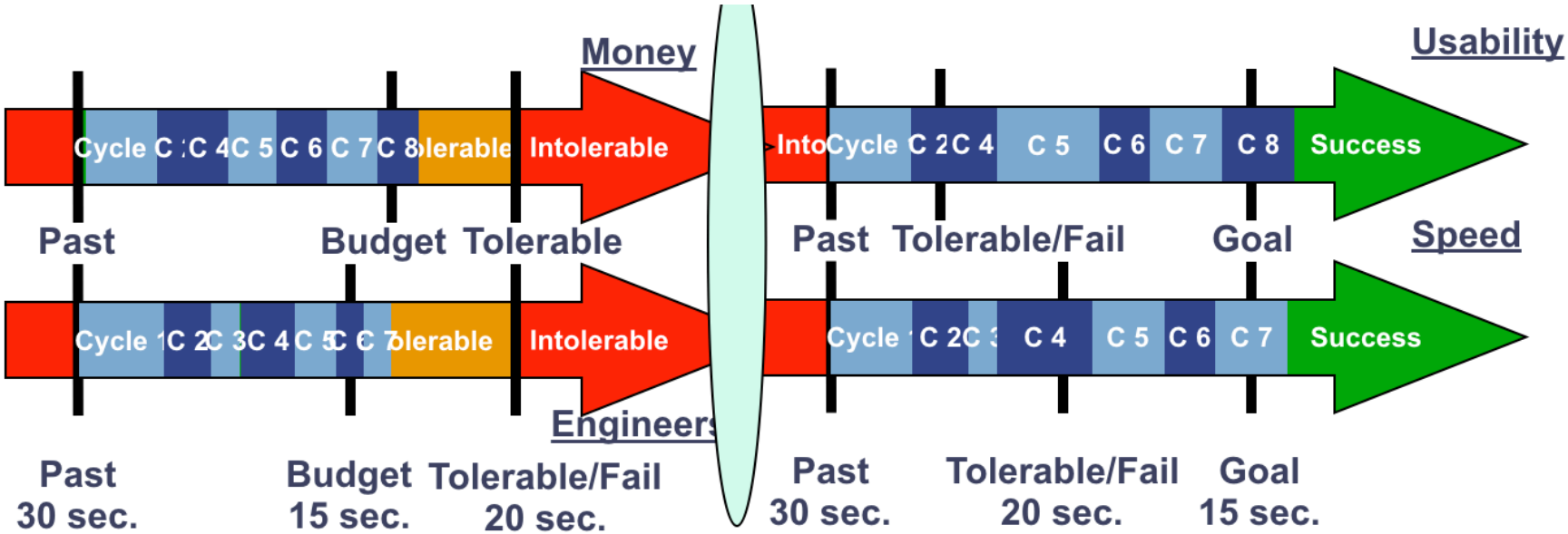
That means we can stipulate very many ‘deadlines', dates, for delivery of increments to selected values, to high priority stakeholders.

We end up with many ‘Micro Deadlines’, spread out over the several years span of a large project.

The big advantage of this is simple: early results, early feedback and correction, no big failures.

I call this ‘agile as it should be’: I call it ‘Evo’. More later.

The Micro Deadlines are a Technoscope into the timing and priorities of the project.



57. AT THE LEFT OF THE ARROWS IS THE 'SHORT TERM', AND AT THE GOAL IS THE LONGEST PLANNED TERM
THIS IS A SIMPLE MODEL, AND REAL PROJECTS HAVE 50 TO 100 OR MORE CYCLES OF DELIVERY
AND OF COURSE WE CAN PLAN *PARALLEL* VALUE DELIVERIES.
THERE IS NOTHING SIMPLE LIKE A RIGOROUS SINGLE SERIES OF DELIVERY CYCLES.

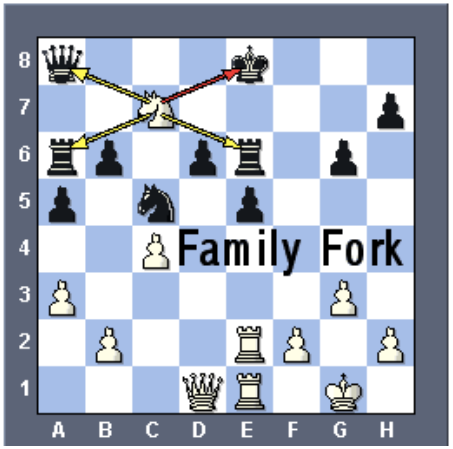
March 9-10 2019 Edit

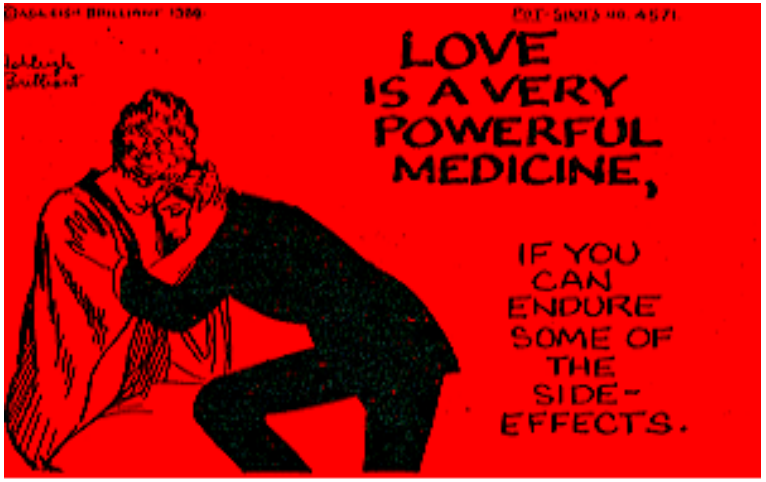
Source: Kai Gilb

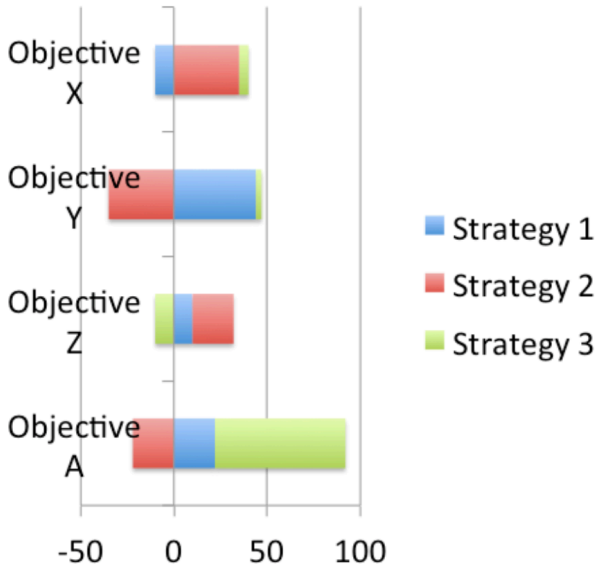
57. Long-Term Short-Term.

- Once we shift our focus from '*building* systems', to '*delivering improved value'*,
  - we have opened the door to a 'series of small deadlines',
  - NOT for delivery of tasks,
  - but for delivery of improved value to stakeholders.

We can consciously plan for short term, prioritized results, while still on an incremental path to longer-term results, or 'value delivery.







58. SIDE-EFFECT IMPACT ANALYSIS: WHICH STRATEGY IS BEST? MIGHT BE NICE WITH A NUMERIC OVERVIEW? (VDT!)

Source: Value Planning, 2.3B

March 9-10 2019 Edit

58. Side-effect Analysis: Technoscopes for side-effects.

- All designs and strategies have a large number of side-effects.
- Side-effects are simply ‘not the primary reason’ you picked the design.
  - You may have focussed on a ‘Security’ objective, and picked a *great* security design. The best you can find.
  - But like it or not, that design comes with a whole lot of other attributes, **good** and **bad**
  - To **begin with** there are all the **costs**, long-term and short-term: nothing in life is free.
    - You cannot simply pick a design because ‘it is the greatest at your primary objective’, or for the objective you are focussing on at the moment.
    - You have to consider the costs, all kinds of costs, not just capital outlay, but just as important, and sometimes much bigger, the running costs, year after year.
    - Not just the money, but *people* time, and the need for rare specialized experts
      - (think ‘Cyber Crime’, the best experts seem to be on the Criminal side)
  - And then there are all the **value** effects, the good the bad and the ugly, on things like usability, maintainability, throughput, and other critical things like that.
    - You need Technoscopes to *help you see* all these side-effects.
    - You need to see these side-effects **quantitatively**. You can’t just note that ‘this design also impacts usability’.
    - You need to *estimate*, as best you can, always quite difficult, but never impossible, hoping for an early insight which will prevent you doing ‘something stupid’ (there is a Sinatra song about that).
- And, that is just the beginning: estimating designs **pre**-selection.
- We also need to get **feedback**, after incremental change implementation, to learn how the side effects **really** work!
  - And to follow up, and adjust, for nasty surprises, early.
- So, as you may have observed by now, we have a Technoscope for side-effect estimation: the Value Decision Table.
    - Most people do not have such a tool. They do not ‘bother’ to manage the side effects. They get problems and they fail. They do not blame themselves and their poor methods. They just say, it was ‘surprising’.
- **If you do not get the side-effects first, the side-effects will get *you*, late**

Threats are anything that can possibly result in an attack on your system.

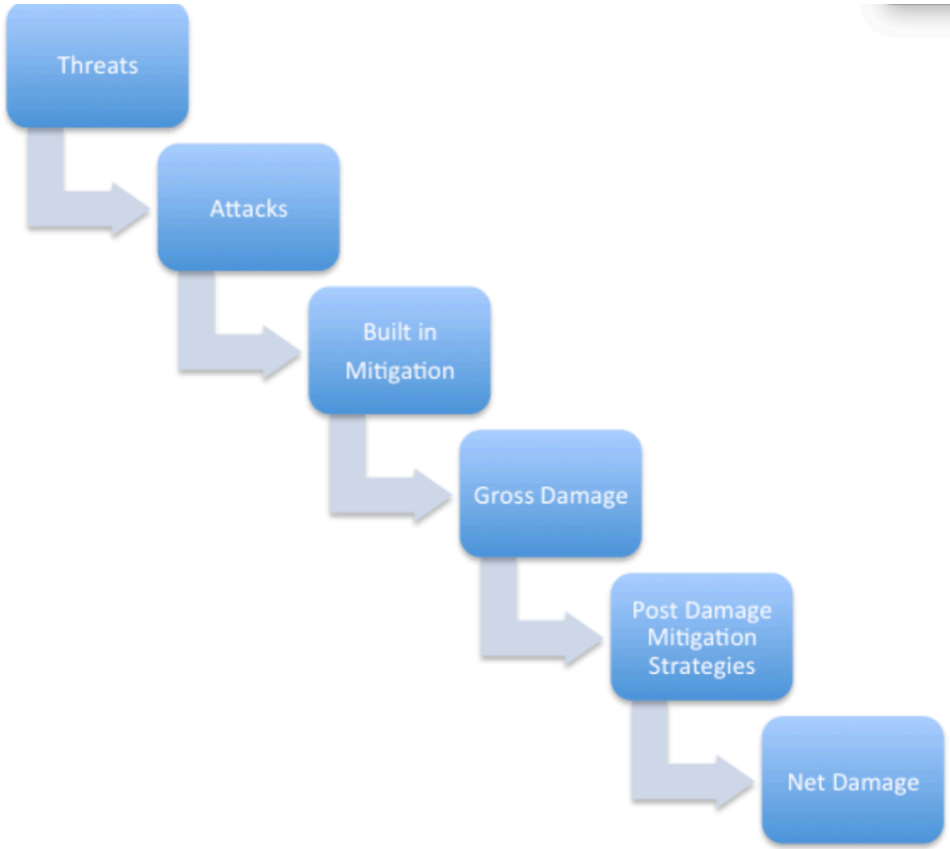
Attacks are when something attempts to misuse or destroy your system

Built in Mitigation is when some device you have planned and invested in, attempts to detect and possibly stop the attack

Gross Damage is the successful attack extent; not prevented by your Built-In mitigation

Post Damage Mitigation Strategies: are what you can do, now that the damage is done

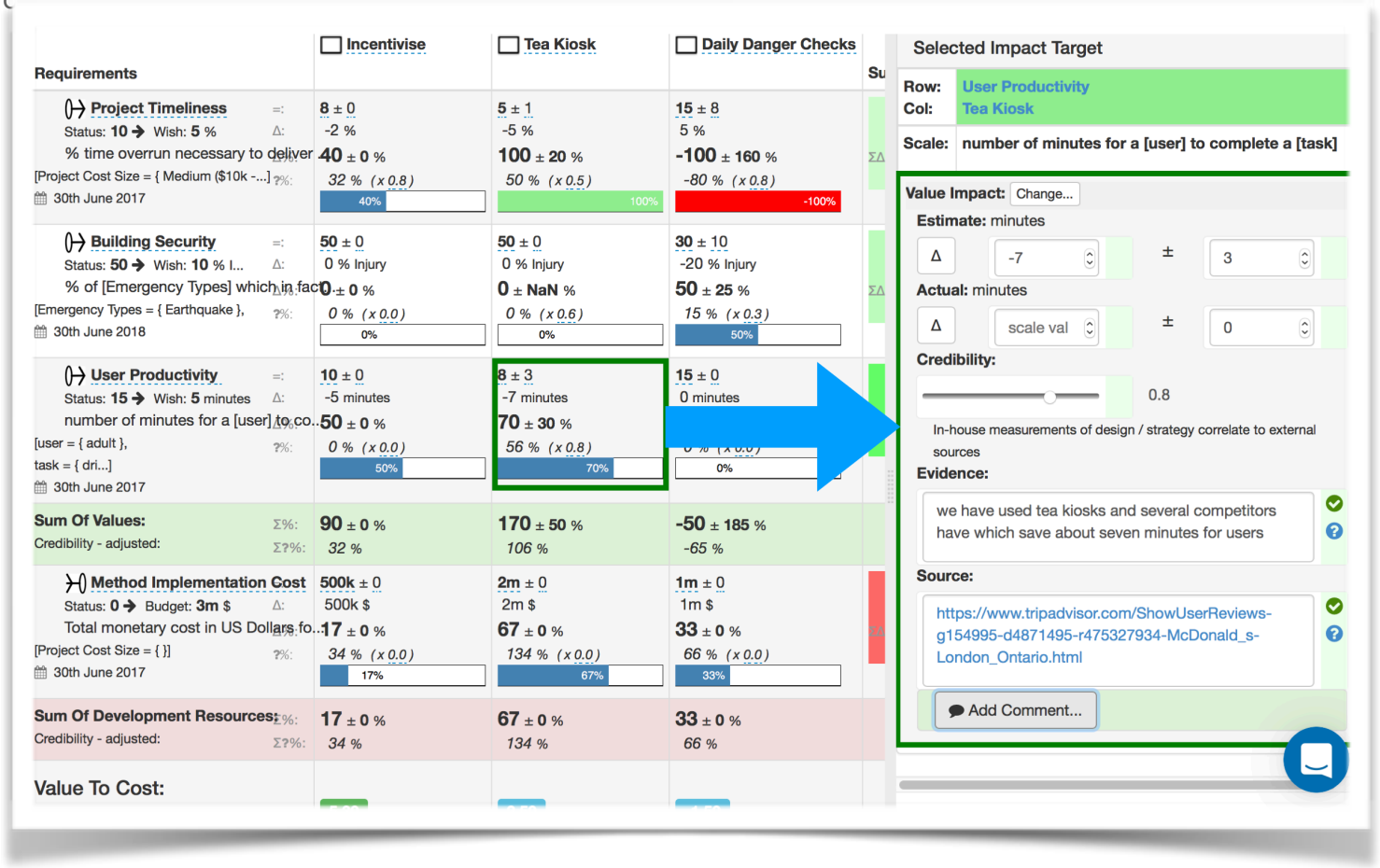
Net Damage is Gross Damage Loss, minus what you can salvage using Post-Damage Mitigation Strategi



59. THREAT AND 'ACTUAL ATTACK' ANALYSIS GIVES YOU INSIGHT AS TO WHAT YOU NEED TO DESIGN.
THERE ARE AT LEAST 2 LEVELS OF ATTACK MITIGATION, PLUS A POTENTIAL TO DESIGN SO THAT THREATS DO NOT TURN INTO REAL ATTACKS (THREAT MITIGATION). SOURCE [VP]

59. Risk Prevention: you have to spot the risks, to prevent them.

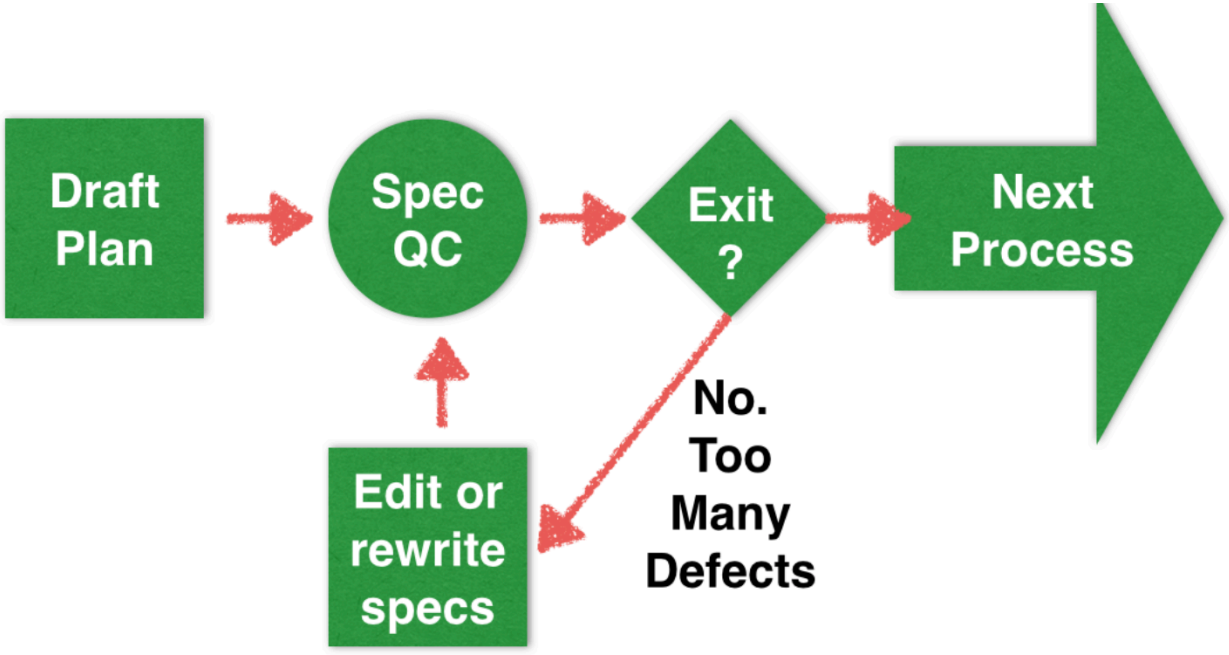
- If you do not see problems, threats and risks in time: *don't worry*!
  - They will ***find you,*** sooner or later: at 100 times the costs of dealing with them early.
- That is the essence of all planning: not just finding 'a way to succeed', but to 'avoid ways of failing', at the same time.
- Understanding the risks is difficult. But you will have to find a balance between effort to deal with risks early, and the pain of enduring them later.
- As should already be clear, for readers thus far (60% of the booklet), we have presented *many* Technoscope tools to identify risks, to specify risks, and to do something early about risks.
  - In fact practically every detail in Planguage is some sort of a Technoscope for dealing with some sort of risks
  - For example Tags, Scale, Record, Safety Factor, ± uncertainty, Source, Value Decision Tables, Evo:
    - practically all Planguage detailed artifacts deal with risks, fairly directly.
- One of the problems with risks is that they are so infinite in variety, and it is very difficult to 'predict' the rare ones.
  - So one useful tactic is to plan for *general classes* of risks, not the detailed ones.
- I particularly like, at risk of repetition, the 'Ericsson Quality Policy' (one of our favorite clients for a decade) which says
  - that **risk analysis and risk management is the ongoing work of every engineer, at all times, in every detail**.
  - Well maybe I exaggerate a little, but not much!



60. ENLARGED 'CELL' DATA. WITH FULL EVIDENCE, AND EVIDENCE SOURCE RATED HIGH, AT 0.8

60. Evidence and Credibility.

- We have explored the idea that we can *estimate* the impacts of designs, on our values.
- We do not think that people can be allowed to just ‘estimate a number they believe in’
- We think that, not only should an estimate specify the uncertainty, ’60±25’,
  - We need to document the *basis* for our estimate,
  - *even* if it is ‘just our personal opinion’.
- Planguage has the following devices, Technoscopes, to *clarify* the evidence and its *quality*.
  - The **Evidence**
  -
  - The **Source** of the Evidence
  -
  - The ‘**Credibility**’ level of the Source and its Evidence
    - The Credibility level is on a scale of 0.0 (none) to 1.0 (perfect)
  - This forces us to think seriously about our estimates
  - Credibility level is a Technoscope to see the credibility, of all estimates, in a Value Delivery Table.
  - Credibility motivates planners to take their work more seriously, and it
  - Gives managers, and plan evaluators, a tool for understanding *risks*.



PART 4. THE 'SPECIFICATION QUALITY CONTROL' (SQC) PROCESS
MOTIVATING PEOPLE TO REALLY FOLLOW STANDARD BEST PRACTICES : A GENERIC TECHNOSCOPE FOR ALL PLANS

Source: Value Planning, 10.2A

Part 4: Spec QC: (‘Specification Quality Control’)

61 Defect
62 Specification Rules
63 Source Specs
64 Major (Defect)
65 Defects Per Page
66 Entry Conditions
67 Exit Conditions
68 Exit Level
69 Sample
70 Individual Learning Process
71 Checklists
72 Checking Rate
73 Checking Teams
74 Spec QC Measurement
75 Inspection Defect Removal (obsolete process)
76 No Blame Culture
77 Result Anonymity
78 Early and Continuous Sampling
79 Defect Prevention Process
80 Automated Spec QC

“Timeless Principles of Quality Control”

- The Egyptian workmen, dressing stone, are getting data feedback from Inspectors, who are
- checking the blocks with boning rods, to measure the deviation from an exact plane
- The pictures are from a tomb in Thebes, c. 1450 B. C

61. DETECTING ‘DEFECTS’ IN RELATION TO RULES OR STANDARDS IS AN ANCIENT IDEA

61. Defect: a window to future threats to your project.

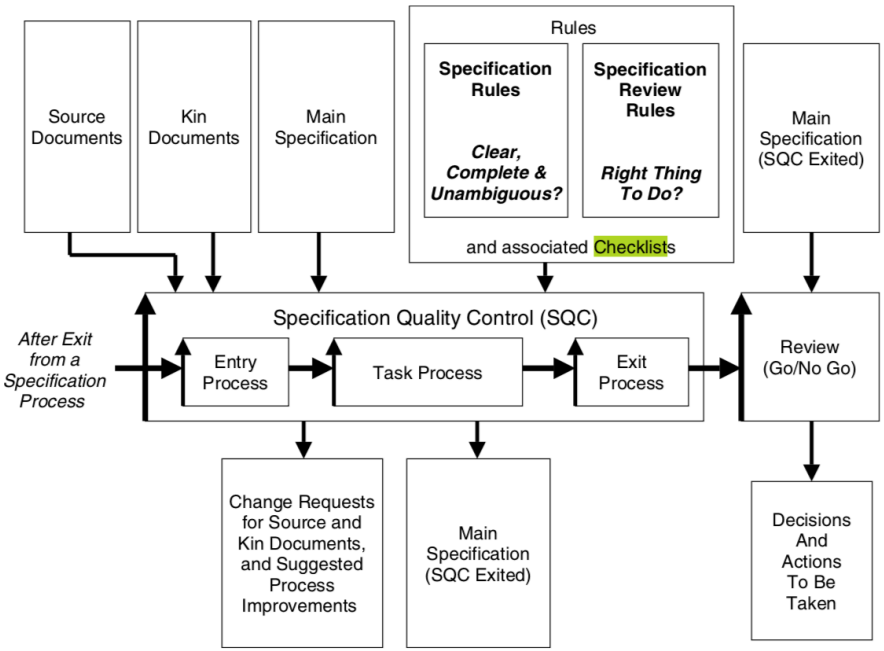
- Our subject now is a Technoscope that helps us to *see potential error in our plans*, so we can both *correct* them, and more important *prevent* them.
- I call it **Specification Quality Control** (Spec QC, or SQC), because it is focussed on Quality Control of *any* type of specification, and plans, contracts, technical drawings, computer programs, test plans, requirements, architecture, etc.
- The thing we are trying to identify is called a '**defect**', and this is short for '**specification** defect'
- A specification defect is like an *illegal act*, it is anything *against current 'laws'*; which we call 'Specification **Rules**'.
- The concept of doing QC against a 'standard', like a law or a spec rule, is at least as old as the Egyptians (see ill. 61).
- But in modern times, we all too often, see people reviewing a specification, by simply reading it, or scanning it, looking for 'something that does not seem right'. And indeed, if *their* personal rules are sound ones, they will find defects.
- The problem is that we cannot trust just any 'individual' to know 'all the good rules'. And even if the best of us did indeed know all the good rules, it might be nice if other people knew those rules too, and that all the top experts agreed on what the 'good rules' were.
- This practice above is *subjective* QC, and there are some good reasons to move serious work to *Objective* QC
- Since specification defects, are in plans for real world systems, we might get lucky. Someone *could* discover the defect in time, or ignore it, or even correct the real-world faults it causes, at an *early* stage, before it does too much damage.
- But it is a good 'lean' preemptive practice to reduce the 'density of defects', in our planning documents. We might *not* get lucky finding all the defects. In fact, in software we get lucky about 70% of the time (Capers Jones), and the defect leads to a 'bug' in the software 'only' about 30% of the time.
- Some people, misleadingly, believe: 'no problem', we will test the system, and find and fix all the bugs. But they do not understand some powerful economic facts:
  - The tests might only find 15% to 85% (Capers Jones data) of the bugs. About half, on average. The rest threaten us.
  - The cost of *preventing* the errors (or software 'bugs') is about 12 times less than the cost of finding and fixing *after* tests (Robert Mays, IBM, DPP)
- And as the old saying goes: **"A stitch in time saves nine"**

62. The Norwegian Mountain Rules

1. ***Be prepared. Be sufficiently fit and experienced for your intended trek.***
2. ***Leave word of your route. This can mean life or death in case a search is necessary.***
3. ***Be weather wise. Check the weather forecasts, but don't always trust reports of good weather. Good weather can turn bad in an instant.***
4. ***Be equipped for bad weather and frost, even on short walks. Always carry a backpack and proper mountain gear.***
5. ***Learn from the locals. Experienced local trekkers can inform you of safe routes, weather conditions and things to look out for.***
6. ***Use a map and compass. GPSes are handy too, but don't rely on them. A flat battery and poor reception can cause problems.***
7. ***Don't go solo. Being all alone in the mountains can be a magnificent experience, but in case of an accident it's good to have someone who can give first aid or get help.***
8. ***Turn back in time; sensible retreat is no disgrace. If you are not sure if you can reach your destination because of weather or conditions, turn around! Others might have to risk their lives trying to rescue you. Also try to notify anyone that may have been expecting you.***
9. ***Conserve energy and build a snow shelter if necessary. Eat and drink frequently, and try not to work up a sweat. If you need to build a shelter, do so before you are exhausted.***
10. *Source: Value Planning Quotation 9.9 A.*

62. Specification Rules: a formal ‘quality culture’ for any specs.

- ‘Specification Rules’ (Rules, for short) are specific written guidance on how to specify something.
- Rules are the ‘laws’ of specification for a given organization.
- Rule violations are called ‘**specification defects**’.
- Rules should, like good laws, be gradually adjusted to capture smart and economic practices, and to keep up with changing times.
- Rules define a culture, and what they have ‘agreed to be’ the ‘right thing’ to do. The ‘best practice’.
- There are two major types of specification rules:
  - **Clarity Rules**: making the spec intelligible, so we can understand the content.
  - **Content Rules**: making sure the content is complete, correct, accurate, and aligned with other specs.
- If a spec is perfectly intelligible for all, it can still be wrong, and useless, or dangerous
- If a spec is unintelligible, it is *logically impossible* to determine if the *content* is right.
  - So you need a two phase review process: intelligibility, then content.
- Very basic **intelligibility rules** are: Clear, Unambiguous, and Right Stuff (‘requirements’ are *really* requirements, not designs)
- **Content rule** basics are things like: Designs must be effective in meeting requirements. Requirements must be aligned with stakeholder interests. Agile delivery planning will plan to deliver the highest value for the lowest cost, in small increments.

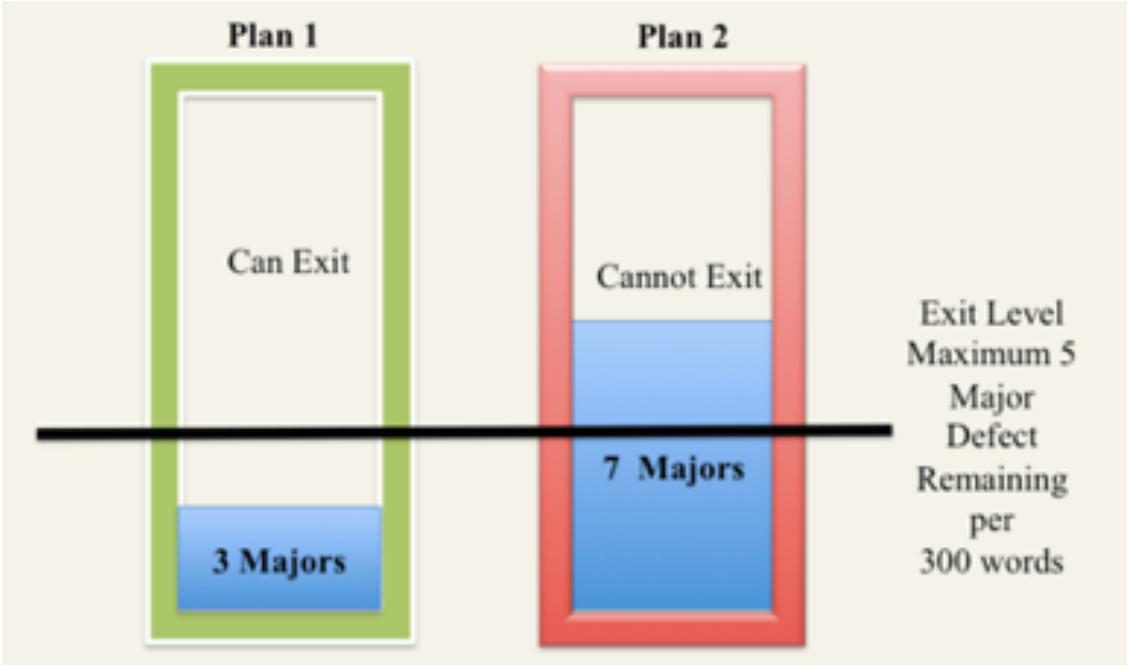


63. AN OVERVIEW OF PROCESS COMPONENTS IN THE SPEC QC PROCESS 'SOURCE DOCUMENTS', UPPER RIGHT CORNER

Source: Competitive Engineering, 2005, Fig. 8.8

63. Source Specs: a smart way to find defects in Spec QC.

- Simply reading a specification, might help you find out if it is unintelligible
- But it will not tell you if it is complete or aligned with earlier, or higher level, specs
- So in order to check for alignment and completeness, we need to bring in one or more ‘source specs’, and analyze the correlation between them.
  - A simple example: a design must have a requirement, and all requirements must have a design
  - Another example: the top-level project requirements need to be consistent with any contracts
  - And planned tests need to be consistent with the requirements
- If you do not consciously bring in all necessary source documents, and *give people sufficient time* to study the correlations, then your quality control is worthless, and your project is in serious danger of doing *wrong* things, and not doing *right* things.
- This may seem obvious when stated. It is a logical truth. But I constantly see otherwise smart organizations who do not systematically practice this idea. How about your organization?
- You will have noted how fanatic I am about ‘noting sources’ everywhere?
  - that is our ‘list of **source specifications'** (these sources are ‘Technoscopes’) that need to be *checked*, in order to evaluate **consistency** with higher powers
  - It is not just a matter of ‘getting out a few key source documents’. You need systematic cross-checking.
  - It may be a matter of digging up cited sources, and checking against dozens or more, source references, like internet links, to ask
    - Is the source being quoted correctly?
    - Have things changed with uses or updates of the source, since then?
- Source Specs increase the power of your Technoscopes. You can ‘see the otherwise ‘invisible’ problem early.



64. VIOLATION OF OFFICIAL STANDARDS SHOULD NORMALLY INDICATE MAJOR DEFECTS
WE SHOULD NOT BE WRITING STANDARDS FOR TRIVIALITIES

Source: Competitive Engineering, Fig G29

64. Major Defects.

- Some specification defects can potentially have **much more serious consequences** than others.
- If we can reasonably well, classify defects, as Major, or *not* (‘minor’) then we have a tool, a Technoscope for *prioritizing* the Major Defects, and perhaps *ignoring* the minor defects.
- I find that experienced people are pretty good at doing this Major classification well enough, when asked.
- In addition, by designing Rules to help us find Majors, we get *some* guidance
- In addition, we can make specialized checklists, to help us distinguish and identify major defects.
- An example of a major defect, would be a contract violation, or a policy violation
- A minor defect would be a misspelling in a footnote

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%***				

65. A REAL INTEL CASE OF USING MY METHODS (PLANGUAGE AND SQC)
INITIAL PLANGUAGE REQUIREMENTS (REV 0.3) HAVE 10 DEFECTS/PAGE BUT THIS IS REJECTED AS 50X WORSE THAN AN ECONOMIC RELEASE STANDARD (REV 1.0)
THIS MOTIVATED ENGINEERS TO IMPROVE BY 5OX BEFORE RELEASE
Source: Terzakis, Intel, Rio 2013 Paper[6]65 Defects Per Page (DPP[7]), jterzakis@verizon.net (2019)

---

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

[7] not the same 'DPP' as Defect Prevention Process, discussed below! Intel used this one, IBM the other. Sorry

65. Defect Density: Major Defects Per Page.

- A useful measure of ‘trouble’ in your specifications is ‘Major defects per page’.
- I usually define a ‘Page', as a virtual page of 300 words.
  - That works, but some clients (Intel, Los Alamos National Labs) chose 600 words. It does not matter at all, as long as it is defined.
- DPP is a **good predictor of future problems[8],**
  - for example, ‘released bugs in software’, aircraft completion delays, or ‘logic defects in silicon chips’
- By analyzing the costs of dealing with the defects early, or waiting until the defect-caused errors, actually cause losses downstream, you can work out, what level of defects, pays off to tolerate (the ‘Exit Level’), and move on.
- And worse than that exit level, you *know* it does not pay off to ‘move on’ (to release the specs for other purposes) so you can use that level as one possible ‘Exit Condition’ (see below) from the work process.

[8] Software Inspections a Case Study by Ronald A. Radice http://www.gilb.com/DL240 a prediction example at IBM

Here is the Defect Estimation Process, including the SQC sub-process in *detail*:

Entry Conditions:
1. A group of two, or more, suitable people, peers, are needed to do it.
2. You need to schedule 30 to 60 minutes. Irrespective of plan size.
3. You should have a trained (1/2 day) SQC team leader at the meeting to manage the process.
(or if that fails, follow this Process 10.5 and the Template 10.5 instructions, below.

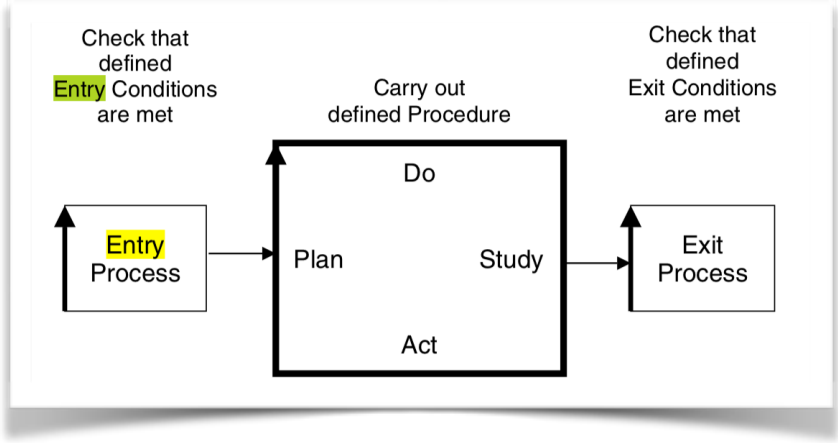
Defect Estimation Procedure:

P1: Identify Checkers: Two people, maybe more, should be identified to carry out the checking. One should be the Owner or Author who drafted or updated the plans to be checked.

P2: Select Rules: The team identifies (or acknowledges a standard) at least three Rules to use for checking the specification. (My favorite Rules are clarity ('clear enough to test'), unambiguous ('to the intended readership'), and completeness ('compared to sources'). For 'requirements', I also use the Rule of 'no design'. Any appropriate set of rules can be used.

P3: Choose Sample(s): The team then selects sample(s) of about one 'page' in length (300 non-commentary words). Choosing a page at random can add credibility – so long as it is reasonably representative of the overall content. The team should decide whether all the checkers should use the same sample, or whether different samples for some people are more appropriate or useful. If 300-600 words or less do it all.

P4: Instruct Checkers: The team leader briefly instructs the checker(s) about how to use or interpret the Rules, the time to be used checking (10 to 30 minutes), and how to document, list, count - any perceived defects, and how to determine if they are major defects (majors).

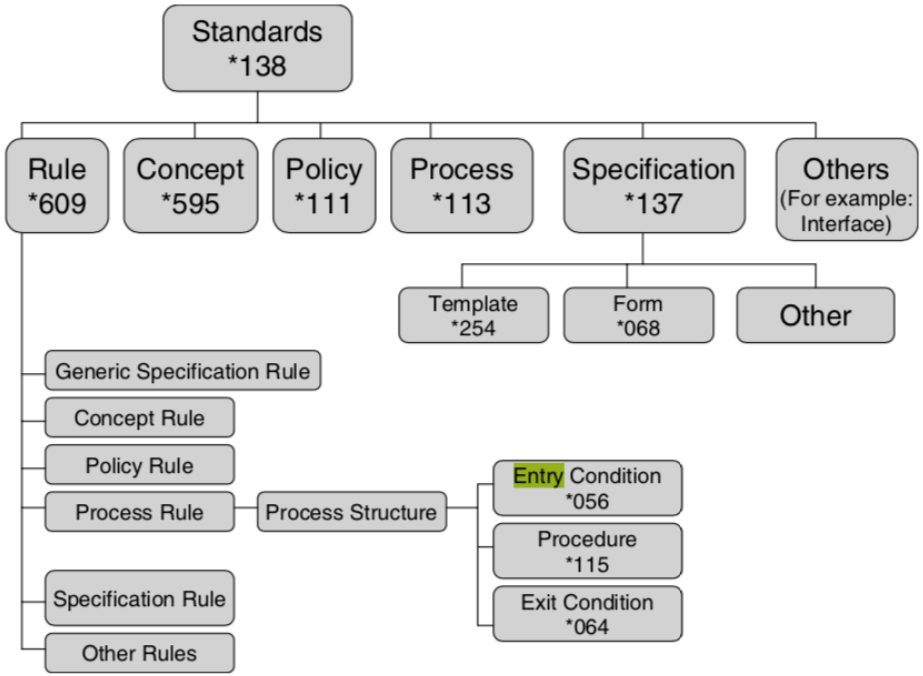


66. Entry Conditions. Source [CE] SQC Chapter

66. Entry Conditions: A Technoscope to see if you are ready for the next process.

- 'Entry Conditions' is a device to help us decide whether we are wasting our time, by proceeding to a next process.
- Entry Conditions can consist of any number of questions, or tests, or measurements, that we need to check (that *pay off* to check) before allowing the next stage of work (a defined process) to proceed.
- The idea is not unlike airport body security, baggage checks, and passport checks before you board the aircraft.
  - Painful, but the alternative is worse.
- Entry Conditions are, in principle, different for each different process.
- Entry Conditions are a formal *cumulation of wisdom*, based on *experience*. They are standards, and best practices.
- Entry Conditions help you resist deadline pressure, when in fact, if you violated the Conditions, you would fail; and lose even more time on the project.
  - In simple terms that's the idea: a 5 minute check, and avoid wasting next week's work, for 4 people.
- It is the *organization* that demands, as their necessary standard, that an Entry Process (Checking the Entry Conditions) determine if the Entry Conditions, are clearly met; before proceeding.
- Failing to do a valid Entry Process, carries a clear responsibility for people involved. It would be like 'cheating'.
- One smart entry condition is to double check that the defect density of all specifications entering the process is low enough to avoid wasting our time with Garbage In, in this process.
- Other people's Exit level might not be the same as *your* Entry Level, and in addition: don't trust other processes to measure correctly: double check.
- Entry Conditions are a valuable Technoscope for helping teams avoid failed work, wasted time, bad situations.

March 9-10 2019 Edit



67. STANDARDS TYPES, INCLUDING 'EXIT CONDITIONS'

Source Value Planning book (diagram) and Competitive Engineering book (text)

March 9-10 2019 Edit

67. Exit Conditions.

- Exit Conditions, is the same principle as Entry Conditions. Just the other end of a process.
- The main idea is that we should not consider ourselves finished, and certainly not hand over any specifications we have developed in a process, if we fail *any* of the Exit Conditions.
- One analogy is ‘airport exit’, with Passport Control and check you got your own baggage.
- As with Entry Conditions, an organization can develop any number of useful Exit Conditions, and improve them, when they learn that *other* Exit conditions are ‘even smarter’.
- Exit Conditions is a kind of *quality control device*, QC usually for any specifications produced, but also can include other things including any type of approval, clearance, declarations sign-offs. Anything that makes *sure* we are REALLY REALLY DONE, and will not have to return and re-do or fix the process soon.
- Or worse than quickly discovering we screwed up! We want to avoid learning, after a *lot more time and effort* has gone by, that we screwed up this process now, and have lost far too much time, money and credibility.
- The consistently dominant Exit Condition is that any documentation or specifications produced by the process, has achieved an economic and stipulated, economically low enough, Major Defect Density **Exit Level**. No Garbage out!
- This powerful Technoscope motivates much better quality of work processes: it helps you see if you are ‘really done’, and not just ‘moving on’ due to un-smart deadline pressure (which in fact threatens the deadline, even more, in the longer term).
- I have witnessed this powerful device installed at serious organizations like Boeing, Intel and Ericsson driving quality improvements at least 100 to 1 (serious independent measures) within weeks of installation. For thousands of engineers (and thousands of engineers per year, increments).
- So it is perhaps my favorite Technoscope Tool, and is one of those things that:
  - You can experiment with see how well it works
  - You can install quickly in a large organization, and get big overall benefits which are very measurable (quick win)
- The ‘Lean’ way to use this is upstream, early (bids, contracts, requirements).
- A note. If you are doing very-short-cycle agile value-deliveries (weekly?) then, for certain purposes, they can measure the results far more realistically, than trying to find problems via Major Defects in specs. Even Contracts can be checked this way through, incremental no cure no pay delivery, with contract adjustment at each cycle. (see below)

P8: Decide Action: If the number of majors/page found is a large one (ten majors or more), then there is little point in the group doing anything, except determining how they are going to get someone (Author, Owner) to re-write the specification properly. There is no economic point in looking at the other pages to find 'all the defects', or correcting the **majors already found. There are too many majors *not* found.**

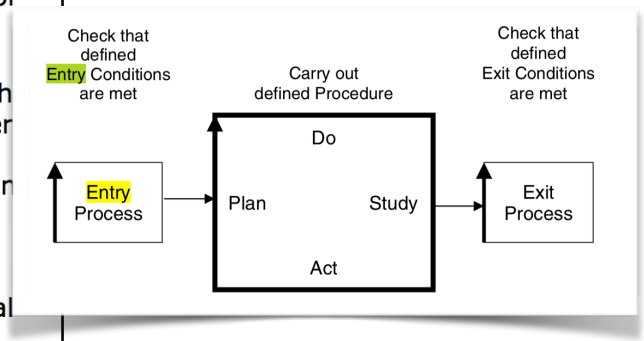
P9: Suggest Cause: Choose any major defect just found, and think for a minute, wh happened. Then give a short sentence, or better still a few words, to capture your ver dict on 'why?'; root cause. Document this, and seek practical action, or experiment, yourselves or others, to see if the insight can reduce defect insertion frequency in ger al.

Exit [From this QC Process]

1. if less than 5 majors/page (or other agreed Exit level standard), extrapolated total defect density was determined, you can Exit the plan to the next processes.

2. You should note the release level in the plan (Quality Level: < 5 Majors/Page)

3. Otherwise, refuse Exit. Expect an upgraded plan to be resubmitted for Spec QC.



68. THE STRUCTURE OF A SET OF STANDARDS AS DEFINED BY THE PLANGUAGE CONCEPT GLOSSARY EXIT CONDITIONS, SUCH AS THE EXIT LEVEL, ARE PART OF THE PROCESS DEFINITION

Source: Competitive Engineering, Glossary

68. Exit Level: A Technoscope on future problems and delays.

- As mentioned above, one possible, and probable Exit Condition, from a work process, is that the Major Defect Density Per Page (DPP) has to be so low, that it ‘pays off’ to suffer the remaining defects, catch them later, and move on now.
- You might wonder why we do not simply wait for ‘zero defects’ ?
  - Nobody knows when we are really there. Absence of *finding* defects, does not prove there *are* none.
  - It might take forever, and cost infinity.
  - If there are few enough defects, and consequent problems, it will pay off better, to tackle them downstream (tests, even customer use, well ‘Beta Test’ then!)
- A larger organization will be able to compute the right Exit levels for each process, based on costs of development and customer economics.
- A smaller organization can simply approximate a level, which is ‘much better than they are doing presently’, and see how that works in practice. That will at least drive down defects by 10x initially, and 100x later.
  - A crude minimum, is a level of 10 Majors remaining/ Page (300 words)
    - That is good because you probably have 100 to 200 Majors per page now (trust me, or measure properly yourself)
  - Most organizations would improve considerably if they used Max. 1.0 majors per Page: *that* takes hard work
  - Exceptional organizations like NASA and Intel have found they need to be at about max 0.1 Majors per 300 words.

A Sampling Case Study

104

- **A Client in Northern Europe**
  - Air traffic control trainer system for export
  - 80,000 pages contracted documentation before code
  - 40,000 pages already written
  - Project seriously late already (customer informed)
  - About 7 management signatures approving the 40,000 pages (pseudocode for coders)
  - Inspection of a sample of three pages
    - chosen by random numbers
    - declared to be representative
    - 19 Major defects found in half day inspection by the 7 managers
    - director checks the defect log and confirms

69. THIS CLIENT BROUGHT ME IN, WHEN THEY REALIZED THEY WERE AT LEAST 3 YEARS LATE.
USING SAMPLING, THE SAME DAY, WE FOUND THE TRUE SIZE OF THE PROBLEM.
IT WAS MUCH WORSE THAN THEY THOUGHT. SEE MORE IN '78'.
IF THEY HAD USED SAMPLING INITIALLY, THEY COULD HAVE AVOIDED THE PROBLEM.
THE DIRECTOR LOST HIS JOB, THE PROJECT WAS SOLD OFF TO ANOTHER SUPPLIER TO AVOID LOSSES AND SHAME !

Source: slides Inspection Economics for Lean QA

69. Sample: a very cost-effective Technoscope.

- Early specification Quality Control[9] was focussed on checking every detail, in all pages of all professional work produced. The objective was to find as many defects as we could, early, thus reducing the final cost of those defects by 10x. This worked pretty well in many situations. But the cost of doing the 'inspections' properly, maximum effectiveness, was about 4 Engineering Hours per Page. Not everybody found this profitable.

  And when we understood 'Sampling', nobody found that cost, for all pages, justified.

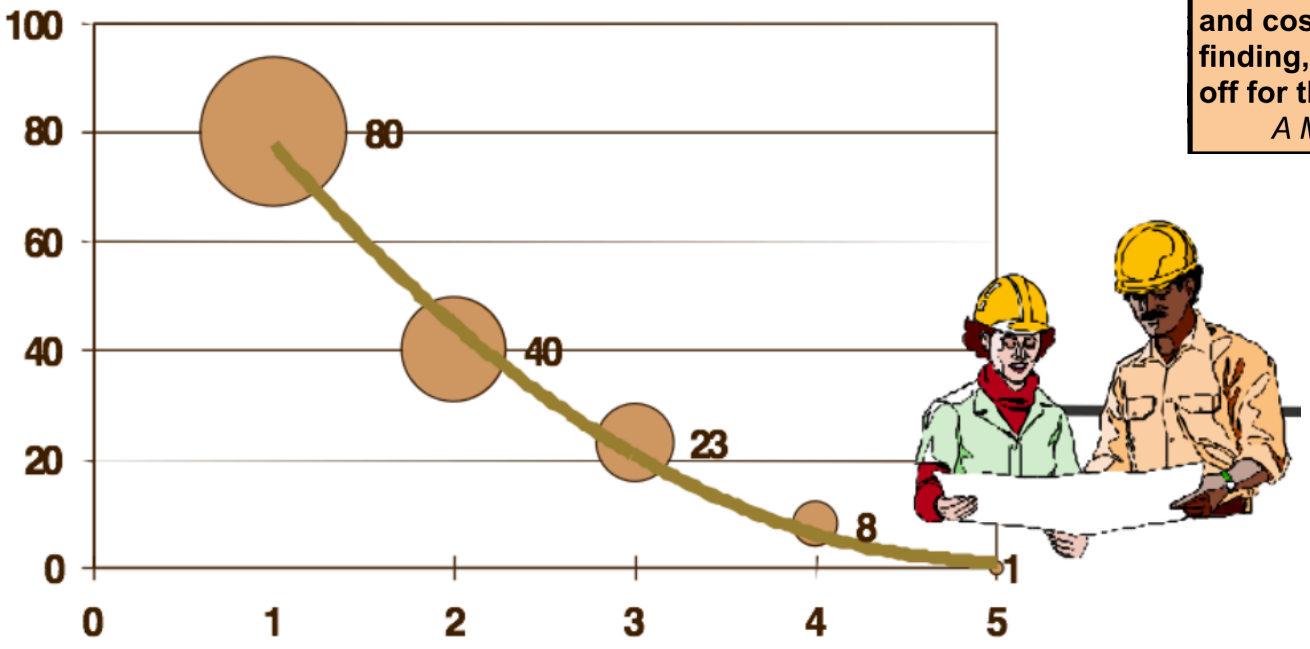
- At one point, about 10 years later, we 'bumped into' the concept of just *sampling,* large amounts of documentation
  - That turned out to be a winner. Here is why.
- Sampling, as is well known in statistics, can measure accurately enough, for many purposes
- Especially if the sample is 'representative'
- If you can get a quick fix on the defect density, accurately enough, early in a large work process, then
  - You can **refuse exit**, and **motivate** staff, to **seriously upgrade** their performance (by 10x, 100x)
    - You effectively motivate people to *do it right the first time*, thereafter
- The consequent 'defect insertion reduction', is far greater than any ability to *find and fix* defects, with the older methods (which were 100% 'samples', aimed at *fixing bad work*). We are now, with Spec QC, and sampling, very 'Lean'.
- We are very measurably *motivating* people to *substantially* higher quality level work. Sloppiness is no longer tolerated.
- The costs are arbitrarily low, but are something like 1% to 5% of pervious QC costs. Cost-effectiveness skyrockets.
- The method can be used early, frequently, and continuously, for a large piece of work (you are early: also 'Lean')
- This Technoscope can change large organizations, at very low cost, as pointed out above (Exit Level)

---

[9] Gilb and Graham, Software Inspection, 1993

"We find an hour of doing Inspection is worth ten hours of company classroom training."
A McDonnell-Douglas line manager
"Even if Inspection did not have all the other measurable quality and cost benefits which we are finding, then it would still pay off for the training value alone."
A McDonnellDouglas Director

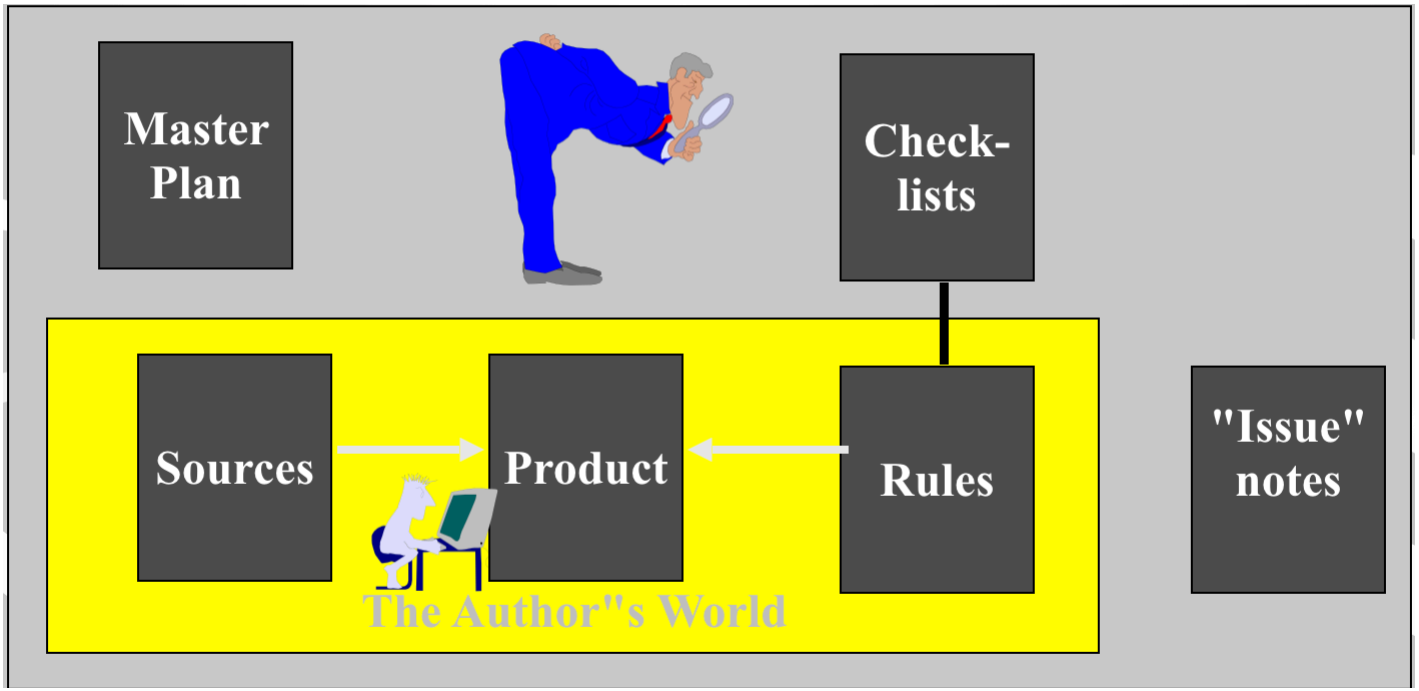
Source: Value Planning, 7.7. From Real Case at our client Boeing (McDonnell-Douglas)



70. GARY, A REAL EXPERIENCED, CLIENT AIRCRAFT ENGINEER, SHARED HIS EXPERIENCE OF MY 'INSPECTION' METHODS WITH US JUST LIKE THE INTEL EXAMPLE ABOVE, HIS ENGINEERING DRAWING WAS REJECTED, NO EXIT, UNTIL HE LEARNED TO PRODUCE IT WITH ABOUT 80X FEWER DEFECTS.
SHORTLY, THOUSANDS OF BOEING (MCDONNELL-DOUGLAS THEN) ENGINEERS JOINED IN
SHORTLY THEREAFTER BOEING IN WASHINGTON STATE IMPLEMENTED THE PROCESS WITH MY HELP

70. Individual Learning Process: A Technoscope into your own capability.

- You may have noticed: people do not learn much detail in a classroom setting, before practicing it.
- They do not learn from thick user manuals
- They learn best by doing, on the job
- But, we don't learn it *all at once.*
- We learn gradually. Something like about 50% better after each iteration, of getting good feedback.
- Think 'coaching an athlete', or a musician
- The interesting problem is 'giving professionals really good feedback', which they *take seriously*
- We found that Specification Quality Control, coupled with setting serious Exit levels, does that trick effectively.
- People reduce their defect insertion level (their bad practices) by about 50% every time they get measured (by their peers, trying to help them be effective team members)



71.CHECKLISTS HELP A 'SPECIFICATION CHECKER' TO EXTEND THEIR UNDERSTANDING OF THE 'RULES', WHICH ADVISE SPEC AUTHORS, ON 'SMART THINGS TO DO', AND 'DUMB THINGS TO AVOID'

Source: Inspection Mechanics, TsG slides

71. Checklists: a Technoscope for understanding specification rules.

- A 'checklist' is a set of questions, which can be asked, about a specification's contents, by a checker, with a view to improving the effectiveness of that checker, in identifying major defects (i.e. Rule violations, of potentially Major consequence).
- Checklist questions are always *directly derived* from individual *official rules.* A specific source Rule Tag is annotated. Like '…**<- Rule 42. '**
  - They are not allowed to **be** the rules, or to **change** the rules, just to interpret them.
- Checklists are '**stored wisdom**' (Technoscopes) aimed at helping to interpret the rules, and to explain their specialist application).
- Checklists are used, to increase a checkers effectiveness, at finding major defects, in a specification. To teach, more deeply.
- Specification Authors have no direct responsibility, or use, for Checklists. They are bound by the Rules.
  - But, spec authors who have participated in Spec QC, using checklists, will become more aware of the Rule intents.
- Checklists are like *law court interpretations of the law*. They are *not* the official 'law' itself, but they do help us understand the *proper interpretation* of the law.
- *Anyone* can write checklists, at any time, to give *advice* on how to check. They are i*ntentionally less formal to create, and to change,* than making specification rules. They do not necessarily have formal 'owners.' They could even be made 'for a day', for one group'.
- Checklists should *not* be used *instead* of a proper set of rules. Rules, which are maintained by a responsible engineering process owner.
- They are only intended as a *supplement* for checkers. Issues found in checking, can only be classified as *real* defects, if they can be shown to violate the *official agreed rules* for a specification.
- When a checker finds a Checking Issue (possible Defect), they should 'note the Tag' of the Checklist Question, which also contains the Rule tag.
- Less formal 'de facto checklists' also exist. These include any documents that can be used to check a document with a view to identification of defects.
  - *These* documents can have other names, and even other purposes, than a 'pure' checklist.
  - Examples of 'de facto checklists' include 'sources,' 'standards,' 'guidelines,' 'templates with <hints>' and 'model documents, and app logic.' If they help check, they must be 'some sort of checklist', irrespective of what people call them, or intended them to be used for.
- The effectiveness of Checklists, in improving our ability to detect Rule violations, is easily measurable.

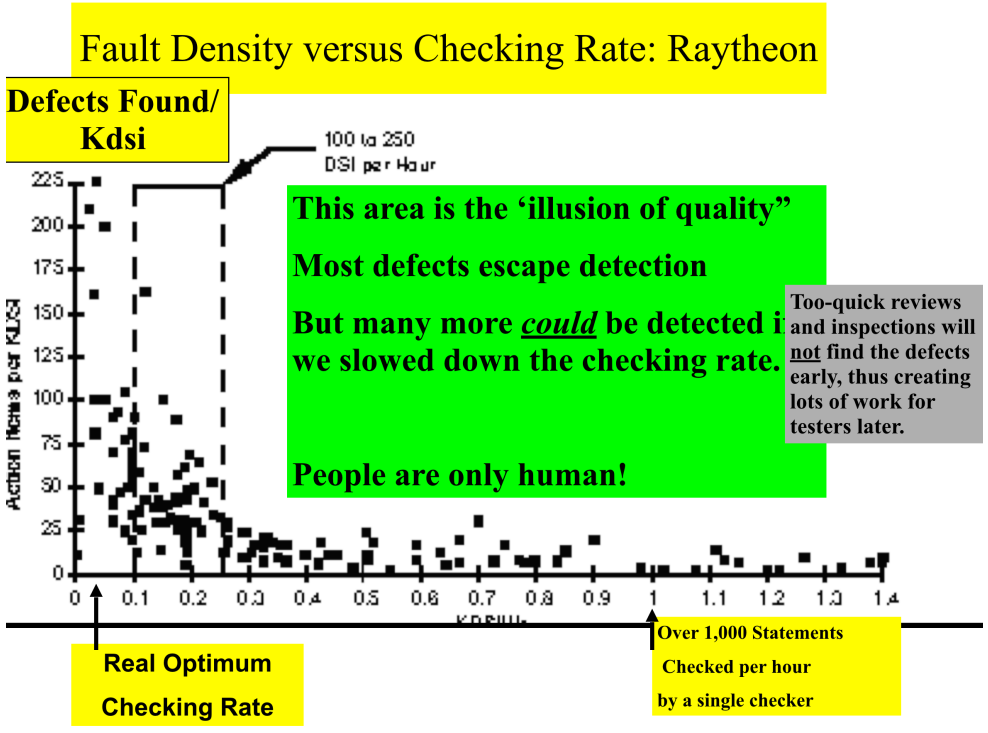
Example:

Rule

R42: All Quality requirements must be quantified, with a Scale definition.

Corresponding **Checklist**: (helps people understand the concept of 'Quality').

C73: Does a word end in '-ility',? Does a phrase imply 'how well' things will be <-R42
C88: Are the concepts on this list? Security, Privacy, User Friendly, Robustness, Flexible,.. <-R42
C42: Are implied qualities, mentioned in a strategy, or solution, description? <-R42
(example) 'The Architecture is expected to be *robust*, and *effective.'*



72. A REAL EXPERIENCE PLOT OF 'SPEED OF CHECKING' VERSUS 'DEFECTS FOUND'. 90% OF REAL PEOPLE ARE 'GOING TOO FAST', AND FINDING TOO LITTLE. GOING 10 TO 100 TIMES TOO FAST. THESE GUYS MAKE WEAPONS!

72. Checking Rate: a Technoscope for understanding your understanding 'capability'.

- The 'Checking Rate' is the speed that checkers run at, while looking for major defects.
- If we know the checking rate, we can determine our ability to find defects, as a % of all defects that are there now.
- The reason this is interesting is that people, especially under deadline pressure, check specs far too quickly.
  - This results in finding less than 5% of the real defect content.
  - But people have an illusion. They think: 'I did not find any more, so there cannot be so many more, therefore it is clean enough to release'. VERY WRONG!
  - The result is that organizations release *very polluted* specifications (100+ majors per page is minimum)
- With the knowledge of actual measured correlation, between checking rate, and % defects found: you can decide how much time you should give people to check, based on the volume of specifications.
- In general you would not want to go to extremes. Too much checking time is a loser: too little you find far too few defects.
- There is a visible **optimum** checking rate.
- People have illusions, about their own expert capability, to 'spot defects': illusions which are easily disproved when checkers are slowed down.
- One reason for *slowing down,* is that people need time to read, and apply all the **rules**, all the **checklists**, and all the **source** documents.
  - If you, for example, have 40 Rules, then you need to check *each* rule on *every* sentence and word.
  - That's just for starters. It takes time to do it properly.
  - It might be useful to automate a lot of the checking, and we are doing it.
- So, people, under pressure, will go too fast: the organization will pay the penalty later, downstream.
- Smart management needs to be aware of this.
  - And to decide to enforce optimum rates, for the benefit of the larger project picture.

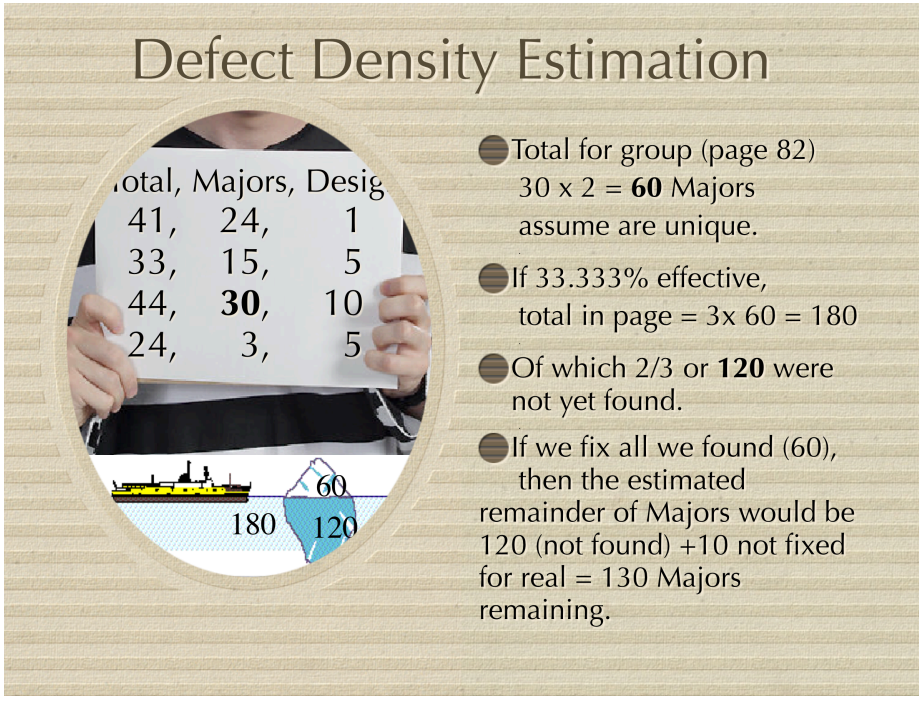
73: CHECKING TEAMS ARE AT LEAST A NICE SOCIAL ACTIVITY.

73. Checking Teams: 4 eyes see more than 2 eyes.

- One person, when checking spec quality, has their limitations.
- They will, *alone*, at *best*, find about *half* of the ‘available defects’, that a larger ‘team’, with them onboard, will find.
- In earlier times, with ‘inspections', we wanted to maximize defects found. We could then fix them early.
- Today, with the focus on sampling, measurement, and motivation to learn (not finding and fixing), Lean, prevention, early, are keywords: it is *not critical to maximize % available* defects actually found. This needs 3 to 6 people eye-balling the pages. We have plenty of Inspection[10] studies to conclude that.
- So, about 2 people is a good option. Not so much because the second person helps you find more defects.
  - I think the mutual support, and motivation, is a good idea, compared to one lonesome cowboy. My opinion.
  - One reason is that we always want the spec author to join the checking team, (no outsiders judging *me*).
    - The author is surprisingly the best ‘defect finder’ of all on their own specifications.
  - The *second person* brings an ‘outside view’. And keeps the author ‘on their toes’, *before* checking, and *during* it. (my opinion)
- I think the *second* person gives us a double check, because if one checker finds 30 defects in a page, and another finds 2 (it really happens). You know something is wrong, and can analyze root cause, and fix the process.
- In current Spec QC, the important concern is to get a ‘good enough’ measure of defect density, to decide, most of the time correctly, that a specification, will not pay off to release, downstream.
  - One to two people, checking a small sample, is good enough, to do that job, quite well enough, most of the time.
- One other reason, for having more people on the team, is that they **learn** Rules (for their own later authoring work) , and they ‘get familiar’ with the specifications.
  - At the level of effort, for a small - few page - sample, **that** effort costs little, and seems like a good idea.
    - You can experiment, and see how it works for you.

---

[10] Gilb and Graham, Software Inspection, 1993.



.

74. THIS SAMPLE OF REAL REQUIREMENTS, LED TO A CORRECT PREDICTION OF A 2 YEAR DELAY, OF WHICH 1 YEAR WAS ALREADY EXPERIENCED.

Source: Real Client Case Study, Cincinnati Ohio

74. Spec QC Measurement: calculation of defects.

- We don't '*count*' defects any more. There are uncountable un-findable numbers of defects.
- We do not *fix* defects any more: there is no end to that. We will never get to zero.
- We have a new game in Spec QC. We help people *learn to avoid defects*, we *motivate people to avoid defects*, we help teams to help themselves. To learn, and to get better. (That's the 'Lean' paradigm)
- And you know what? That is FAR more cost-effective than the: 'lets clean up our mess' tactics, at test or in the field.

We do need to make a reasonably credible estimate, of our defect density (Major Defects Per Page).
And we need to make that good-enough estimate as quickly, as early, and as cheaply as possible.
So that we can quickly, and early, return those bad specs, to their struggling authors, and motivate them to learn more, quickly.

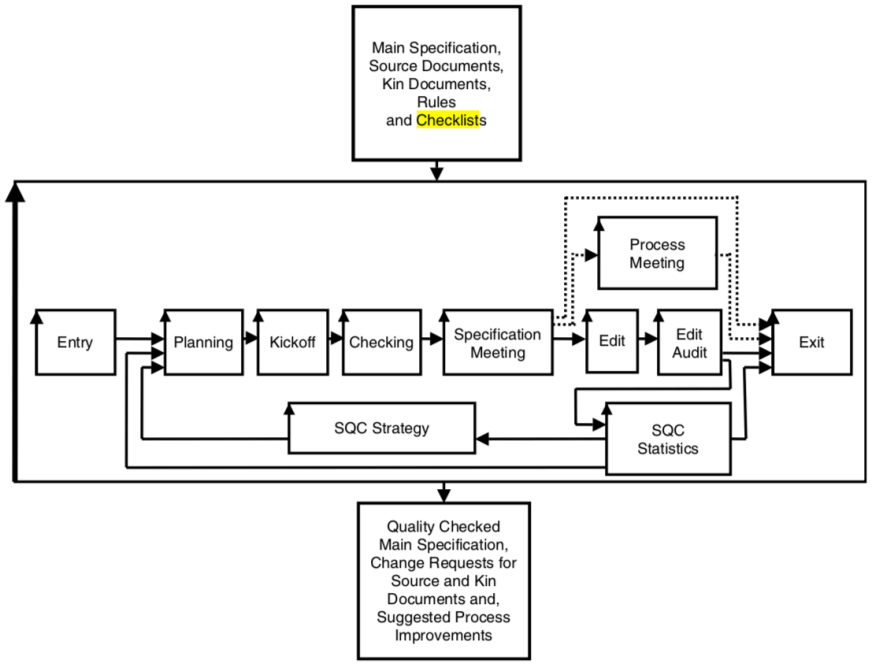
To do this we **extrapolate** the defects and their consequences, from the sample taken.

In very simple terms we do this:

1. The largest number of defects found by one person, is about ⅙ of the total in that page.
2. So, if that is '10 defects maximum found,' there are probably 60 in the page (±20, who cares! Limit is 1)
3. If you have 100 pages, you have 100 x 60 = 6,000 Major defects
4. Of which only 1/3 will cause about 10 work-hours delay, extra effort, to your project (20,000 hours delay)
5. No exit, by FAR.
6. Cheaper to '**burn'** the document (delete button), and get some sober people, to write according to the Rules.

This was brief, and I did not tell you where I got my numbers from (Capers Jones[11] is a good source). Trust me. This is absolutely real. We have decades of research and experience to prove it. It is in my books (SI, CE, VP) in detail, but you want to read a short booklet now, so there we are.

[11] www.NamCook.com



75. A PROCESS DIAGRAM OVER THE DEFECT DETECTION AND CORRECTION PROCESS THE 'EDIT' AND 'EDIT AUDIT' PROCESS IS THE CORRECTION COMPONENT

Source: Competitive Engineering

75. Inspection Defect Removal: The Defect Detection Process (DDP).

- From about 1976, when IBM publicized their Software Inspection process (IBM System Journal), and for at least 10 years, we practiced and spread a 'Defect Detection Process'.
- The emphasis was on 'find as many major defects as you can', because those you do not find now, will cost you 10x more when found in test, and 100x more when customers find them.
- This was a clear well-studied measurable success, in a lot of large global companies (IBM, HP, Ericsson, Nokia, Philips Medical, Siemens).
- It did not so much change the quality levels of delivered products: the companies were quality conscious 'whatever it took'.
  - What it did was to reduce a lot of downstream (test, field use) costs and delays.

When we gradually realized that there was an even-smarter option for all players, SQC (sampling, measurement, No Exit Levels, Motivation, Reducing Defect Injection Levels), then the DDP effectively died. We certainly were done teaching it. It seems so cost-ineffective by comparison. But we were not very good at really spreading SQC. In fact the world went on a binge of testing, too late too little effectiveness, and really did not seem to care that prevention was a much smarter idea. I have not been impressed with the process leadership, which has let the testing binge happen, and completely ignored lean prevention methods. They have not consciously rejected Spec QC - lean prevention, they did not seem to know about it, or seek it out. There were no market forces pushing it, and selling it. Agile was the solution to 'everything', and it was marketed well. Intel was one great exception from about year 2000[12]. Boeing from about 1990-1992. Parts of Citigroup 2003.

Here are some of the economic factors that killed DDP (aka IBM Inspection)

- It cost about 100x more, to actually do the defect finding (no sampling, 100% coverage)
- The the stream of defects, was stable, un-abating, rarely improved. People were sloppy, we cleaned up their slop (we still do with test)
- Downsizing and Market changes in the large organizations, that had adopted Inspections. Culture lost, rather than brought up to date.
- Agile made people confront their bad practices much earlier. In a sense that gives small samples of reality, and a chance to improve process, practice and peoples capabilities: But my view is that this has not been exploited well, for quality.
-

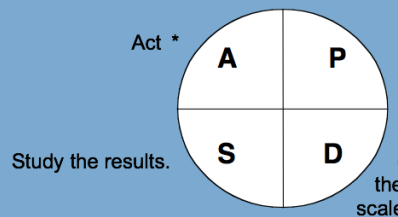
"Cease dependence on mass inspection. Require, instead, statistical evidence that quality is built in, to eliminate need for inspection on a mass basis". W Edwards Deming, 1 of his 14 Points.

---

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

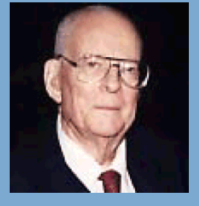
The P D S A Cycle from Deming

"The Shewhart Cycle for Learning and Improvement
The P D S A Cycle



Act. Adopt the change, or Abandon it, or Run through the cycle again, possibly under different conditions. "

Exact reproduction (- '(Do)' from a letter to Tom Gilb from W. Edwards Deming 18 May, 1991



76. THE TECHNOSCOPE FOR FIXING THE PROCESS: 'PDSA' : SIMILAR TO OUR 'EVO' VALUE DELIVERY CYCLE. NO BLAME CULTURE: UNDERSTAND THAT 'IT IS THE PROCESS', NOT 'THE PERSON'. DEMING.

76. No Blame Culture.

"**Blame the process, not the people**".

- Deming Seminar, Alexandria, Virginia, 19-22 January 1992

- People screw up. We are just 'human'.
- It is a tempting, knee jerk reaction to blame the individual for wrong stuff.
- But, Deming was always clear that the individual is a victim, or hostage of the 'process'.
- So, we need to blame the process, not the individual victim.

This concept, is a Technoscope for seeing 'what to do', when things go wrong.

The 'process' is *anything and everything* which equips people to do work:

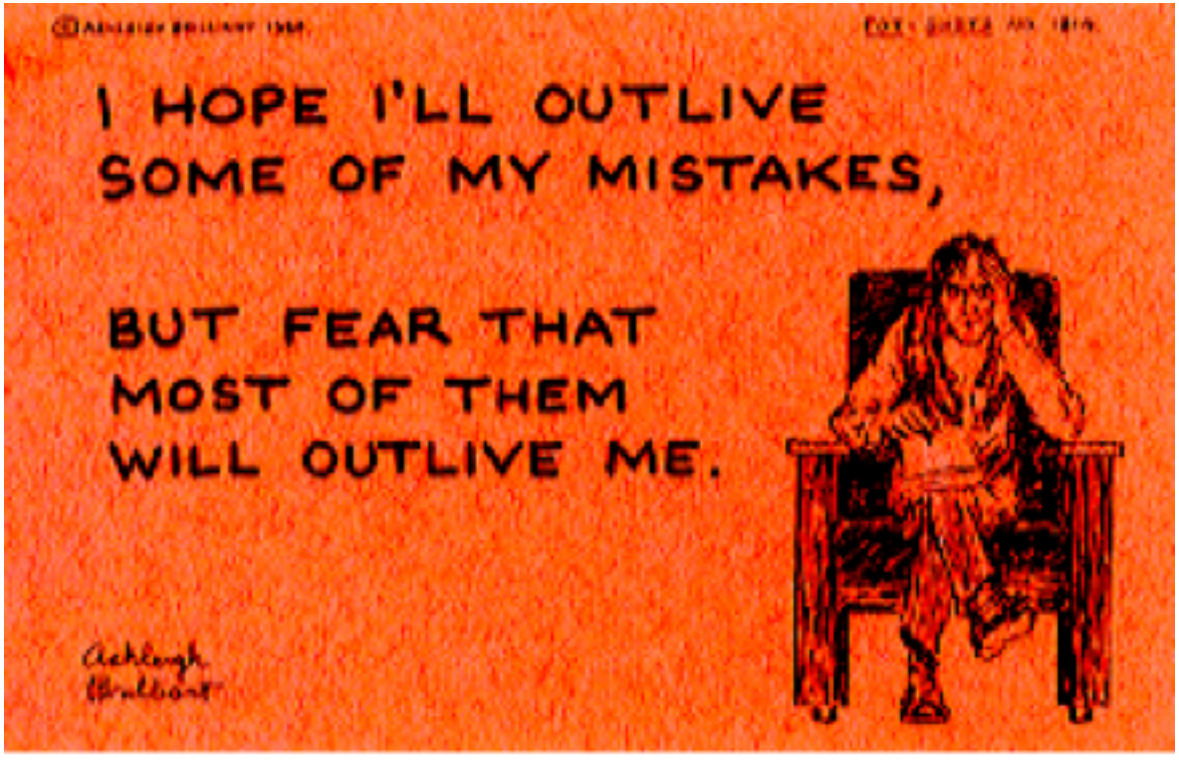
Training, recruitment, motivation, tools, leadership, empowerment, and much more.

In other words, 'management stuff' (unless you delegate that management to the troops!)

When a human screws up, your first Technoscope is **root cause analysis**, preferably by the foot soldiers themselves (empowerment). What is the '**process** fault' that is forcing people to make errors?

Find it, change it, and improve it; then test and verify that it works.

77. ANONYMITY FOR MISTAKES



77. Result Anonymity: seeing final results, not blaming individuals.

- If you publish result numbers that can be tied by managers to an **individual**, or a small team, YOU LOSE
- People will defend themselves, by finding ways to alter those numbers
- For example, tell who did the 'major defects per page' in a document that just got 'refused exit'.
- The question should **not** be 'how many major defects were found in your work?'
  - It should be 'when will you successfully reach the exit level?"

"Eliminate numerical goals, posters, And slogans for the work force, asking for new levels of productivity without providing methods."
Dr. W. E. Deming's 10th Point

"Drive out fear, so that everyone may work effectively for the company.", Dr. W. E. Deming's 8th Point.

Keep the message: 'management will never know the numbers for your beginning of the learning curve'. 'That is between you, your team and SQC checkers' (half of which is 'you' the author). In fact mutual blackmail helps keep it secret. Next time you are checking their specs.

What counts, is that you learn fast. We have all had to go through that same learning curve.

If there are problems, then management and the organization needs to make it easier and better for you. Maybe you would like to help in that effort, personally?

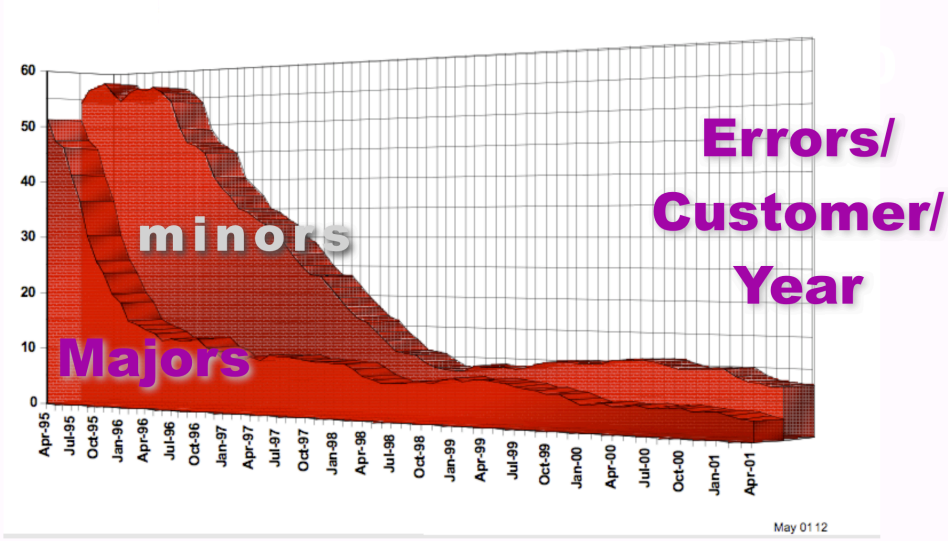
In one UK client corporation, we made it difficult for managers to snoop. The individual SQC results had 'cover names' like Micky Mouse, Donald Duck. But management should make it clear that they are not interested in such details. Only the final result, the big picture. The value to the stakeholders. Motivate the right things, or you get sub-optimization. People should use numbers about their personal work, for personal self development, and team development.

Improving the ***Reliability*** Attribute

Primark, London (Gilb Client)

see case study Dick Holland, "Agent of Change" from Gilb.com

Using, Inspections, Defect Prevention, and Planguage for Management Objectives



78. A SMALL (60 PEOPLE) LONDON FINANCIAL APP PRODUCER CLIENT
THEY WERE VERY GOOD AT EARLY AND CONTINUOUS SPEC QC (INSPECTIONS)
AND ALL OUR OTHER METHODS
THEY CONQUERED THEIR MARKET BY BEING SO GOOD AT IMPROVING QUALITY, MUCH BETTER THAN COMPETITORS

78. Early and Continuous Sampling: A Lean Technoscope.

- As mentioned above, it is an outdated, and inefficient idea to try to find product or system faults (defects that became faults in the system) by using *final testing* processes. Or even by extensive Beta Testing. Too LATE!
- We have methods, as mentioned above, for *preventing* the defects in the first place (SQC Sampling, Exit Levels, Motivation to Learn), and for finding them early (SQC, Inspection).
- But, none these methods are very effective if you (as in the Air Traffic Control (ATC, see Illustration 69) case, above) wait until 40,000 pages of air traffic control logic specification, by air traffic control specialists, have already been written, and 'approved'.
- I asked one of the 7 directors, who signed off on the 40,000 pages ATC Specs, why he had signed what we had just proven, with a sample, was a very defective specification He answered 'because the other directors signed off on it'.
    - I lost all respect for management approval signatures from that point on.

Imagine if the ATC Manufacturer had written just the first-100 pages of ATC Specs.

- and a sample of 5 pages at random, showed the same as we discovered in our 3 page sample:
- that there were with extrapolation, about 20 Major defects per page ? (scary idea flying in that airspace, right?),
- Meaning that in the projected total 80,000 pages of ATC specs we would get 20 x 80,000 major defects (160,0000)
- Of which about ⅓ (53,000) would become 'bugs' in the software
- Of which maybe half would be found by pre-delivery testing (27,000) and the
- Other half would be delivered to the customer (26,000)[13]
- DO THE MATHS!

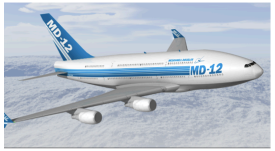
Crazy and dangerous, not least for the ATC Producer. Forget the ATC, and forget 'you the airplane passengers'.

- But they never thought of any sort of 'SQC', and never thought of taking an early sample.
- And they *were* a high quality, mature engineering corporation!

The Director of it all, when he reviewed the 20 major defects we found that first afternoon, said to us:
"My boss, the CEO of the Corporation would fire me if even ONE of these 20 defects ever got to the customer."
He did get fired too. He did too little, too late. *Not* 'early' and 'continuously'. Don't meet his fate. Early, often!

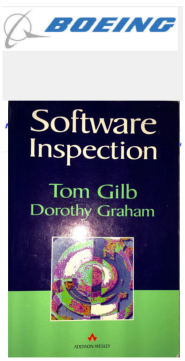
---

[13] if you think this is exceptional, I can tell you it is not. I have personally witnessed many large projects like this totally out of control. The simple concept, this Technoscope here, of QC early and continuously, is not well known. Just common sense. Now ask yourself, what is really happening here in your company's projects?



DPP case at McDonnell Douglas/ Boeing

- About 1989, El Segundo California
- Successful Spec QC done on Engineering Drawings
- Director (Mac) asks for more tricks from TG
- Suggested 'DPP' (recently learned from Mays IBM)
- Got team of 4 young engineers
- Problem: defects in engineering drawings
- Initial suggestions: big automated project to do Engineering Orders better
- KISS: Keep it simple (I told them)
- They came up with a form with cutouts and notes, for all 19 types of engineering order, to prevent bad engineering orders. IT WORKED!
- Made in plastic and color for all 2500 engineers
- With names of the 4 young inventors on them
- Mac had to protect the 4 engineers from their supervisors (daily grind)
- Mac said: this is so simple, we could have done it decades ago: why not?
- Tom: nobody ever asked the young engineers to think about the problem
  - No empowerment!



...

79. DPP BOEING

79. Defect Prevention Process (DPP): Empowering Troops Gives Best Insight.

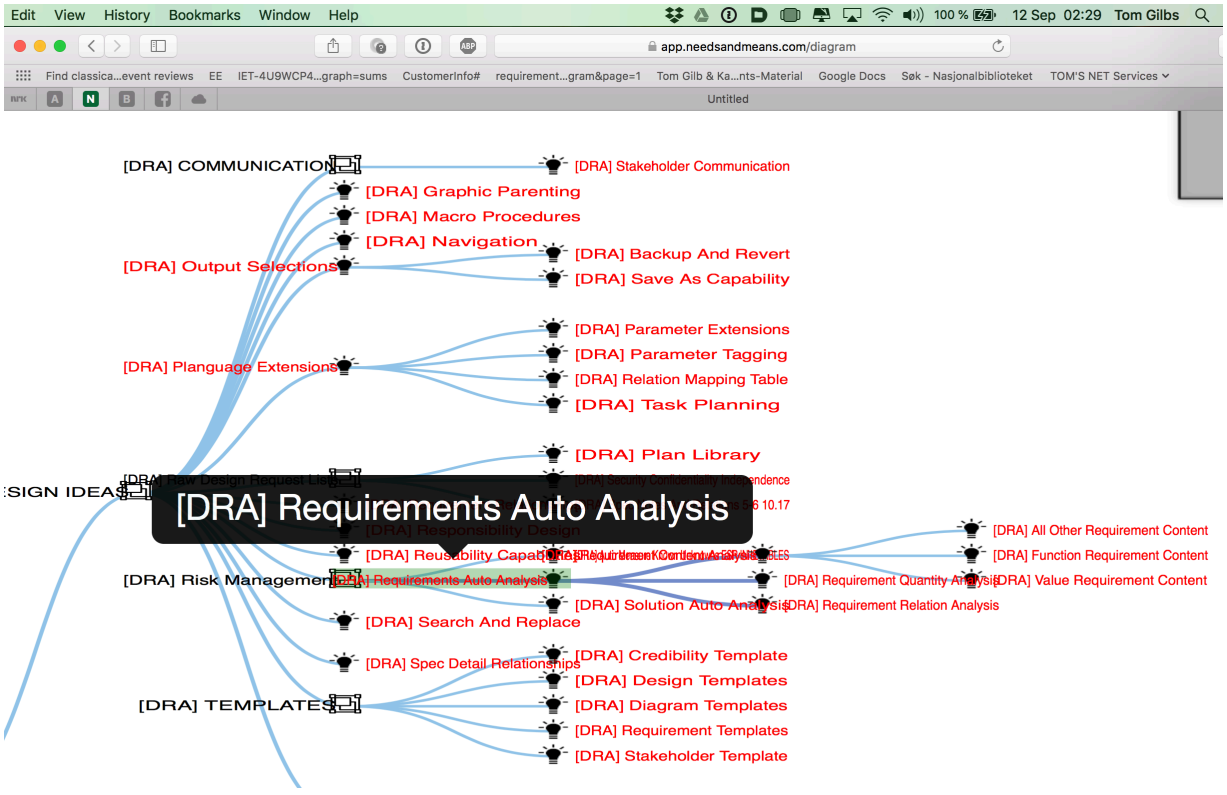
- In 1990 Robert Mays, Linda Jones, and a team at IBM Research Triangle Park NC, proved out and published (IBM Systems Journal[14]) their DPP made a major advance in Technoscopes, for understanding and improvement of complex and large-scale projects. They got a CEO-signed, big-cash award, in recognition.
- That step was what IBM had *tried* to do with the original 'Inspection' process, 16 years earlier: to *learn* from the defects data, and *improve* the organizational process.
- IBM had plenty of data in the 1980's. They *tried* to learn. But it did **not** succeed, for reasons unknown to all of us.
- The breakthrough idea, that Mays and his team had, was to **delegate power** of *analyzing defects*, and *suggesting process changes* TO THE TROOPS. The the thousands of smart IBM engineers. Mays reported 12:1 ROI !
- It worked brilliantly. For example at IBM Labs in Minnesota, with about 2,000+ engineers, they actually suggested and installed 2,176 process changes in about 2 years. They were also rated the best Lab in IBM
- This DPP idea was built into the IBM Capability Maturity Model (later into DoD SEI CMMI, for military suppliers and similar): as 'Level 5 (**Continuous Process Improvement**) The highest defined level of a product development organization.
- Using this process (DPP + Inspection), for example Raytheon Missiles reduced rework from 43% of all effort, down to about 4%. More than doubling productivity of 1,000 engineers.
- My first experiment with the method, at Boeing (see left page, 79) had same-quarter impact on the work of about 2,300 aircraft engineers, and brought the engineering-order defect-level down dramatically (93 or so to 3 or so, per order). This was the company's biggest-known problem, threatening their existence, the President told all employees.
  - Notice, **I** did not solve the problem.
  - But I empowered and coached 4 smart young engineers, who **did** solve it! Delegation works.
  - And I got great support from a Director (Mac, thanks!) to make sure we were allowed to try out these new methods, in spite of tremendous pressure to 'stop fooling around, and get those young engineers back to the daily deadlined grind'

This Technoscope helps us 'see' what is *really* wrong with our work processes, and how to really fix them in *practice*:

- **Not** by directors deciding what to do (there was a former NASA Director there with a big budget, who failed)
- **Not** by smart-Alec overpaid, Big consulting companies, to advise on it, at inflated costs, and it does not work

---

[14] http://agileconsortium.pbworks.com/f/Mays1990ExperiencesDefectPreventionIBMSysJ.pdf



80. AN OVERVIEW OF SOME OF THE ARCHITECTURE AND DETAILED DESIGN OF THE APP.NEEDSANDMEANS.COM TOOL FOR THE PURPOSE OF MORE-AUTOMATED SPEC QUALITY CONTROL.
THESE ARE SPECIFIED IN SOME DETAIL, AND REMIND US OF THE AUTOMATED QC CHECKS WE CAN DO, AND WANT TO IMPLEMENT GRADUALLY. IT IS A PART TIME PROJECT NOW.

80. Automated Spec QC: can we automate quality control of ideas?

- The Spec QC process described here, has until recently, and still largely, been a ‘manual’ human eyeballing process.
- People read the Rules, and Checklist, and Source Documents, and main documents, and ‘find defects’
- Now, if you are doing small, few-page samples, that works pretty well.
- But if the volume you would want to check is high, and if the change rate is high, and if you have globally dispersed teams keeping things up to date…
  - Then it begins to sound like a nice idea that there is some degree of automation, keeping a constant watch on the quality of the planning specifications.
  - Human systems tend to fail, and get corrupted, under deadline pressure, and profit pressure.
    - I’d sleep a lot better at night if I knew that I had a ‘sort of’ AI system keeping watch over as much of the planning detail as possible. Computers are fast, tireless, and good at checking clearly defined ideas.
- My Planguage was in fact designed with the idea in mind, that it was well-defined, and various planning-object relationships were so well defined, that we could apply a bit of logic incrementally, to check some things, at least.
- Here are some examples:
  - Completeness checks are easy: where there is supposed to be some spec, then we can signal a ‘blank’
  - Safety Factors: we can highlight in Red when a safety factor is not enough (see example in 52 above)
  - Rules for particular types of Specs: for Example in **‘Type: Quality Requirement**: it MUST have a Scale defined, and at least one target or constraint , defined: or it is NOT in compliance with the rules.
  - Stakeholder relations: if 4 different stakeholders have an interest in one requirement, then the Rules that any change shall be discussed with all other stakeholders, can be automated and checked, because we know which stakeholders are related to every single requirement.
  - All outstanding Issues, Assumptions, and Risks can be tracked and residual unresolved instances can be sent to the responsible Specification Owner.

There are hundreds of opportunities for QC automation, when the language you are using is structured and well-defined.

We have implemented some of these things, and planned others, which will be implemented in the near future.

What we have learned historically about this is that once the QC is automated, it ‘drops off the map’ as something **people** need to do in Spec QC. It just becomes everyday automated support. You hardly notice the transition.
One day we have a vision that human QC will all be automated. We have laid the basis. We’re working on it.

Part 5: Evo: Evolutionary Value Delivery. An Overview.

Evo, is a concatenation of **Evo**lutionary. Of late, an afterthought, for fun we have an acronym for those who need one.

E V O, **E**volutionary **V**alue **O**ptimization.

I first started 'doing it' about 1960 (at 19 years old) by delivering value in 20 value-to-customer delivery steps, on a project in Oslo (Dobloug, Invoicing System) which used an IBM 604 Electronic Calculator, with 60 plugged-with-wires programmable steps. And with punched card systems otherwise. I continued using this intuitively-appealing method, and it always worked well, and worked early. While other's projects, around me, failed miserably (2 failed projects in parallel, with my successful Evo project at University of Oslo, Publisher, IT, late 1960s).

I gradually realized that I had a 'method', that deserved teaching to others. So I taught, lectured, and wrote about it, in papers and books. I was documented as the earliest writer, about Evo and Agile[15]. Other people also intuitively used, and wrote about the method, and I collected that in my 1988 Book[16], which, it is documented, inspired the Agile Software Movement (Bekk, Cohn, Sutherland, and others explicitly credit my Evo as First).

Many early agile people used the term **IID**, Iterative and Incremental Development. But that just showed they did not understand what I was talking about, and did not understand the meaning of Evo. The US DoD *did* understand, and officially clarified 'Evolutionary[17]' . The key distinction is that Evo *measures* iteration results, increments results, and *analyzes* the feedback, and *learns* and *changes* what it needs to change, exactly like IBM Cleanroom did, in the Seventies (Mills, Quinnan). Evo is not simply a 'delivery mechanism', it is an experimental, learning, and course-correction mechanism.

Evo is an 'engineering' approach, which requires a concept of 'quantification of critical values and qualities', and incremental measurement of those values, and redesign when necessary to achieve critical objectives. Good 'engineering: but unfortunately 'software engineers' were *not* engineers, so they simply ignored the 'quantifying quality' stuff, and focussed on delivering code incrementally (Scrum). They got rapid code delivery, but no control of qualities at all.

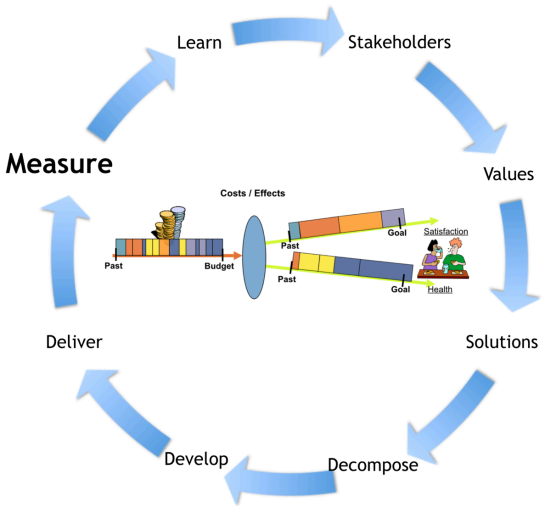
---

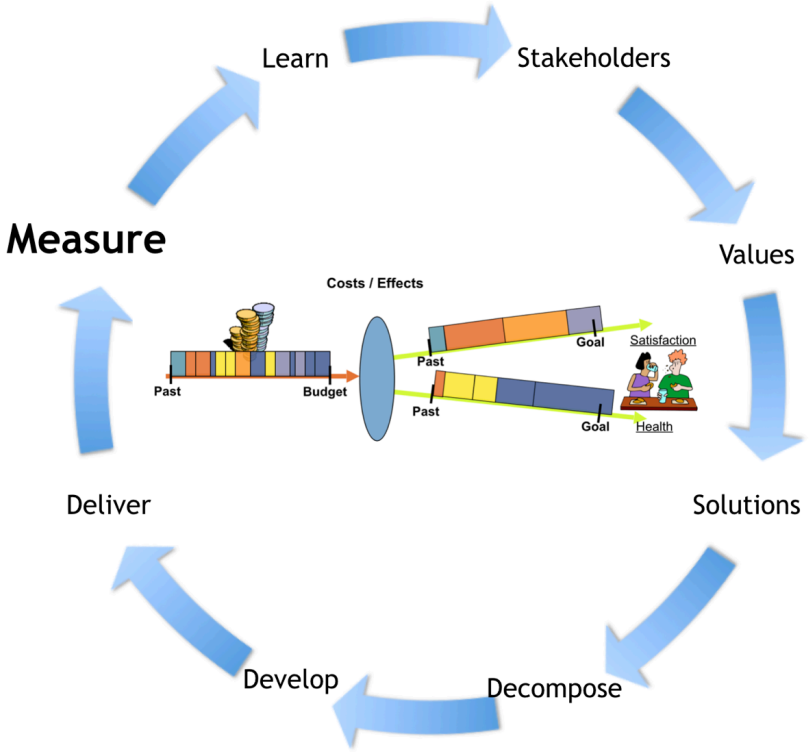
[15] https://eprints.soton.ac.uk/266606/1/xp2008camera_ready.pdf cites my 1985 Evo paper, See explicit mention in Software Metics book, 1976

[16] Gilb: Principles of Software Engineering Management, (1988). Chapter 15 Deeper Perspectives on Evolutionary Delivery www.gilb.com/dl561

[17] MIL-STD-498 5 December 1994 Very explicit about Evolutionary, learning requirements as it evolved and got feedback.

Part 5. Evo: a process for value delivery and learning





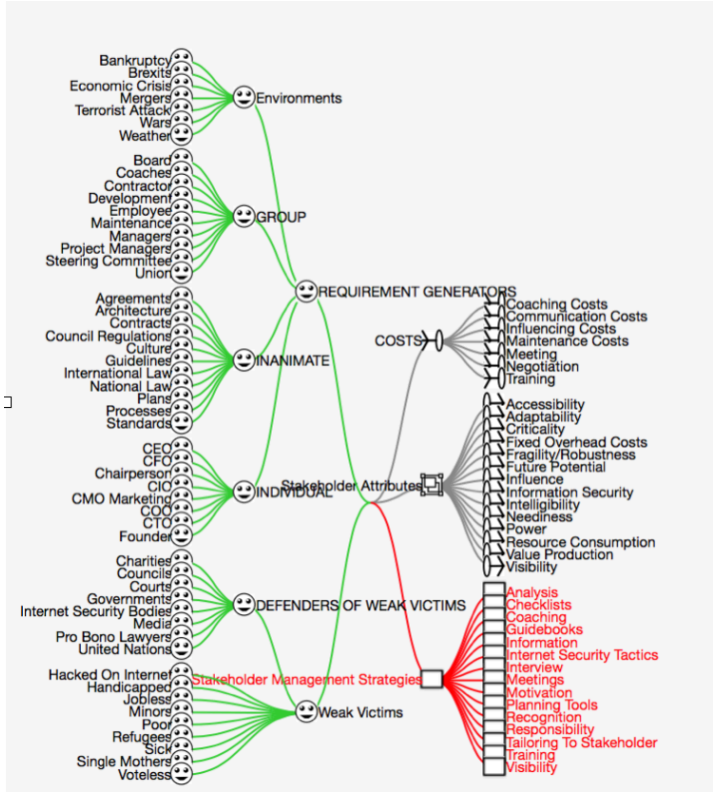
81. THE VALUE DELIVERY CYCLE:

SOURCE: KAI GILB DESIGNED THE CYCLE, AND THE CONTENT IN MIDDLE.

THIS IS A REUSE OF HIS GRAPHICS FROM MY MENSURA LECTURE, 2017, CONCEPTS.GILB.COM/DL918

81. Value Delivery Cycle: seeing the real world, and learning.

- Evo is based on Planguage, and its ability to quantify multiple values, qualities and costs simultaneously.
- Evo is also strongly based on the notion that there are very many stakeholders, with very many values, some of which are more critical than other stakeholders. Some values are more critical than *other* stakeholder values.
- The basic idea is that you can enter the cycle anywhere that strikes you as interesting. You could start at the Learning process, then analyze stakeholders (who just gave you a hard time) and continue around until things get measurably better. There is no fixed entry, and the cycle is rapid (a week, not a year).
- The cycle is an extension of the Deming/Shewhart Cycle (Plan Do Study Act, see Illustration 76). Our cycle is not as simple, but Evo is more explicit, about important sub-processes. Here is a comparison and summary.
  - **Plan**: Stakeholder Analysis, Stakeholder Value Analysis, Solution Analysis, Solution Decomposition
  - **Do**: Develop the system based on specified and estimated-values and costs solutions. Deliver solutions to a real world system.
  - **Study**: Measure the effects, and costs, of the solution delivery to the real world. Learn what is *really true* about solution effects.
  - **Act:** jump to any part of the cycle, that your learning tells you, needs to be improved (example, jump to Solutions (design process), make solutions better).



82. ON LEFT ARE EXAMPLES OF STAKEHOLDERS FOR A PARTICULAR PROJECT

ON RIGHT, COSTS OF LIMITED RESOURCES, MIDDLE RIGHT 'VALUES', BOTTOM RIGHT 'SOLUTIONS'

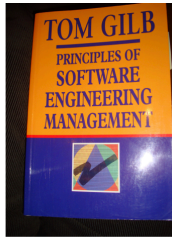
Source: Mensura Slides 2017, concepts.gilb.com/dl918

82. Value Delivery Process: the main point of all projects.

- It might seem obvious, that the main point of all projects, is to deliver *some* value to *some* stakeholders
- But it is surprising how many projects, and project management methods, do not deal with this fact very well
  - They do not quantify and specify the values and their levels
  - They can be very narrow in *stakeholder* analysis (just customer, client, user): not the other 50
- They focus on building a 'solution', and assume that the values will just happen automatically. It is not that easy!
- There are a large number of stakeholders, and each has at least one 'stake' (requirement) or several..
  - These need to identified, acknowledged, clarified, specified, quantified, and then, and only then, PRIORITIZED
  - You can't 'logically' prioritize what has **not** been: identified, acknowledged, clarified, specified, and quantified.
- There are *many* stakeholders, and *many* values, and *limited* resources:
  - every stakeholder, and every value, is simultaneously competing, for those limited resources.
  - That is why it is important to *prioritize*, to get the *most value* for the *critical* stakeholders, **early**, before resources are gone.

- This view of values, stakeholders, and design capabilities is itself a Technoscope.
- It gives us a method for understanding complex projects and systems.
- Crime TV says, 'Follow the money'. Our Technoscope says 'Follow the Values'
- The values are 'outside the black box' of systems complexity: they simplify understanding the rats nest inside.

Robert E. Quinnan: 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 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)

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

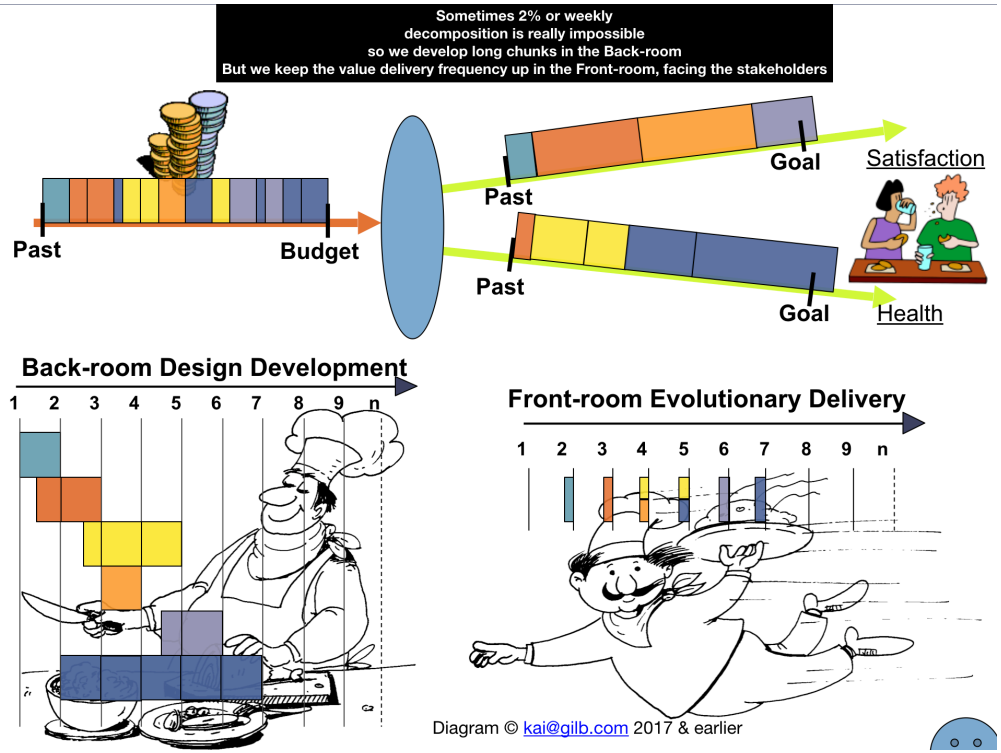
83. DYNAMIC DESIGN TO COST: QUINNAN CLEANROOM.

83. Dynamic Design To Cost (DTC): How to see what needs to be changed immediately, and if the change really works! : a Techno-scope.

- The engineering culture has a powerful paradigm, 'Design To Cost'
  - Don't ask what the cost will be…
  - Ask how to design it, so that the cost *will* be, what you can accept
- Dynamic (design to cost) is when you do not simply do Design To Cost, once, up front
  - It is when that DTC process is repeated, *every value delivery cycle*.
- This is very similar to how sailboat adjusts course, or most other non-trivial voyages and travel.
  - You have a destination (which can itself be adjusted, any time), a deadline for arrival, and you regularly and constantly ask
    1. Am I on course for the destination, on time?
    2. If not, 'what can I do to get back on course?'
    3. Try it out, see if it works. If not, go to step 2. Until you arrive, or give up.
- When this 'Evo'-like method was applied at IBM FSD, they reported all projects delivered on time, under budget, with highest contracted quality levels (Mills, IBM SJ 4/1980[18]). This is obviously a fantastic product development method.
- But few have heard of it, teach it, use it: we use inferior methods:
  - "***Those who cannot remember the past are condemned to repeat it"*** .(Santayana, 1905-6)

---

[18] Mills, H. 1980. The management of software engineering: part 1: principles of software engineering. IBM Systems Journal 19, issue 4 (Dec.):414-420. http://trace.tennessee.edu/cgi/viewcontent.cgi?article=1004&context=utk_harlan



84. THE 'BACK-ROOM' IS A PLACE WHERE WE UNOBTRUSIVELY GET SOLUTIONS READY FOR VALUE DELIVERY, WITHOUT BOTHERING OUR STAKEHOLDERS WITH THE DETAILS.
THIS IS A PURE 'INVESTMENT' PHASE.

THE FRONT-ROOM IS THE STAKEHOLDER-FACING VALUE-DELIVERY MECHANISM.

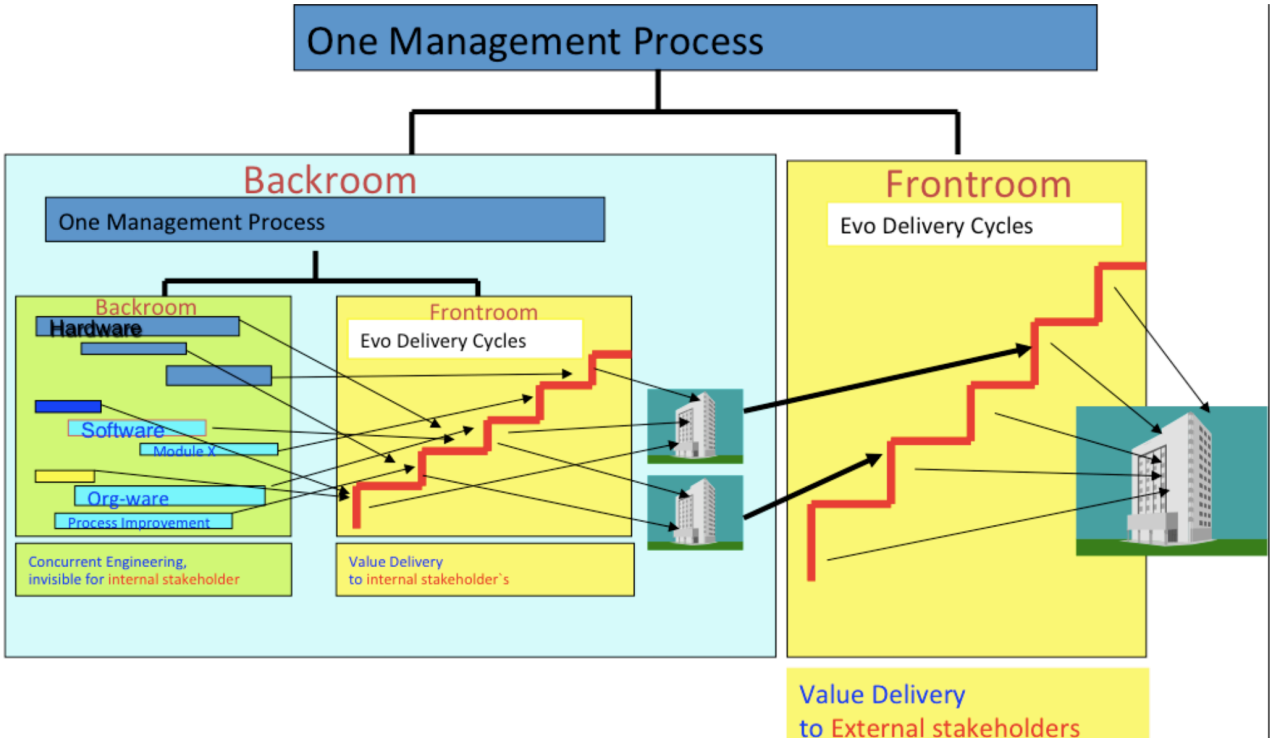
THIS IS THE 'PAYBACK' PHASE-

84. Backroom: a place to build systems with value, that take more than a cycle to build.

- As discussed (Decomposition, above) it is useful to decompose your larger solutions (larger than your cycle of delivery, like a week) into sub-solutions, which can be built and delivered, within your value-delivery cycle time.
- But sometimes that is not actually possible.
  - For example when a supplier cites a long delivery time, or when a solution just takes 'months', and there is no identified way to decompose that into value-delivering sub-solutions.
- We know that it is not that critical that sub-solutions can be 'built', in as short a time as our delivery cycle.
- We are however, very interested in having *something* delivered, immediately, and every short cycle thereafter.
  - Because: it keeps the value flowing to the stakeholders
  - It gives us feedback on how things work, and are received
  - It keeps the team focussed on the real world, early, and continuously.
- Our solution is that anything we cannot decompose easily, is 'started' (when we want to), in development, or a supplier ordering process, and placed in what we call the '**Backroom'**.
  - We simply wait until it is done (ready for value delivery implementation in the real world), however long that takes,
  - When it is 'done', i.e. ready for deployment to the real live system, we can put it in a queue of possible deliveries
  - And when it is prioritized, as the smartest thing to do now, we deliver it
    - It might be finished *before* it is the 'prioritized solution', for that system. But we *can wait* to deploy it.
- Of course solutions can be delivered in parallel, with the advantages and disadvantages that has.
- There is a nice practical analogy, the restaurant kitchen (even home kitchen) which prepares certain things in advance, say a dessert, and delivers it when the guests prioritize ordering it, and eating it.

This Technoscope helps you keep track of potential short-term value-delivery options, while it makes sure that the 'defeatist excuse' *'this solution cannot be decomposed into smaller deliveries'* is *not* a *valid* excuse, for trying to stop the early, frequent, flow of value delivery to stakeholders, from going continuously.

Source:Value Planning, Fig. 5.1



85. PHILIPS CORPORATION (IN HOLLAND) IS A CLIENT WHO USED THESE IDEAS TO ADVANTAGE, AND EVEN INVENTED THE 'FRONT AND BACK ROOM' INSIDE THE BACKROOM. IT IS A COMPLEX SUPPLY CHAIN STRUCTURE FOR 'INTERNAL STAKEHOLDERS' AND 'EXTERNAL STAKEHOLDERS'

85. Frontroom: where we expect to see and measure progress.

- The Frontroom is the ‘stakeholder-facing’ process, primarily focussed on implementing a solution, in the real world.
- This would consist of activities like these:
  - Prioritize and select, the next viable-to-integrate delivery step, based on value for money, and with regard to risks
  - Make estimates (improved by previous frontroom experience) for the value increment, the side effects, and the costs
  - Make, worst-case estimates (lowest value, highest costs)
  - Engage stakeholders: do they need to be in place, doing certain activities, willing to be trained, agree that they want the promised values, willing put up with some ‘teething troubles’, give feedback and give constructive suggestions.
  - Final quality control, and field testing of the new increment (before bothering the value-receiving stakeholders)
  - Designing a test and measurements process for the increment. How much QC effort is warranted?
  - Implement, turn on the lights, and measure, test, review, and iterate if necessary, or move on to next cycle.

Note, that this is a subset of project management, but there is much more that needs managing before we can do this.

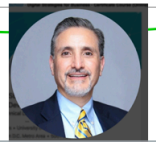
We might invent a Value Delivery Manager, for leading the effort.

This Frontroom Technoscope is a primary and simple way to understand any project. We focus on tracking the value delivery increments. Again, we are ‘outside the black box’ of the complexities of the project. We are focussed on the critical value objectives, and their progress.

How do Lean & Agile Intersect?

Agile Values	Lean Pillars	Lean Principles	Lean & Agile Practices	Flow Principles
Empowered Teams	Respect for People	Relationships	• Customer relationships, satisfaction, trust, and loyalty • Team authority, empowerment, and resources • Team identification, cohesion, and communication	Decentralization
Customer Collaboration		Customer Value	• Product vision, mission, needs, and capabilities • Product scope, constraints, and business value • Product objectives, specifications, and performance	Economic View
		Value Stream	• As is policies, processes, procedures, and instructions • To be business processes, flowcharts, and swim lanes • Initial workflow analysis, metrication, and optimization	WIP Constraints & Kanban
Iterative Delivery	Continuous Improvement	Continuous Flow	• Batch size, work in process, and artifact size constraints • Cadence, queue size, buffers, slack, and bottlenecks • Workflow, test, integration, and deployment automation	Control Cadence & Small Batches
Responding to Change		Customer Pull	• Roadmaps, releases, iterations, and product priorities • Epics, themes, feature sets, features, and user stories • Product demonstrations, feedback, and new backlogs	Fast Feedback
		Perfection	• Refactor, test driven design, and continuous integration • Standups, retrospectives, and process improvements • Organization, project, and process adaptability/flexibility	Manage Queues/ Exploit Variability

❶ ❷ ❸



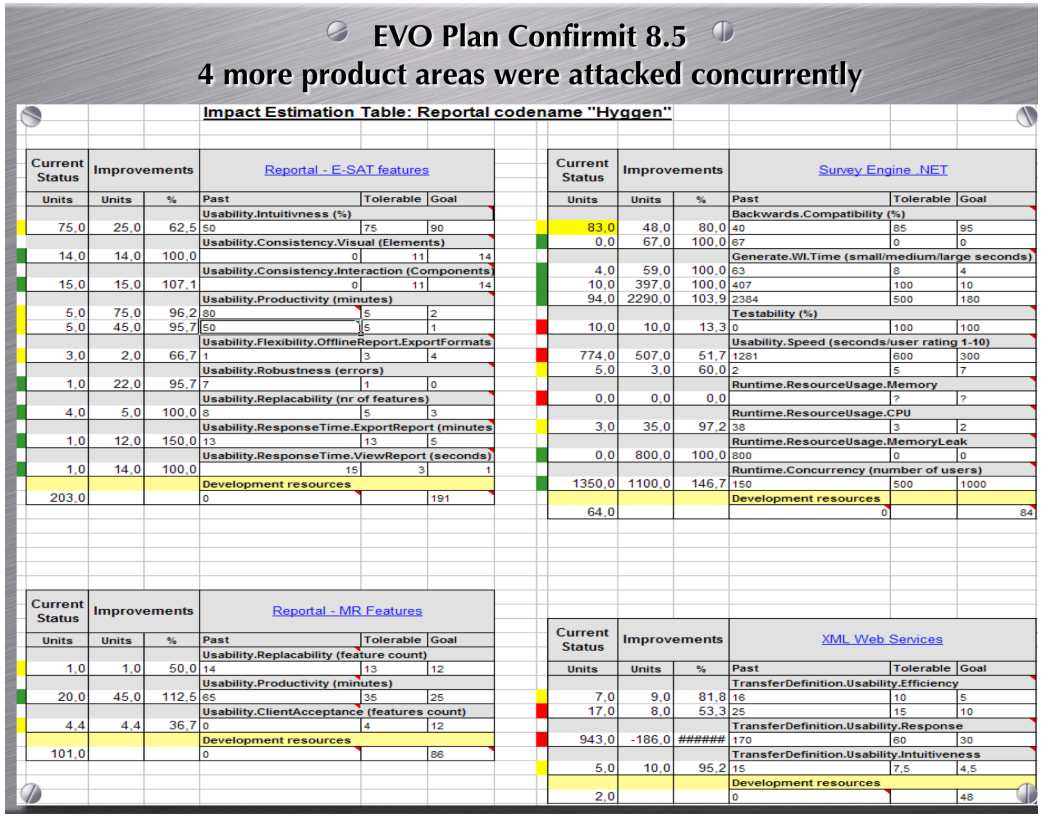
© Gilb.com

Source: David Rico

86. DAVIDFRICO.COM

86. Agility: Technoscopes for seeing that you need to ‘be agile’ and change.

- Being agile, refers to an organization’s ability to change course, when it is needed: quickly, and to a relevant course.
- Evo offers signals for agility: real measures after a delivery cycle with unusual results, for example.
  - Evo which looks for small **value** increments, 2%, weekly, makes sure that we have not spent too much time or money, before we get **relevant** (Value increment delivered, real costs) feedback, and can act quickly.
- Planguage offers signals for agility: change in stakeholders (different people, different stakeholders), change in Value Objectives, change in Resources.
- These signals can be tracked manually, but they can easily be automated warnings, if the plan is digital.
- What is necessary is that somebody (Plan Object Owner for example) is assigned to do something with the signals



EVO Plan Confirmit 8.5
4 more product areas were attacked concurrently

Impact Estimation Table: Reportal codename "Hyggen"

Current Status	Improvements		Reportal - E-SAT features		
Units	Units	%	Past	Tolerable	Goal
			Usability.Intuitivness (%)		
75,0	25,0	62,5	50	75	90
			Usability.Consistency.Visual (Elements)		
14,0	14,0	100,0	0	11	14
			Usability.Consistency.Interaction (Components)		
15,0	15,0	107,1	0	11	14
			Usability.Productivity (minutes)		
5,0	75,0	96,2	80	5	2
5,0	45,0	95,7	50	5	1
			Usability.Flexibility.OfflineReport.ExportFormats		
3,0	2,0	66,7	1	3	4
			Usability.Robustness (errors)		
1,0	22,0	95,7	7	1	0
			Usability.Replacability (nr of features)		
4,0	5,0	100,0	8	5	3
			Usability.ResponseTime.ExportReport (minutes		
1,0	12,0	150,0	13	13	5
			Usability.ResponseTime.ViewReport (seconds)		
1,0	14,0	100,0	15	3	1
			Development resources		
203,0			0		191

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	1281	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

Current Status	Improvements		Reportal - MR Features		
Units	Units	%	Past	Tolerable	Goal
			Usability.Replacability (feature count)		
1,0	1,0	50,0	14	13	12
			Usability.Productivity (minutes)		
20,0	45,0	112,5	65	35	25
			Usability.ClientAcceptance (features count)		
4,4	4,4	36,7	0	4	12
			Development resources		
101,0			0		86

Current Status	Improvements		XML Web Services		
Units	Units	%	Past	Tolerable	Goal
			TransferDefinition.Usability.Efficiency		
7,0	9,0	81,8	18	10	5
17,0	8,0	53,3	25	15	10
			TransferDefinition.Usability.Response		
943,0	-186,0	######	170	60	30
			TransferDefinition.Usability.Intuitiveness		
5,0	10,0	95,2	15	7,5	4,5
			Development resources		
2,0			0		48

FIGURE 87. NOTICE THE 'IMPROVEMENT % COLUMN. 100% MEANS WE DELIVERED THE GOAL LEVEL, DUE 12 WEEKS FROM START OF THE QUARTER. THIS IS 9 WEEKS INTO THE 12 WEEKS. HOW 'DONE' IS THIS PROJECT? 3 WEEKS AHEAD OF THE DEADLINE. NOTICE THE TRAFFIC LIGHT SIGNALS: RED = NOT EVEN TOLERABLE, GREEN = DONE

Source: Trond Johansen, Confirmit Public Presentation.

March 9-10 2019 Edit

87. Definition of Done: A Technoscope to know when you are really done, and when you are not.

- There is a widespread culture, that acts as though they are finished, when the ‘solutions are built’, or ‘implemented'.
- Maybe for some of them, the builders or suppliers, their job is then done. Even if a lot of stakeholders are unhappy.
- But the project *is there to deliver value*, **not** just solutions, that we HOPE will give value, but might NOT give value, and might even have unexpected negative side-effects. Who takes responsibility for Value Delivery?
- So we need to be quite clear about exactly what it means to ‘succeed’, to be truly done with the project assignment.
- Planguage, and the related processes (Evo, Spec QC, Value Decision-Making) have a number of specific devices to help us objectively decide: WE SUCCEEDED, WE ARE DONE. CONGRATULATIONS.

Here are the basics of being done:
1. All Goal levels, within deadline, to now, are measured delivered. GOOD
2. All Goals Delivered, and Stakeholders asking for more value, now BETTER
3. For suppliers: All contractual conditions met, and agreed met. OK
4. Critical stakeholders are clearly happy, and want more value from you. GOOD

Here are some more-advanced ideas of being done:
1. Degrees of done: % of Goal Levels delivered to date (see example, Fig. 87, at left)
2. All Goals delivered, all resources used, are less than budgets, with some movement towards, Stretch Goals

Mixture: Somewhat Done
1. Many Goals reached, some not, some are late
2. All goals reached, but unacceptably late or over budgets
3. All Tolerable or OK levels reached, but few or no Goals reached

Really Embarrassing Level (we are done, spending someone else’s money)
1. Lots of money and time spent, no planned measurable value delivered at all

88. Constant Dynamic Reprioritization

Do you want to go much deeper into prioritization? Try VP book, Chapter 6 Prioritization, 60 pages. https://www.dropbox.com/sh/34llx1a7ckyagxl/AAA0pDz-SxN5WmoP9lOKR0Mpca?dl=0

March 9-10 2019 Edit

88. Dynamic Re-prioritization: re-calculating that you have new priorities. The Priority Technoscope. Objective priority logic.

- By ‘priority’ , we mean, chosen to do first, or next.

This should not be a static, one time, subjective decision (What is your First Priority?)

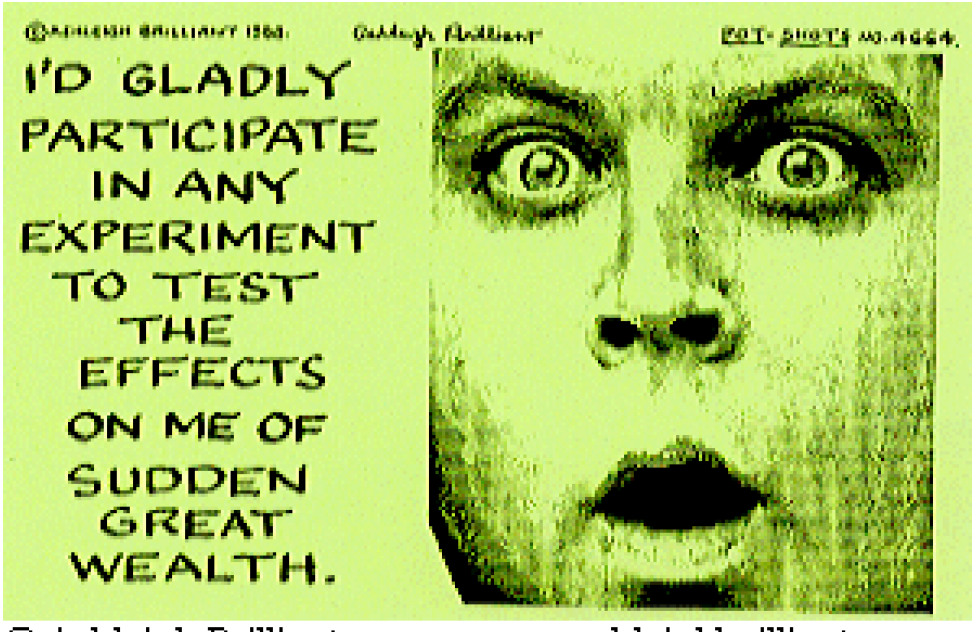
- For ‘serious projects’ (yours) you should use facts and logic to determine priorities, continuously. Agility..

- People, and groups, still need to have subjective opinions on certain aspects of determining priority:
  - How do you prioritize values, value objectives, and their levels.
  - How do your prioritize limited resources (time, people, money)
  - How do you prioritize various stakeholders, and their consequent satisfaction.
  - How do you prioritize risks?

- Once we have our general priority policy
  - EXAMPLE: **“stakeholder X, Y AND Z FIRST, Values A, B, and C must reach Goal, Budgets cannot be exceeded, Risk must be lowest and most conservative”**
  - We can use logic and computation to determine, which ‘Solutions’, will deliver Values A, B and C, at least cost, for Stakeholders X, Y and Z, without exceeding any budget, with least risk.
  - This is exactly what the Confirmit example does (Fig. 87) , to some degree, with a spreadsheet, and traffic Lights. (Red has highest priority)
  - The ValPlan.net (needsandmeans.com) tool does this even more automatically, as plans change, and feedback dictates.

If you are doing serious projects, you should upgrade your approach to prioritization:
- Conscious use of stakeholders, values, costs, and risks
- A transparent priority policy, agreed by responsible instances
- Automatic updating of current priority, based on changes to the above factors, and to other insights (‘there is no more time left, but plenty in the budget, so we have to buy our way to success’).

89. SCIENTIFIC EXPERIMENT



89. Gradual Integration (not all at once) Cause and Effect.

- ‘Agile’, implies that you develop things one small step at a time.

- As we have discussed previously, there are several inherent advantages to doing one step at a time:
  - The ability to get results quickly
  - The ability to prioritize critical and valuable things early
  - The ability to reduce risks, by not betting too much at a time.

There is another Technoscope aspect of step by step, I want to discuss here.

We can frame this as the **Scientific Experiment** aspect.

As you might well be aware, if you mix 10, or worse 50, food ingredients, with 5 food preparation processes, and it tastes horrible: you really are not sure *why* (except that you did ‘too much at once’). But I can tell you that a lot of real projects I have seem, do exactly that, if not with food. And they fail, totally, and frequently (just Google ‘Failed XXX Type Projects).

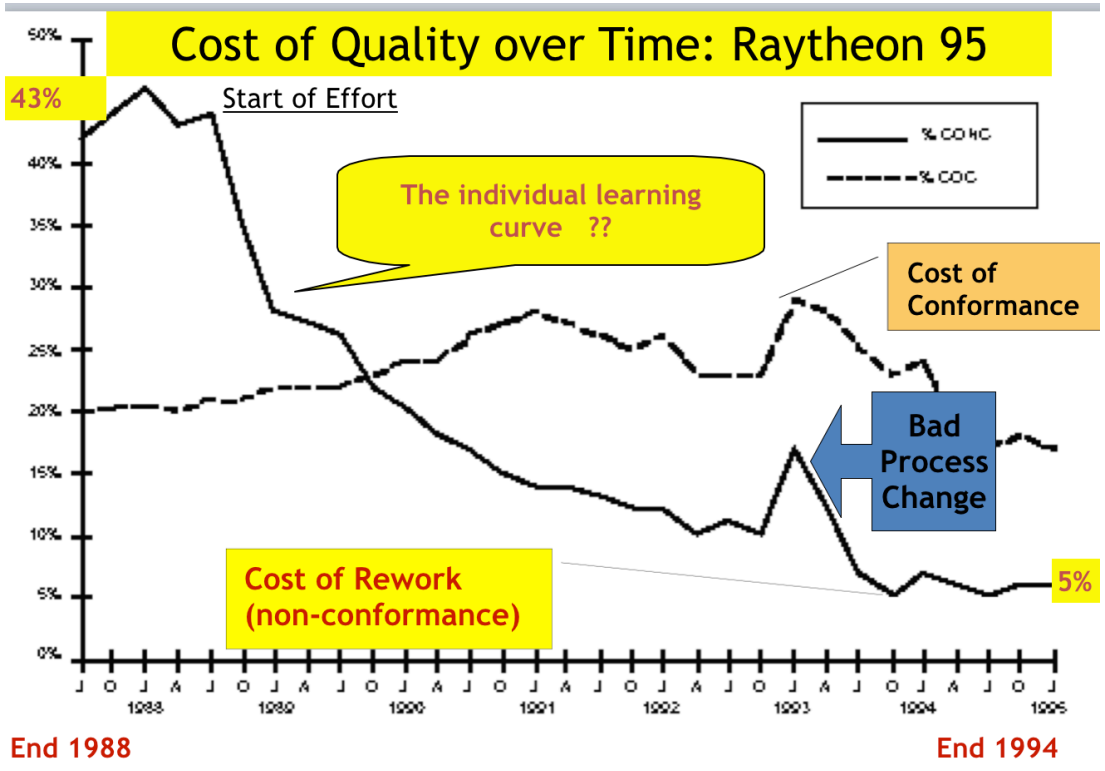
We have also seen, that for example (above) the Cleanroom projects, done in 2% steps, succeeded, every time. And they were doing state-of-the-art Space and Military projects. Lots of cutting edge untested ‘stuff’.

There is, as you probably know, a basic rule of understanding ‘causes of effects', like ‘food and its taste’. Do *one single thing at a time*, and *keep all other conditions the same*. Then if the taste goes sour, it was probably the last thing you added (if it was not due to a temperature change, or congealing, or effect of light ! — but keep *those* constant, if you can).

This aspect of an agile process, means you can afford to take a fairly high risk, with any one sub-solution. And with a bit of luck, most things work, and the sum of the risks that went well, is much better than if you were forced to be ultra-conservative with all components, because they ‘all had to work ok’ when you ‘put them all together at once’.

> ***“Shallow men believe in luck or in circumstance.***
> ***Strong men believe in cause and effect.”***

Ralph Waldo Emerson (1803-1882)



90. RAYTHEON USED THE DEFECT PREVENTION PROCESS, DELEGATING POWER TO ENGINEERS THEY REDUCED 43% REWORK BY 10X, AND DOUBLED PRODUCTIVITY

90. Delegation of Design Power: The Technoscope of seeing things with the real workers.

- I discussed in connection with **Defect Prevention Process**, above [79], how powerful it is to delegate the power of analysis and change, to the people most-directly ‘living with the problems’. Not Directors, or Consultants.
- See **75, Deming’s** 14 Points (some others quoted above, 76, 77) said a lot about the ‘worker on the shop floor’ empowerment, too.
- **Confirmit**, case above [49, 87], was very good at delegating analysis and design power to the Grass Roots teams of engineers, getting feedback from Evo cycles.
- There is a lot of experience and writings on this subject, but still far too little adoption as a widespread practice.
- There are still too many managers making bad decisions, advised by poorly-qualified overpaid consultants.
- In terms of the Evo method, I suggest we need to get really good trustworthy feedback from the real people affected on a daily basis, like ‘nurses and patients’ affected, not just the ‘hospital administrator'. This means we need to identify them (nurses and patients, for example) as **critical stakeholders**, and make sure, after a value delivery cycle, that we design tests, measurements, and observations, to really truly find out what is going on.
- Never trust the managers, or the suppliers. They do not know, and they will hide the truth.
  - Sorry, there might some exceptions, but I have seen a lot of companies and people, all over the world, for decades (and some managers are really good, too few). Trust the workers, to know where the shoe pinches.
  - Stakeholder and stakeholder value analysis are the key Technoscopes for seeing this level of a system.

***Elon Reeve Musk* - Chairman & Chief Executive Officer Tesla.**

"Okay, I think that's a pretty open-ended questions, but – we have a philosophy of just continuous improvements, so every week there are approximately 20 engineering changes made to the car.

So it's not nearly as discrete as you're alluding to. With other manufacturers, they tend to sort of bundle everything together in a model year.

In our case, it's a series of rolling changes. So model year doesn't mean as much. There are cases where that step change may be a little higher than normal as, for example, with having the Autopilot camera, radar, and ultrasonics.

But we try to actually keep those step changes as small as possible.

And so that – I mean, essentially like the common questions that I get is from friends, they say, "when should I buy a Model S?" and my answer's always "right now," because – and they say, "well, aren't you going to make a better one in six months?" I'm like, yeah, of course.

But if their goal is to only buy a Model S when there aren't significant improvements happening, then they will never buy one."

91. *Musk on eternal continuous Tesla improvement.20/week x 50 weeks = 1,000 improvements per year. Source: http://seekingalpha.com/article/3642146-tesla-motors-tsla-elon-reeve-musk-on-q3-2015-results-earnings-call-transcript?page=2&p=qanda&l=last*
November 3 2015.

91. 2% Delivery Steps: A Feedback Technoscope.

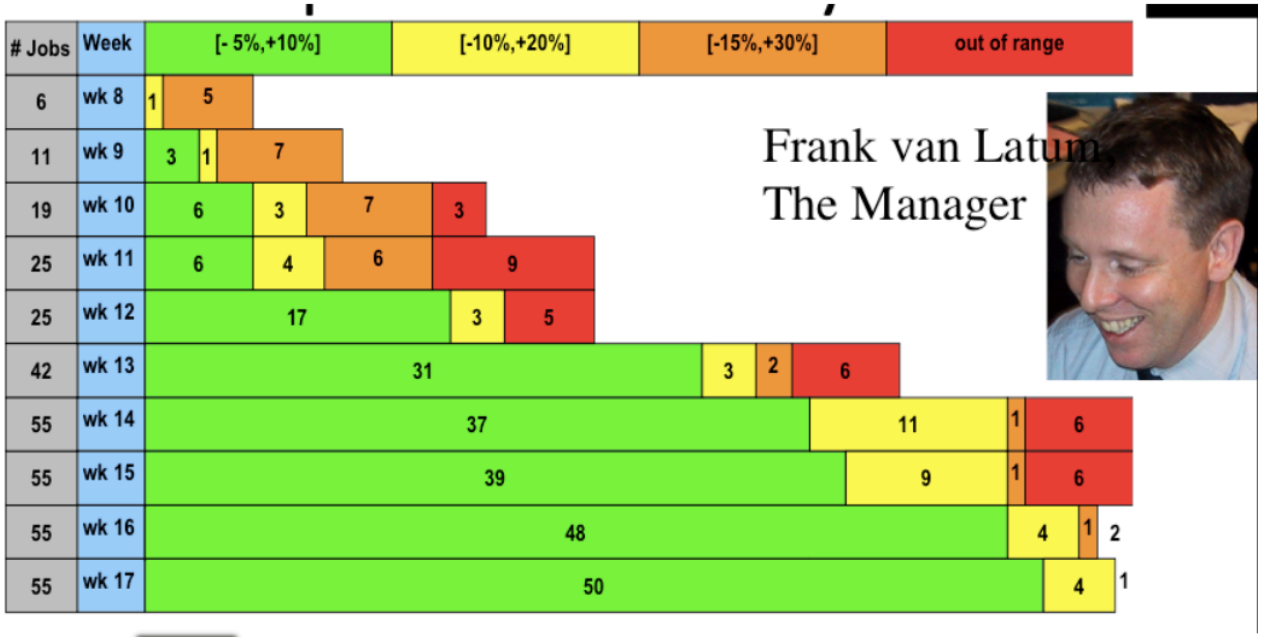
- Here is a quote from earlier: [35]
- **Suitable Size Effort**. The 'value delivery steps' as we call them, need to be small enough to do, in a short time (a week or so), but large enough to make a real difference (non-trivial result). This delivery cycle size is roughly 2% of the entire project effort, or 50 weekly increments in a year's project, or 48 monthly cycles in a 4 year project.
    - Some good reasons for the 2% size is that it is big enough to *make a difference in results*, and it is short enough to *keep people's attention focussed*, and *small enough to 'lose totally'* (2% of 100%) without destroying all credibility and resources.

Lets look at this from the Technoscope point of View:

- The 2% increment is not an isolated component build, it is *integrated* into the ***real*** **ongoing system** (product, app, organization, manufacturing system). That means that you can more **reliably** measure the effect it has, on the incremented quality, value, performance of the *entire* system.
- Since the 2% increment is '**the only change**' usually, we can, as discussed above (Cause and Effect) reliably infer that the incremental value improvement is due to *that particular* change.
- We can use Deming's **Statistical Process Control** (SPC), to make sure the system performance is stabilized, statistically: giving added assurance that the 2% increment is really the cause of the performance effects, we measure.
- We can do our best, to estimate the multiple effects, but especially the primary ones, of the 2% change, and then *compare the actual step measures with these predictions*: if they agree. Nice. If not, we can analyze, and seek root cause. We can learn either about the components design, the implementation method, or the measurement method.
    - Doing this 50 times, throughout a project, gives *experience* and *insights*: a Technoscope in *itself* (that gives 50x learning opportunities).
    - I have clients who do this learning and feedback process, continuously for years, after initial deployment.
    - And Tesla (see left quote [91]) is another excellent example of doing this 'forever'. I am driving my second[19] Tesla S, and experience the incremental quality improvement dramatically, and have for 5 years now.
- Some planners have great difficulty envisaging this 2% decomposition in practice, in their systems and environments.
    - But my experience is that the problem is never that 'it can't be done'. It is their lack of knowledge, imagination, motivation, persistence, and ambition that is the real problem. Tesla has proven that to the auto industry.

---

[19] I had to buy my *second* car to get, the **over 1,000** hardware increments since my first Tesla S was made. I also then got the corresponding new component software, like Auto Pilot. The car is near perfect!



# Jobs	Week	[- 5%,+10%]	[-10%,+20%]	[-15%,+30%]	out of range
6	wk 8	1	5		
11	wk 9	3	1	7	
19	wk 10	6	3	7	3
25	wk 11	6	4	6	9
25	wk 12	17	3	5	
42	wk 13	31	3	2	6
55	wk 14	37	11	1	6
55	wk 15	39	9	1	6
55	wk 16	48	4	1	2
55	wk 17	50	4		1

CASE 92. IN THIS CLIENT CASE, NO PROGRESS HAD BEEN MADE FOR MONTHS.
I SUGGESTED 'DECOMPOSITION' AND 'EVO'.
THE BIG BOSS DIRECTOR SAID 'IMPOSSIBLE'
BUT FRANK (PROJECT MGR) AND I (AS COACH) WORKED THE PROBLEM PATIENTLY
AND FINALLY, WEEKS LATER, GOT A BIG ROUND OF APPLAUSE FROM THE DIRECTOR AND STAFF
WHEN WE BEGAN TO PRODUCE VALUE IMPROVEMENTS EVERY WEEK
AS SHOWN BY THIS ILLUSTRATION
THE PRODUCT WAS A TOOL FOR PREDICTING RUN-TIME, FOR A ROBOT, THAT PUT CHIPS ON BOARDS
WE FOCUSSED ON 'INCREASING ACCURACY OF PREDICTION' OF FABRICATION RUN TIME (MAIN VALUE)

92. Decomposition for Evo: the method of decomposition gives a better Technoscope.

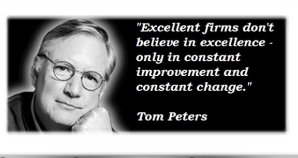
- Decomposing, discussed above, serves many useful purposes. In simple terms it gives us bite-sized real value deliverables, to a *real* system, which allows us to get **value early**, *continuously*, and to *learn fast* and *correct* certain bad designs.

- **Some Decomposition Tools: A Top level overview of decomposition technology.**
  - Value Decision Tables decompose by about 100 to 1 value opportunities. Look at the cells.
  - They decompose vertically by values and costs, then, on the horizontal axis, by strategies or solutions.
  - Value Decision Tables point out the best 1% opportunities for value, costs and risks. (10x10 = 1% cells)
  - The '111111 Decomposition Methods' [20] pushes you to think simple, 1 is better than 0. Do it. Get the value. Learn.
  - The 20 principles of decomposition[21] give more tools to trigger constructive decomposition ideas.
  - The Tolerable and the Goal levels of value give you 2 stages of decomposition to focus on. Go for the tolerable levels first. Decompose solutions to satisfy the higher priority (Tolerable) alone first.

---

[20] 111111 Unity Method of Decomposition into weekly increments of value delivery". (10 min. talk slides) http://www.gilb.com/DL451

[21] Decomposition of Projects - How to design small incremental result steps, 2008 Paper www.gilb.com/dl41



"There is no situation - even at Boeing - where you cannot concoct a sorta-real-world-micro-test of some piece of your project .. Within a few hours to two or three days.

1. Now. Right now. Take some little - tiny! - element of your project. Corral a surrogate customer. Talk to him/her about it. That is ... test it. Now.

2. Your immediate goal: "Chunk up" the next three weeks. I.e.: Define a set of practical micro-bits ... that can be subjected to real-world tests."

Quick Prototyping Excellence = Project Implementation Excellence. (No kidding... it's almost that basic!)"

Tom Peters, ***Reinventing Work, the project 50. Pages 138-9 (Quote 93A). He is right: I did the decomposition on 25 Boeing aircraft projects with 25 teams in a week each.***

"I think it's very important to have a feedback loop, where you're constantly thinking about what you've done and how you could be doing it better. I think that's the single best piece of advice: constantly think about how you could be doing things better and questioning yourself."
Elon Musk, Co-Founder/CEO Paypal, Tesla Motors, Inc., Space X

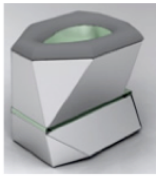
Quote 93B

Photo From a TED Talk Video

93. Spreadsheet Tracking Feedback: project progress Techoscope.

- Refer back to Figure 87, the project tracking spreadsheet, on week 9 of 12 week cycle before Customer Release Quarterly.
- This was presented there (Technoscope [87]) as a device for understanding the degree to which we were ‘done’.
- But at the same time it does other things too:
  - It gives a red flag, when any one of the values is still under the minimum Tolerable level
    - This give a clear priority to the project manager or the self-managing team in this case, to focus immediate future steps (there are only 3 of the 12 steps left before release) on reaching at least the tolerable level (hopefully the Goal too)
    - Then it computes the need for a Yellow Flag, for each and every Value which has reached the Tolerable Level, but not yet reached the Goal Level success level). This is the team’s second priority in the remaining 3 weeks.
    - It also computes the Green Flag, meaning that the Goal level is reached, and that no priority, no team effort, should be spent in the next 3 weeks, on improving beyond that Goal level. It would steal resources from the other 2 priorities.
  - It is not done in this Case, but we could easily compute the average % improvement for each team. I often find myself doing this mentally. We know we have spent 75% (9 of 12 weeks) of time to delivery deadline, so if the average % delivery was 60%, 75%, 85%, what would that indicate to someone trying to get feedback on the project?
    - If you are less than 75%, you are ‘behind the curve’:
    - If you were a lot over the 75%, you probably have enough resources to crack the remaining value gaps (distance to 100%)
  - Similarly we could compute things like ‘% of all Values not reaching Tolerable’, and ‘Not reaching Goal’.
  - There is some ‘engineering effort tracking' here. It is not well exploited, but we could relate actual effort to the degree of goals reached. Perhaps with conclusions like 'a team is understaffed’ (perhaps due to sickness, or wrong priorities)

An Energy Producing Waterless Toilet System



Impact Estimation Table for Gates GCE Project

Key Values	Detailed risk assessment with associated impact estimation table for methods of	Research trip to madagascar (x3)	Detailed design research	Building financial models at community level	Research into existing sanitation projects	Creation of knowledge 'database'	Codification of our acquired knowledge	etc....	Total Impact	Safety Factor
	Impact (% progress towards target from given action)									
Improve Sanitation Target: 25% - 75% Unit: Waste collected / waste produced by user group	10	20	40	18	15	0	0		103	1.03
Sustainability and Longevity Target: 0$ - 0$ Unit: Cost to single user per month	0	5	20	50	10	0	0		85	0.85
Story and Data Target: 0.4 - 0.8 Unit: Average of factors rated 0.0 – 1.0	5	35	20	15	3	15	5		98	0.98
Managing Risk Target: 0.2 – 0.8 Unit: Average of factors rated 0.0 – 1.0	50	20	20	15	15	0	3		123	1.23
Methodology Target: 0.4 – 0.8 Unit: Average of factors rated 0.0 – 1.0	15	0	0	0	0	0	10		25	0.25
Diffusing Knowledge Target 0.15 – 0.8 Unit: Average of factors rated 0.0 – 1.0	0	8	0	0	10	50	15		83	0.83
Total impact of design / action	80	88	100	98	53	65	33	0		
Total cost of design / action (person days)	8	30	20	15	5	15	4	0		
Benefit to cost ratio	10	2.9	5.0	6.5	10.6	4.3	8.3	####		

CASE 94. THE LOOWATT IMPACT ESTIMATION TABLE, SUMMARIZING THE FIRST 3 STEPS OF A PROJECT START-UP WEEK: OBJECTIVES, STRATEGIES, AND ESTIMATION OF HOW WELL THE STRATEGIES WILL HELP MEET THE OBJECTIVES. SEE CURRENT ACTIVITY LOOWATT.COM.

94. The (Agile) Project Startup Week: forcing the pace, to focus priorities, avoid bureaucracy, and become agile *value* deliverers.

- For at least 3 decades we have used a project organization process, I call the (Agile) **Project Startup Week**[22].
- In a time-constrained week (no overruns or excuses permitted) we do the following steps: each in max. 1 day
  1. Define the top-10 critical **values**, quantitatively (Scale, Goal etc) (can be summarized on a page)
  2. Identify the Maximum top-10 **strategies** for reaching those top objectives. (1 page output)
  3. Estimate how effective, and costly the strategies will be, for those Goal levels, VDTable (1 page output)
  4. Identify a first **real delivery-step for next week**. (a 1-liner, but several options are usually identified)
  5. Present plan to management, and get OK to **just do it** next week.
- I act as coach, for small teams. They have no previous training in Planguage or Evo. (it is practical in-house training')
- It works every time, and we always get the OK from a manager, who knows they are going to pay the team next week, in any case. But now the manager will get to see if they can accomplish **any measurable value result** (that the manager can brag about to THEIR manager) next week.

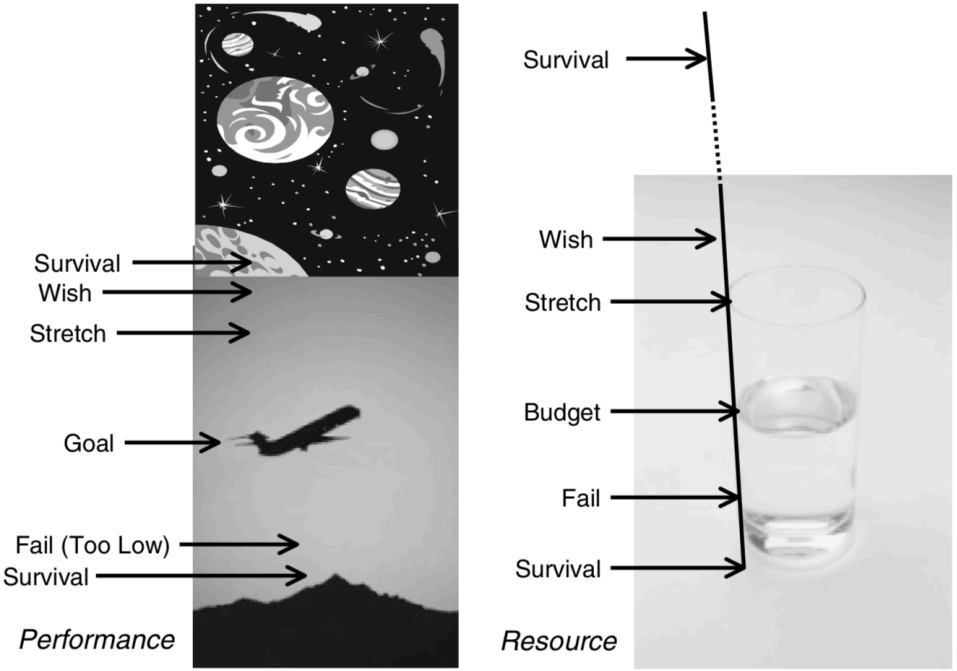
. In 1990, at McDonnell Douglas Aircraft (now Boeing) I did 25 aircraft projects, 5 parallel teams, in 5 weeks that way. They paid me a princely sum for the rights to the method, and for approving 2 teachers

- In 2007 we were using it on large multinational bank IT projects, which had been stuck for months in 'meetings'. We got them unstuck (not that the international political infighting did not try to get stuck again later!)
- In 2011 Kai Gilb and I volunteered for fun to help a startup, from Imperial College UK, working on advanced toilets.
  - We did the first 3 steps in a single day. (see Case 94 at left)
  - Bill and Melinda Gates promised the best startup $1 million. For Field Trail success in Madagascar (Health in Africa)
  - LooWatt, as their Startup Advisor Nicholas Coutts predicted, won the $1 million, and a handful of other Best Startup prizes. We *like* to think it had something to do with our method; but of course *they* are *great* learners and doers!

---

[22] 'An Agile Project Startup Week': Papers and slides , Talk slides pdf from ACCU Conference, Bristol UK, April 9 2014
90 minutes talk. Includes Startup Planning for Business Startups, Confirmit, US DoD case, 2 Bank cases, Detailed Startup week outlines and links to sources.
http://www.gilb.com/dl812

- The Loowatt Team told all other 20 Imperial Startups, they should also use this method
- Because it gave Loowatt the ability to ‘see the bigger picture’, ‘focus on the ultimate results, not the toilet.
- *THAT* IS A GREAT **TECHNOSCOPE** FOR A STARTUP !



95. A SET OF SOME OF THE PARAMETERS WHICH CAN DESCRIBE CONDITIONS ON A PERFORMANCE OR RESOURCE SCALE

Source: Competitive Engineering book, 4.6

95. Parameters in Planguage: well-defined and well-conceived specification parameters are also a useful set of Technoscopes.

- Planguage has roughly 685 well-defined Concept-Glossary Concepts[23], of these maybe 100 are Planguage Parameters, or used as subsidary specification options for the parameters.
  - The parameters are things like Type, Ambition, Scale, Meter, Past, Tolerable, Goal, and Stakeholder.
  - Then there are sub-parameters like for Type: 'Requirement Specification', 'Architecture Spec'.
- The simple fact that all of these concepts have a *published* (CE book is best), *stable*, *matured* and *deeply thought out* **definition**, which is proven out in many industrial organizations for decades, means you can trust them.
  - They are not 'invented on the fly'. They are not academic balloons. They are not compromise 'standards'.
- The Concepts 'know about each other', and are even defined *using* each other. They are *integrated*.
- They are *powerfully deep*: *for example:*
  - When a **Wish** Level (a Stakeholder *desire* or *need*, without respect to practicality or costs), is converted into a project commitment level (a **Goal**), the Wish Level *must* pass about 8 logical tests, which are good *common sense*, but are formally stated, in the Glossary under Goal (feasible, affordable, prioritized, etc.). Take a look. Or see Technoscopes 16 and 17 above)
  - You will find it difficult to find a Planguage level of *logical depth*, in any other 'same *systems* domain' methods of specification. Try.
- Some people would prefer '**simple**' (me too !), *fewer* Parameters and concepts. And it *can* be very simple!
  - Methods should be 'be as simple as possible, but no simpler'[24], avoiding failure. Most methods are 'too simple for purpose'.
- One thing is that anybody can choose a subset, at any time, for any purpose, and ignore the other Planguage details.
  - My clients do exactly that. Choose a subset suitable for them, and expand later, add own stuff, if they decide they need it.
- I do the same thing in my papers, talks, practical consultancy, apps, templates, and booklets like this. Simple subsets.
- But, I have to *somewhere*, expose the *complete* picture of all the 'little Technoscopes', so that people have a choice
- I can promise that no Planguage parameter is invented 'just for the fun of it'. They all have a *use* and a *purpose*: maybe not just now, for just you, or for *this* phase of your project.
  - **Don't worry: Planguage Technoscopes will all be there, patiently waiting, if and when you need them!**

---

[23] Full Planguage Concept Glossary http://www.gilb.com/dl830 , http://www.gilb.com/DL387 , http://www.gilb.com/DL46

[24] No, Albert did NOT, say that. But I did in Principles of Software Engineering Management book, 1988, p17. http://quoteinvestigator.com/2011/05/13/einstein-simple/

96. I WAS SO UNCERTAIN AS TO WHICH POTSHOT I SHOULD SHARE WITH YOU, THAT I WILL LEAVE THE DECISION TO YOU

March 9-10 2019 Edit

96. Uncertainty: <fuzzy brackets>, and other expressions of imperfection.

Confucius: "True wisdom is knowing what you don't know"

- It should be clear when things are unclear !
- We presented the '± Uncertainty of Estimate' Technoscope [44] earlier. Now let me enhance the idea a bit.
- Some people, when they are uncertain, are afraid of being *seen* to be uncertain, so they ignore it, and push on.
- I believe we need to encourage a culture, such as we find in some academic cultures, of bringing out the uncertainty explicitly.
- It should be considered unethical to 'know that something is uncertain', and to *not document* that fact, in some way.
- It should be considered threatening, to your team, that you know or suspect problems, but you do not help *them* to know or suspect.
- **'I *know* I don't know'** should be considered a sign of wisdom, knowledge, ethics, and professionalism.
- The 'Policy' of the organization, and the ethics of the professional should say:
  - **As a planner (designer, engineer, tester, professional) I will explicitly document all the uncertain stuff, so that my team, and my stakeholders, will be less likely, to be hurt, in any way.**
- Planguage has a great many simple, well-defined ways to help you express uncertainty. They cannot all be listed here. In fact, at the extreme, you could say that almost every detail of Planguage, is a Technoscope for helping you *deal with* uncertainty, to *perceive* that things are uncertain, and finally to *explicitly document,* or specify, a *potential* uncertainty.

The simplest and most popular device is the **<fuzzy brackets>**
This can be used anywhere, surrounding one or more terms. It announces that:
'BEWARE', this is FUZZY. That means you cannot trust it. Use it with caution. Or resolve the uncertainty now. I wrote the <fuzzy> brackets as a courtesy to you the reader, because I did not have time or the ability, to specify something with certainty. I did write the best I knew, in the time given. But I know that if you or I, have more time and insight, we can potentially specify something better, or at least remove the <fuzzy brackets>, indicating that it is safe, and can be trusted. We recognize that every <fuzzy statement> is a risk to all of us using the specification. So when we have time, work capacity, priority, or new sources of analysis and correction, we will get back to it, and fix it up. In the meantime, be very careful, and if you do make use of this <fuzzy spec>, perhaps you will pass on the warning that you did use it to others. Don't let the virus spread, unchecked, without warning. Be as helpful to others, as I am trying to be to you !

Bliss, Charles: Semantography

- http://www.symbols.net/blissymbolics/bhome/blissbio.html

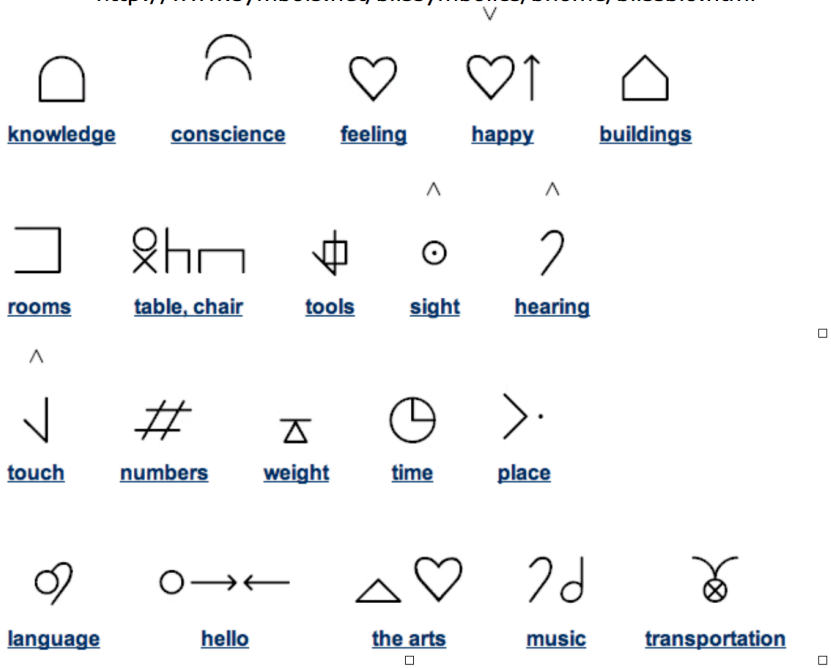


Illustration: Sample of Blissymbolics. They cannot be keyed from conventional keyboard characters.

97. SYMBOLS WITHOUT FRONTIERS

97. Icons: Plicons[25], Technoscopes for Humanity.

- Decades ago I saw a TV documentary about Charles Bliss, and Blissymbolics, Semantography. I corresponded with him.
- He developed a symbolic language capable of doing science, engineering and arts. It was modelled on Chinese writing, which stands above the various spoken languages. But it was designed to be so simple, a child could learn it.
  - And indeed one early successful application was helping orally disadvantaged children to communicate better with parents and teachers.
- I love the idea: a writing notation that communicates the *same idea* to all people, irrespective of their own native human language.
- That is sort of like music notation (not to mention music itself!), and electrical notation, and ballet! (and there *is* ballet notation)
- So, in parallel with my development of Planguage, I tried developing such a symbolic notation for systems engineering.
- We already use some few Plicons in our apps, and of course they are designed to be easily intelligible to computers.
- I decided there were two, related but different, types of icons: **drawn icons**, and **keyed icons**: the point being the keyed icons could be typed, on a keyboard, so special keyboards, or special graphical capability was not necessary. And it was easy to mix text and Plicons.
- I also found that when teaching Planguage, it was useful to use some icons, on a white-board or slide, to get concepts across visually, like Scales of Measure (——->) , design (Rectangle icon, or [Design X]). See example below.

[Design D] **O—<———>———> R** Meaning, a Design 'D' impacts a value requirement 'A', reaching the Goal (—->—)

Another example:

X -> Y [Oslo] 60% ? <- Tom

Means
'a Design tagged X, impacts a requirement tagged Y, in the city of Oslo , possibly, by '60% of the way to the Goal', according to Tom'

I have made attempts to define Planguage (or at least specific concepts) using a maximum of Plicons. It seems like a potentially new systems engineering notation. But it needs work, I need help, and I consider the larger body of the Plicons work, a personal experiment. I'm sure music notation began that way too, and Plicons might take a very long time to mature.

---

[25] Plicons: A Graphic Planning Language for Systems Engineering (Paper) http://www.gilb.com/DL37

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	

98. SOTO SPECIFICATION
(FROM CONTRACT TEMPLATE)
SHORT-TERM STATEMENTS OF TARGET OUTCOMES

98. Source: www.flexiblecontracting.com Benefield, Atkins[26]

[26] We do not have documented case studies of doing this format yet, so do report them to me if you do a variant. I have however made such contracts for decades, as a consultant.

98. Contracting As You Go[27]: I'm from Missouri, Ya gotta show me!

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) are *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 derived. 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.
- This contracting method, 'pay for results only', is an Evo-based and Planguage-based method, and it is a Technoscope because it helps suppliers understand, what they will get paid for, It helps you see what you expect to pay for of value and results. It reduces the risk of paying 'a lot for nothing'.

---

[27] Agile Contracting Column 8 2013 http://www.gilb.com/dl581 www.flexiblecontracts.com

99. SOME PRINCIPLES OF USEFUL KNOWLEDGE. FOOTNOTE [28]

UNIVERSALITY: 1. Knowledge is more useful when it applies to more circumstances

ETERNALITY: 2. Knowledge is more worth learning, if it can be applied for a long time, after learning it

VALUE: 3. Knowledge is more useful, if there is a high value from applying it

SHARING: 4. Knowledge is more useful, if it can easily be shared with others

PROOF: 5. Knowledge is useful, when early feedback, can prove its usefulness in practice

SYNCHRONOUS: 6. Knowledge is more useful, if it can be used together with a larger body of knowledge

MEASURABIILITY: 7. Knowledge is more useful, when the results of its application can be *measured*

ACCEPTANCE: 8. Knowledge is more useful, when it is widely accepted in your culture.

COST: 9. Knowledge is more useful, when the cost of applying it is low.

GENERATION: 10. Knowledge is more useful, when is can be used to generate even more useful knowledge.

28 Gilb, 'Some Principles of Useful Knowledge's (slides) INCOSE/KSSE 2015, Kongsberg http://concepts.gilb.com/dl844

99. Principles of Evo: principles are Technoscopes too.

- I *love* principles[29], powerful guides to smart action. Condensed wisdom.
- Principles can guide you in many ways, but let's focus here, on the idea that principles 'act asTechnoscopes': they can help you see, and analyze, complex situations.

For example:

Principle 1.1 Quantify Objectives [25]

All critical performance objectives *can* be quantified, and must be.

- The moment you take this principle seriously, and most people do **not** yet take it seriously or practice it, you move from:
  'Seriously great world-class security' (fluff) to
- **Security: Scale: % hackers caught within 5 seconds. Record 90%, Goal 99.9%.**

- This is a world of difference, from 'guaranteed to be misunderstood, and fail'; over to 'Fair Chance of Catching a lot more hackers'. (some level of success).
- Technoscope Principles can directly, immediately and clearly, help us understand, the complex systems and plans we work with.
- Study more principles, they are at least 'short'. [4. PPP, and CE, and PoSEM book 1988]

---

[29] https://www.gilb.com/store/4vRbzX6X 100 Practical Planning Principles booklet 2018

100. COMPETITIVE ENGINEERING BOOK 10.6, PRINCIPLES: EVOLUTIONARY PROJECT MANAGEMENT

1. The Principle of 'Capablanca's next move'
   There is only one move that really counts, the next one.
2. The Principle of 'Do the juicy bits first'
   Do whatever gives the biggest gains. Don't let the other stuff distract you!
3. The Principle of 'Better the devil you know'
   Successful visionaries start from where they are, what they have and what their customers have.
4. The Principle of 'You eat an elephant one bite at a time' .
   System stakeholders need to digest new systems in small increments.
5. The Principle of 'Cause and Effect'
   If you change in small stages, the causes of effects are clearer and easier to correct.
6. The Principle of 'The early bird catches the worm'
   Your customers will be happier with an early long-term stream of their priority improvements, than years of promises, culminating in late disaster.
7. The Principle of 'Strike early, while the iron is still hot'
   Install small steps quickly with people who are most interested and motivated.
8. The Principle of 'A bird in the hand is worth two in the bush'
   Your next step should give the best result you can get now.
9. The Principle of 'No plan survives first contact with the enemy'
   A little practical experience beats a lot of committee meetings.
10. The Principle of 'Adaptive Architecture'
    Since you cannot be sure where or when you are going, your first priority is to equip yourself to go almost anywhere, anytime.

100. Attributes of Evo: the qualities and costs of the ‘Evo’ value Delivery Process.

- Methods are ‘good for you’ (like Guinness) if their ‘attributes’ are 'better matched’ to your objectives and resources, than other methods.
- No costs to learn Evo, or do Evo, or ‘get certified’ as a Master.: Evo is ‘free’, like geometry and philosophy.
- Totally focussed on **delivering stakeholder values quantitatively**
- Measurable **values** should flow to stakeholders, step-by-step (not just code and features, and functions)
- Evo can focus on a number of **long range and short range costs** and resources.
- Evo works on **any class of system**
- Evo is based on a **learning cycle**, and being agile.
- Evo is a **Technoscope** to see Values, Stakeholders, and Costs - better.

Last page

Did you really read this far? Or did you read a lot, and checked in at the end.

I'd be happy to hear from you, about what you think of the book.

If you have any ideas for improvement,

and if you have any suggestions for this I should read,

perhaps something you wrote?

tom@Gilb.com

September 2018

Editing notes

260119 Pawel Nowak edit for printing etc.

270119 Tom Added BOOK REFERENCES

29310119 spelling corrections. , clarifications

References

1. [IC] Tom Gilb: **Innovative Creativity**. 2018.
https://www.gilb.com/store/QMMQhn2g

2. [CC] Tom Gilb: **Clear Communication**. 2018. https://www.gilb.com/store/oJCCxtsM

3. [100T] Tom Gilb: **Technoscopes: Power Tools to Master Complex Plans and Problems**. 2018. https://www.gilb.com/store/Pd4tqL8s **THIS BOOK**

4. [PPP] Tom Gilb: **100 Practical Planning Principles.** 2018 https://www.gilb.com/store/4vRbzX6X

5. [LD] Tom Gilb: **Life Design**, 2018. https://www.gilb.com/store/kCBGcG6L

6. Tom Gilb: **Vision Engineering**,
(this is also a subset of the VP book above)
Value Planning: Top Level Vision Engineering
CONCEPTS.GILB.COM/DL926

(pdf format). A 63 Page book. Aimed at demonstrating with examples how top management can communicate their ‘visions’ far more clearly.

There is a German and Russian Edition of this, at beginning of full Value Planning book.
leanpub.com/valueplanningdeutsch
leanpub.com/valueplanningrussian

7. [VPB] Tom Gilb: Δ : **"Value Planning Basics"**
For Advanced Management Results .
A very short text main book’s (VP book’s 800 pages condensed into 10 text pages)
Read in an evening or on the plane. Free at the moment.

A 23-page book. (half are illustrations). Contains the 1 page book summary.

5A: PDF version
https://www.dropbox.com/sh/yjqd50rzipoxcan/AAA2vWWwHZg7M4557vysjtsya?dl=0

5B. ePub version
https://www.dropbox.com/sh/rwykqi05uux4f0u/AAADzymMhjrF6cV_NWrl18uOa?dl=0

8. [TT] Or maybe you prefer the 18 minute Tom Gilb TEDx video? On **Quantifying Love**.
https://www.youtube.com/watch?v=kOfK6rSLVTA

9. [CE] Gilb, **Competitive Engineering**, 2005. https://w.gilb.com/p/competitive-engineering
Paper copies available via internet, bookstores. $41.27, 480 pages. www.elsevier.com and others

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

https://www.amazon.com/Competitive-Engineering-Handbook-Requirements-Planguage/dp/0750665076 (paper copy)

10. [VP] Value Planning, 2014-2019

Value Planning (digital book manuscript), 893 pages.

Get 50% discount on Value Planning

Use this link: https://goo.gl/MB6kaR

Coupon Code: CONNECT

One reader paid €20 to print the book out.

11. [SI] Gilb & Graham, Software inspection, 1993

https://www.amazon.com/Software-Inspection-Tom-Gilb/dp/0201631814

Table of Contents