LEAN VALUE PLANNING

By tom@Gilb.com

‘Lean’ is a smart concept. It is a philosophy of doing things in an efficient way. ‘Avoid waste’ is one way to summarize it. Lean is a set of ideas, working together to avoid wasted resources.

The exact set of ideas that is called ‘lean’ is not standard, or official. Anyone can ‘make up’ a useful set of ‘Lean’ ideas, as long as they arguably contribute to efficient value delivery. In fact, one useful lean idea is to keep your set of ‘lean’ tools up-to-date, and make sure they really are continuously effective in reducing costs, and delivering value. Keep ‘lean’ lean.There is no need to have a fixed or constant set of Lean Ideas, just because they were defined in a paper, talk or book by some lean ‘authority’. You have to take responsibility for being your own best local ‘lean expert.

Everything not adding value to the Stakeholder is considered to be waste.

This includes:
1. Unnecessary code and functionality
2. Delay in the (software) development process
3. Unclear requirements
4. Bureaucracy
5. Slow internal communication

And ‘lean’ means to:
6. Amplify Learning
7. The learning process is sped up by usage of short iteration cycles – each one coupled with refactoring and integration testing. Increasing feedback via short feedback sessions with stakeholders helps, when determining the current results of development, and adjusting efforts for future improvements.
8. Decide as late as possible
9. Deliver as fast as possible
10. Empower the team
11. Build integrity in
12. Separate components work well together ,as a whole, with balance between flexibility, maintainability, efficiency, and responsiveness.
13. See the whole system, or ‘picture’
14. “Think big, act small, fail fast; learn rapidly”

http://en.wikipedia.org/wiki/Lean_software_development

Figure 1: A Software development view of ‘Lean’. It is general enough to apply to most forms of development, and planning, outside of ‘software’ and ‘IT’ too. It does not pretend to be a complete or correct list. Your situation might require other things. But, it will give you some initial idea of what ‘Lean’ means in practice.

The purpose of this paper.

I have invented and developed a set of tools for planning ‘most anything’. The details are in my writings (gilb.com), and in particular in my books ([1] Competitive Engineering (2005), [2] Value Planning (2016))

The main body of these tools is called ‘Planguage’ (Planning Language). It is my nature to find intelligent ways of doing complex projects, and doing complex things. Consequently my Planguage is ‘naturally lean’.

Many people, like yourself are already convinced that lean approaches are a ‘good thing’. Certainly most every sane person wants to be efficient, to not waste time, people and money, etc., even if they have never heard of the popular ‘lean’ concept.

So the main purpose of this paper is to show the reader precisely which parts of Planguage, are a potential toolset, to help them be ‘as lean as they need to be’.

The simple answer, to the question of what Planguage has to offer for lean, is *‘all of Planguage’*: read the books. If you buy that idea, this overview is all you need. Stop reading this paper and start learning ‘Planguage’.

But, some readers will want more specific detail, more convincing arguments about *how lean* Planguage really is. If not for themselves, then for colleagues and clients, students and readers of their own writings. So the rest of this paper will argue the case in more detail.

The High level View of Lean

Let me arbitrarily, classify the basic ‘lean tactics’: i.e. ‘tactics for being efficient’. This is based on the detail in Figure 1 above.

1. Avoid Unnecessary Product.
2. Avoid Unnecessary Delay.
3. Avoid Unnecessary Bureaucracy.
4. Learn Fast.
5. Get Facts Right.
6. Seek Value for Stakeholders
7. Get good value for money
8. Avoid sub-optimization.

Everything else is a function of these ‘lean objectives’, or ‘lean principles’ if you prefer.

The result of doing practical things, like using Planguage specification and process tools, in order to follow these principles, should be greater efficiency in delivering the values, which are the core purpose of all projects.

Specific Planguage Lean Areas.

Here is a top-level overview of Planguage tools, that I believe can help you attain the Lean Objectives above.

1. RQT: Planguage for clear and full requirements (objectives) specification. End states.
2. DESIGN: Planguage for clear and full design (architecture, strategy) specification. Means to ends.
3. SQC: Specification Quality Control (SQC) to measure how good the requirements and designs *are*.
4. IET: Impact Estimation tables for quantitatively relating all levels ends and means. Good design?

5. EVO: Evolutionary Value delivery (Evo): an 'agile' project management process for delivering stakeholder value early and continuously and measurably; while 'learning' and ' correcting' continuously.

It may be worth adding at this point, that all of our methods are true 'engineering' methods. They are based on pervasive and logical *quantification* of all values, costs and process measures.

This implies that we can 'prove' that these lean Planguage methods work efficiently. We have decades of international case studies confirming that these ideas really are lean. And you can expect to do the same, prove by measurement, that you are indeed more 'lean': more cost effective than your older methods are. If you apply Planguage methods.

Specific Planguage Lean Tools

So let me go to a second level of level of detail on the lean tools.

1. RQT: Planguage for clear and full requirements (objectives) specification. End states.
2. DESIGN: Planguage for clear and full design (architecture, strategy) specification. Means to ends.
3. SQC: Specification Quality Control (SQC) to measure how good the requirements and designs *are*.
4. IET: Impact Estimation tables for quantitatively relating all levels ends and means. Good design?
5. EVO: Evolutionary Value delivery (Evo): an 'agile' project management process for delivering stakeholder value early and continuously and measurably; while 'learning' and ' correcting' continuously.

Figure 2: the top level of Planguage tool body of tools.

Here is some more, second level, detail: but by no means the most-detailed ideas at a practical level [See 1, 2, 3 for detail]:

1. **RQT: Planguage for clear and full requirements (objectives) specification. End states.**
    1. QUANTIFY: All variable requirements (values, qualities, costs) are always expressed quantitatively. No exception. Nothing critical is treated by Planguage, as 'soft', using management BS words (like *extremely high security*).
    2. MEASURABLE: all critical variable requirements, can be tracked and measured frequently as they emerge, and are being delivered by the project or the process. This confirms our estimates and theories; or invalidates them quickly. It is the basis of our learning: what works and does not work.
    3. RICH: all requirements are data-rich with all useful related information, such as who are stakeholders, justifications, related designs, minimum levels to avoid failure, goal levels needed for formal success, parameters for each requirement - about *when, where, who and under which conditions it is valid*.

4. PRIORITIES: there is a comprehensive set of information about requirements that enables us to compute their current and instantaneous priority, during a project, in spite of changes and complexity.

2. **DESIGN: Planguage for clear and full design (architecture, strategy) specification. Means to ends.**
    1. DETAILED: you can include enough detail about a design, so that it cannot be misunderstood by anyone (contractors), it can be thoroughly tested for complete implementation, it can be decomposed to allow early partial implementation for value flow. And the design detail will allow us to estimate the multiple values the implemented design *can* deliver, and the design's multiple long-range costs, and short-range costs: with useful accuracy.
    2. PURPOSES: every design (architecture, strategy, solution, means) must address at least one defined and 'official' accepted 'purpose (requirement). A set of designs must finally all deliver the requirements, within the defined constraints of the system. We document the intended purposes of every design, we estimate the degree of satisfaction the design is supposed to give us, and its costs - numerically. We know exactly what a design is supposed to do, and ultimately what it is *really* doing in practice. We know the design's intended efficiency, and real efficiency. We can *manage* the designs 'leanness'.
    3. RISKS: we document in detail, all suspected and known problems with a design, together with the core design idea itself. We are honest ethical, skeptical and transparent about design problems. Risks are documented at the individual design level, and may be systematically mitigated at that same design level. Finally, by being extremely quantitative about designs, we have very strong tools for managing risk of the designs, so that they deliver real value, at profitable costs. Most other methods *do not even try* to do this at all. Check it out.
    4. IMPACTS: The purpose of any design is to have an efficient impact. That means to deliver value early, at lowest costs. Planguage has a rich system of direct and numeric factual objective connections, between any or all designs - and the value and costs we need to manage. No fluffy assertions will be accepted. No design can survive our process if it really does not measure up. We manage the leanness of all designs from cradle to grave. From inception of design idea, and through the lifetime of the system.
3. **SQC: Specification Quality Control (SQC) to measure how good the requirements and designs *are*.**
    1. MEASUREMENT: all planning and specification artifacts, everything written by people in your system planning, needs to have a systematic quality control, with respect to your best practice standards. Requirements, architecture, contract. request for proposals, code, test plans. The QC should never be some kind of sign off, or 'approval'. It needs to be based on an objective measure of the *level of defects* in the document. This is a powerful upstream early (lean!) device, that is far more efficient than the usual discovery of problems downstream, bemoans of testing or with field use. SQC is proven roughly 100 times more 'lean'. (ref Terzakis, Intel 2013, [4])

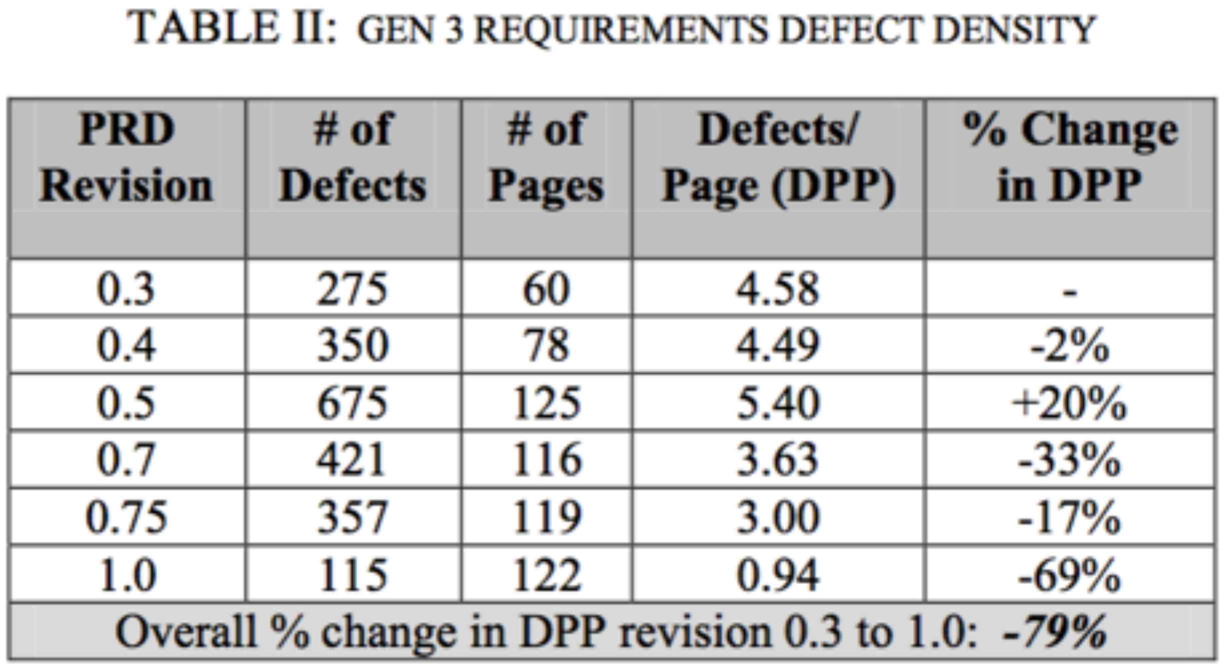


TABLE II: GEN 3 REQUIREMENTS DEFECT DENSITY

ng

PRD Revision	# of Defects	# of Pages	Defects/ Page (DPP)	% Change in DPP
0.3	275	60	4.58	-
0.4	350	78	4.49	-2%
0.5	675	125	5.40	+20%
0.7	421	116	3.63	-33%
0.75	357	119	3.00	-17%
1.0	115	122	0.94	-69%
Overall % change in DPP revision 0.3 to 1.0: ***-79%***				

Figure 3. Terzakis, Intel. 2013 [4] This use of our SQC, to measure delivered requirements written in our Planguage. Exit to next process is refused, until the requirements are delivered with an acceptable (economic, upstream level) level of defect density per page. Terzakis cited 200% to 300% engineering productivity increase in his projects, as a result of this upstream QC on the requirements to prevent *garbage in* downstream.

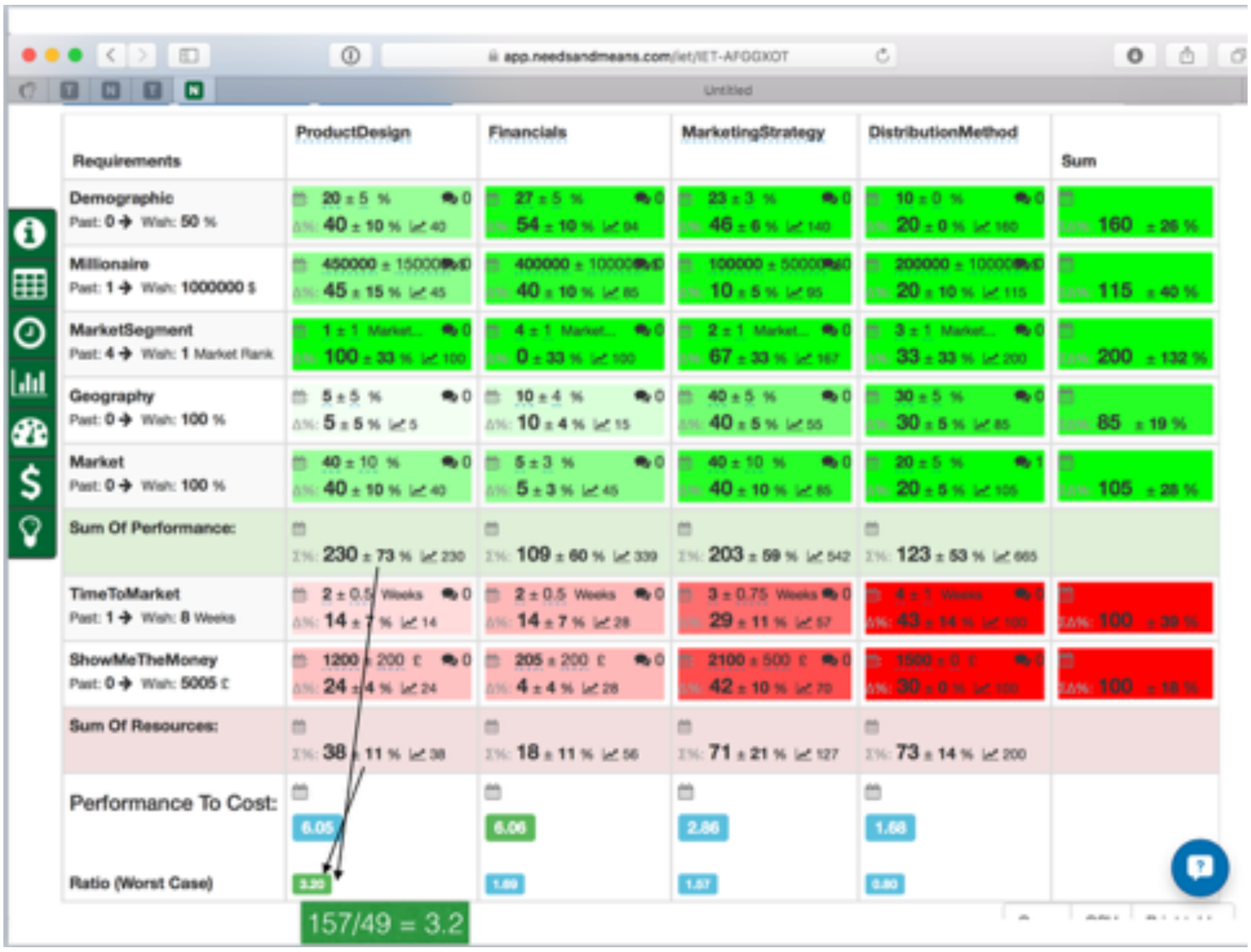
2. NUMERIC GATES: when the defect density of engineering specification work is measured, we have the opportunity to release work, only when the level of defects is so low, that it pays off to release it. Intel for example publishes use of our methods with a gate-exit defect level for requirements, of no more that 1 defect per 4 pages (2400 words). *You, by comparison, I believe based on my experience,* are *right now*, wallowing in filth (garbage in requirements, designs, contracts) with about 800 major defects per 2400 words, and you do not measure and do not know it. You just pay at the downstream, customer, end. You cannot see the microbes on your hands, but they are there. Hope you washed your hands afterwards, anyway. Se Fig. 3 above and ref [4].

4. IET: Impact Estimation tables for quantitatively relating all levels ends and means. Good design?

1. VALUES ESTIMATES: IETs insist that you estimate not only the primary value expected from a design, but that you also estimate all side-effects, on all other critical values (your top-ten objectives). This IE Table requirements/design development, is an engineering process, of analyzing side-effects, and is a *lean* way, compared to 'getting surprises later' (security destroys user-friendliness, for example)
2. COSTS ESTIMATES: IETs encourage you to estimate, all manner of initial, and long-term costs, which are *consequences* of a design. Capital expenditure, staff, time to develop initially as well as corresponding annual maintenance costs for lifetime. This will help us avoid costly surprises later, 'surprises' due mainly to *not even trying* to think about the costs of design or architecture (the norm, I am afraid).
3. RISK ESTIMATES: IETs capture, for all estimates of value delivered and costs, the upper and lower *range* of possible results (60±20 for example) as well as the *evidence* and *sources* for the estimates. The impact estimate evidence is used to determine how shaky the estimates are (Credibility scale 0.0 to 1.0). These data can be used to prioritize safer and more certain designs implemented early. Or the credibility ratings can be used to

consciously accept risk, while you are trying some new, uncertain, technology of high 'promise'.

4. PRIORITIZATION: IETs build a picture of *value delivered progress to date* as you evolve towards value targets, and a picture of consumed budgeted costs and deadlines. This update information about value delivered, and resources used, can be used to calculate current priorities (like 'focus remaining resources on un-met value targets'). This, in turn results in high values delivered early, in a prioritized stream of values. Very Lean.
5. ALIGNMENT: IETs can model, and numerically relate, any number of stakeholder levels of planning. Business levels, individual stakeholder types levels, and system development levels (like IT). The connection is numeric. The numbers say how the lower levels are contributing to the higher levels. This give management exceptional overview of how the whole organization is synchronized with top level objectives and top level strategies.



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	ΣΔ%: 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	ΣΔ%: 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	ΣΔ%: 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	ΣΔ%: 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	ΣΔ%: 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	ΣΔ%: 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	ΣΔ%: 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	1.89	1.57	0.80	

157/49 = 3.2

Figure 4: a real Impact Estimation table for planning a small business. Using the Needs and Means tool app. The app can show and hide detail as necessary, but the underlying detail is easily available as a mouse click. Tags like MarketingStrategy and MarketSegment are just cross references for much more detailed planning, you can drill down to detail as needed. To understand this table, as a reader of this paper, you would need to read up on it in a book [1,2]. But the table shows the things we have discussed in the paper. The green boxes represent 'best' options 6.06 for value for money, and the 3.20 is best with respect to worst case risk evaluation.

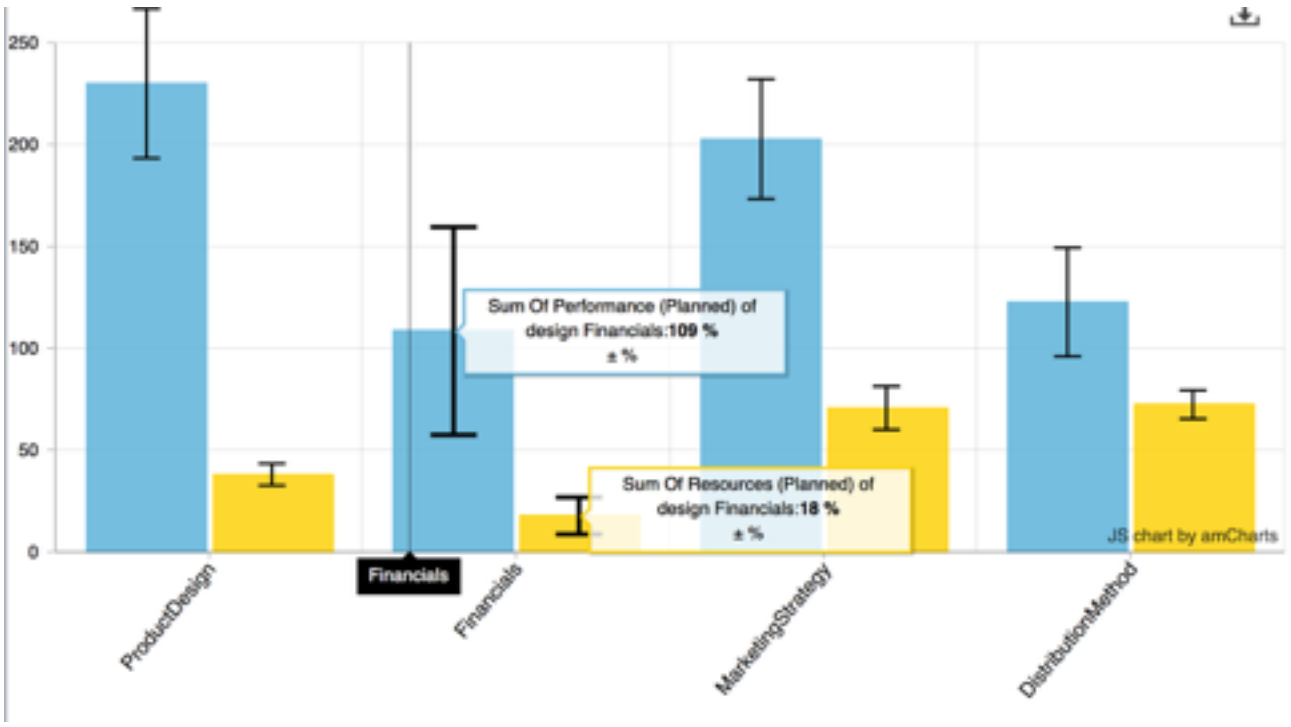


Figure 5 The needs and means tool can provide summaries of the table data such as this. it shows the design options, their total value, their total costs and their uncertainty (the I bars). The tool can sort the options into a prioritized sequence based on efficiency (value to cost), and or by riskiness.

5. **EVO: Evolutionary Value delivery (Evo): an 'agile' project management process for delivering stakeholder value early and continuously and measurably; while 'learning' and ' correcting' continuously.**
   1. ENGINEERING THE VALUE IN: Evo is an agile (incremental delivery) project management process, with a difference. It is focussed on delivery of multiple numeric targets of key stakeholder values. Other older PM process are focused on building a system, which we assume, or hope, will deliver value. But there is little or no actual quantification, estimation and measurement of these values, in older methods. And there is almost no measurement of the various costs (for example annual maintenance costs).
   2. LEARNING EARLY: Evo, since it is totally numeric, can sense that there are real deviation in delivery value and costs, by comparison with the estimated ones. This triggers quick analysis of causes and quick upstream cures better design, for example). In addition to getting the value delivery, back on track early, lessons are learned about how to do it right in future steps.

Summary

The hundreds of actual details, examples and case studies are already written up in the CE and VP [1,2] books, as well as freely available slides and papers [gilb.com). The reader needing more detail is referred to these sources.

Planguage, a synchronized collection of tools, supporting l*ean thinking* at the practical and detailed level. It is not a mere generic recommendation, that I see in so many papers and talks ('analyze the cause, prioritize value first, get it right the first time, design quality in').

Planguage is devoted to designing quality, value, and low costs; into any system, by conscious detailed numeric specification of desired levels of value and cost. Followed by conscious design with estimated impact values to rate alternative designs. Followed by quick (2nd week and every week) attempt to try out the highest priority designs, and then we can learn if they are as good as we thought they were.

The history of Evo, and of identical parallel methods such as IBM Cleanroom ([5] Mills and Quinnan) tells us that all projects are usually delivered on time, under budget, even for the most demanding, high-quality, large-scale, and complex projects. Agile itself has about 40% failure rate, and Scrum (source J. Sutherland) has 'only 19% failure rate'. These popular frameworks are not lean enough to consistently success in complex and large scale development. We could enhance the Scrum and Agile frameworks with Planguage methods to deal with this. Scrum delivers the code faster than their forerunners; but they do not deliver the values and costs, because they are not focussed on trying to manage value and cost. They do not even quantify their critical objectives such as 'security' amongst many others.

REFERENCES

1. Value Planning. leanpub.com/Valueplanning, or via Gilb.com
2. Competitive Engineering. 2005 Elsevier
3. needsandmeans.com Automated Planguage tool by Richard Smith, info@NeedsAndMeans.com
4. 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
http://selab.fbk.eu/re11_download/industry/Terzakis.pdf (this is the 2011 paper.
I don't have a good free link for his 2013 paper. But http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6636731&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6636731
Paper link requirs purchase and sign in

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