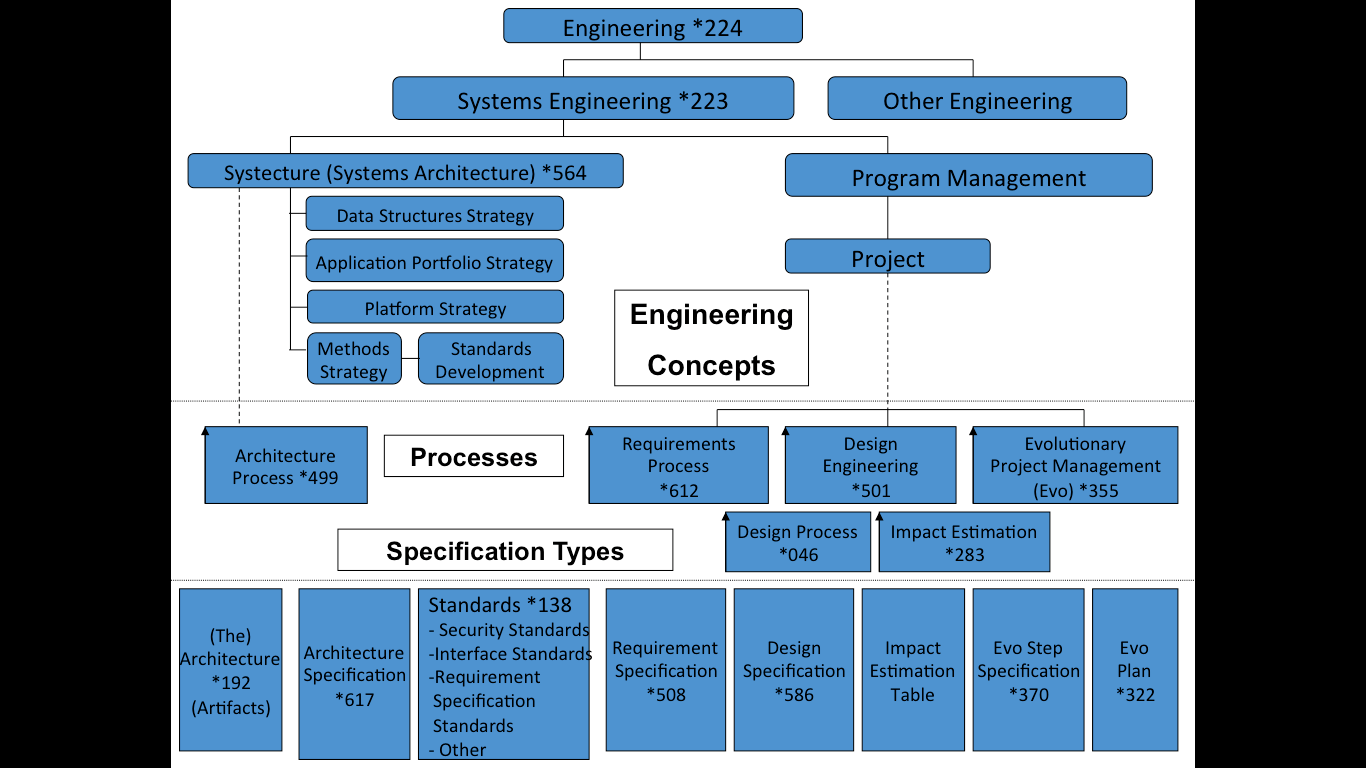
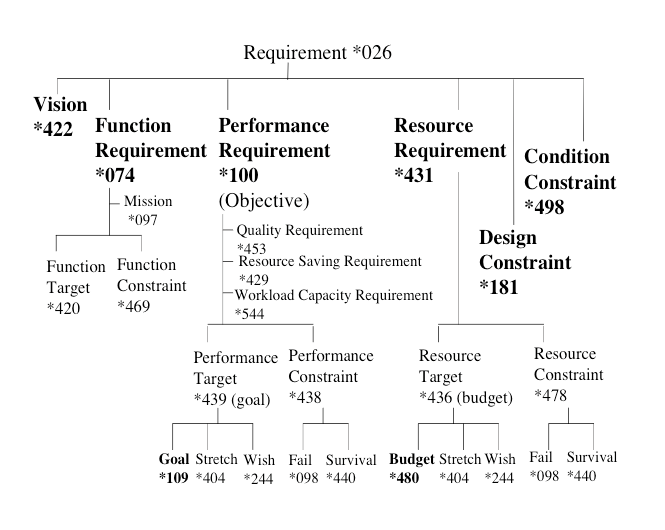
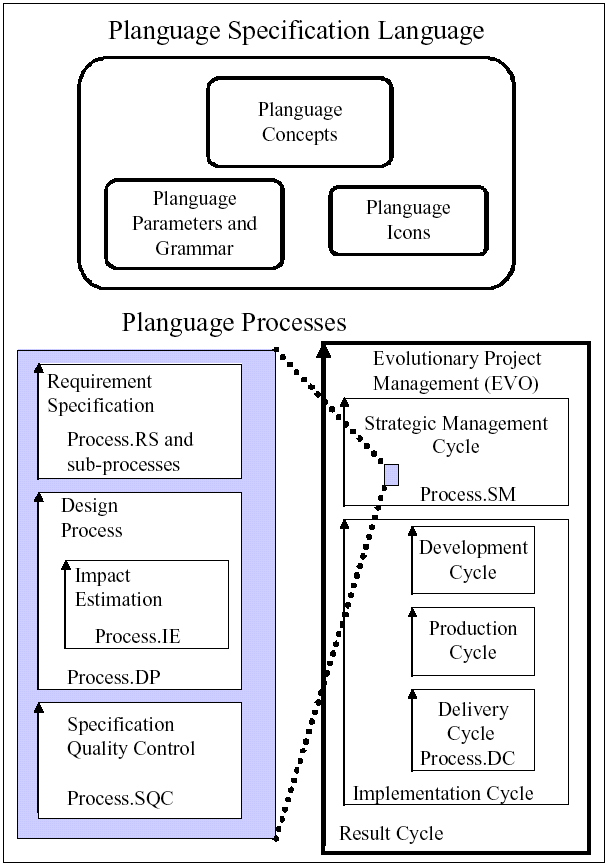
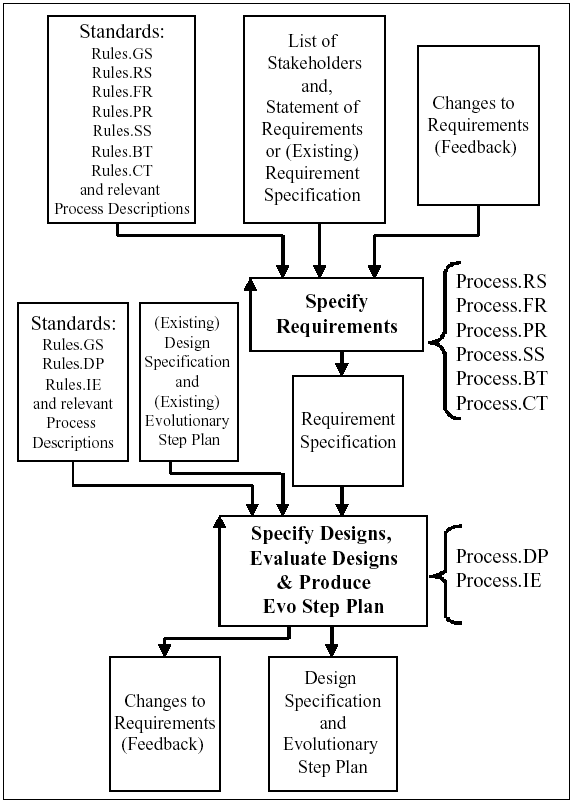
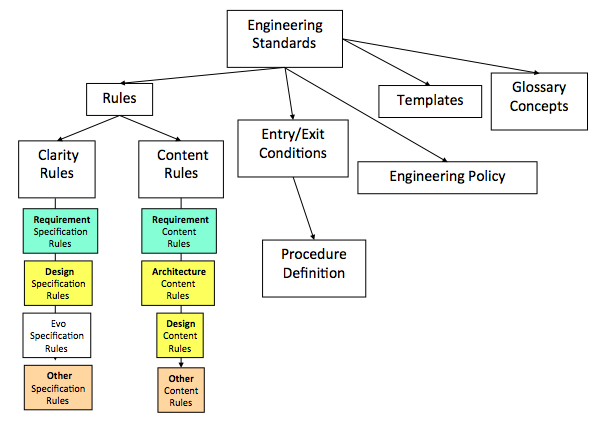
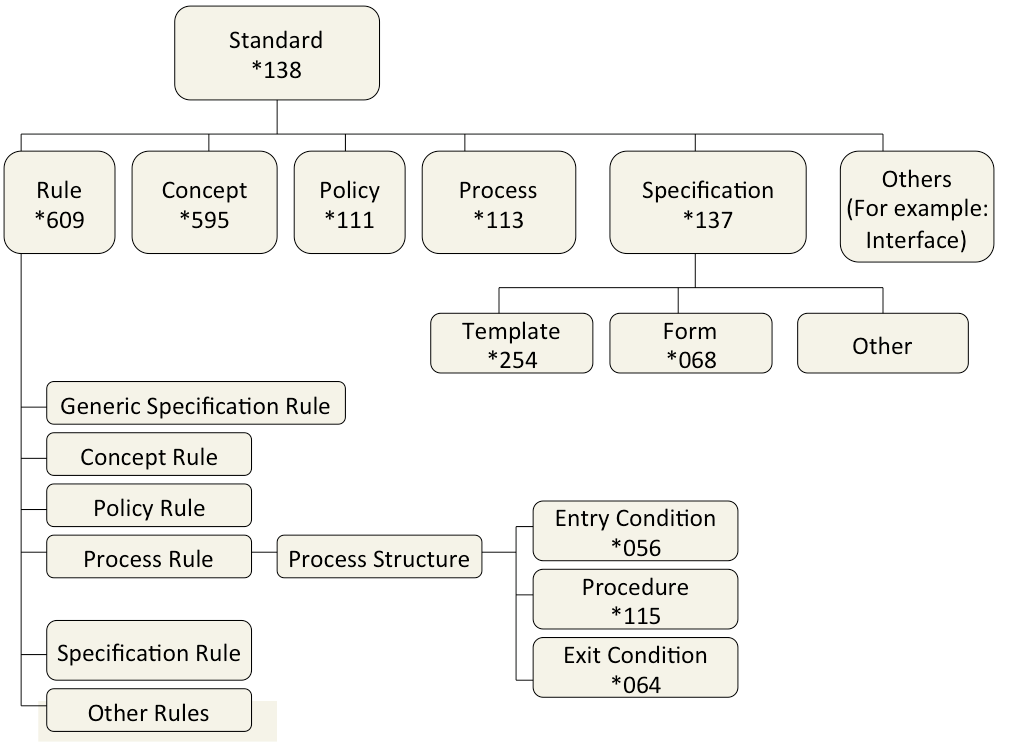
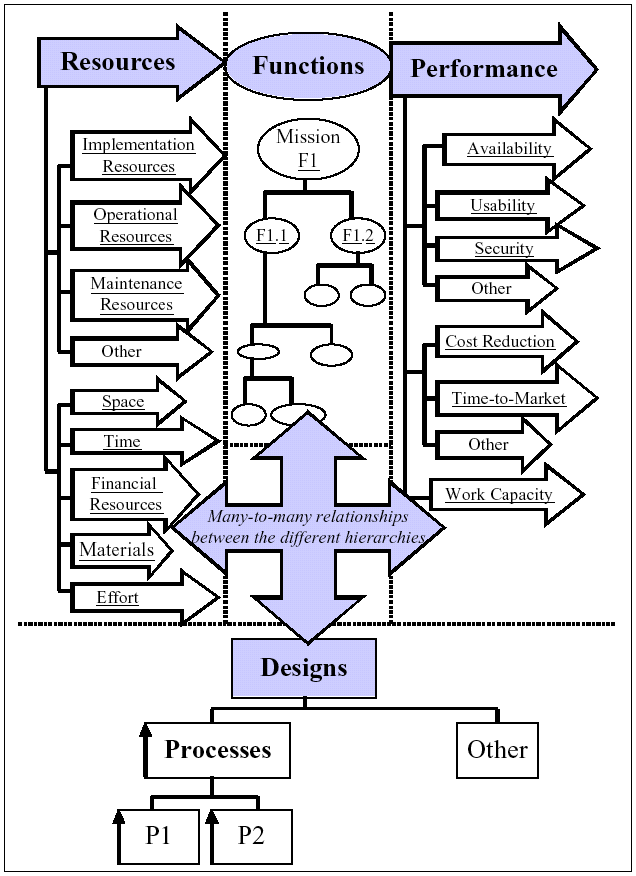
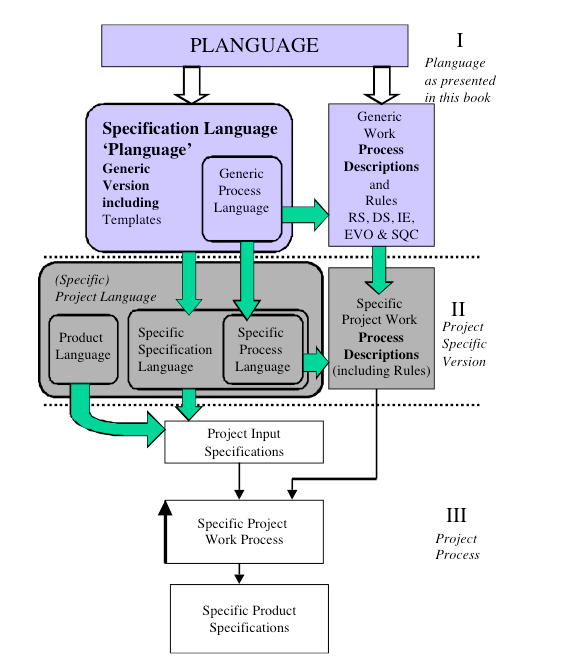
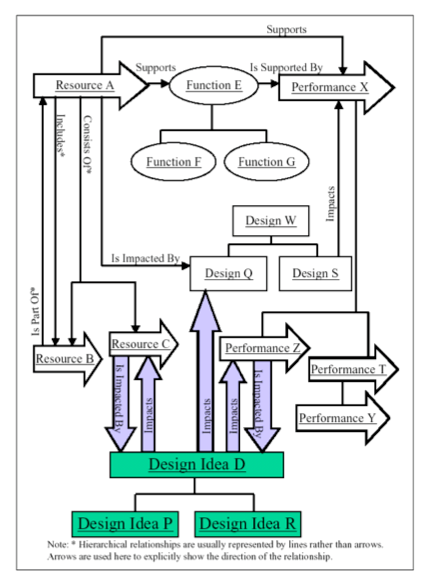
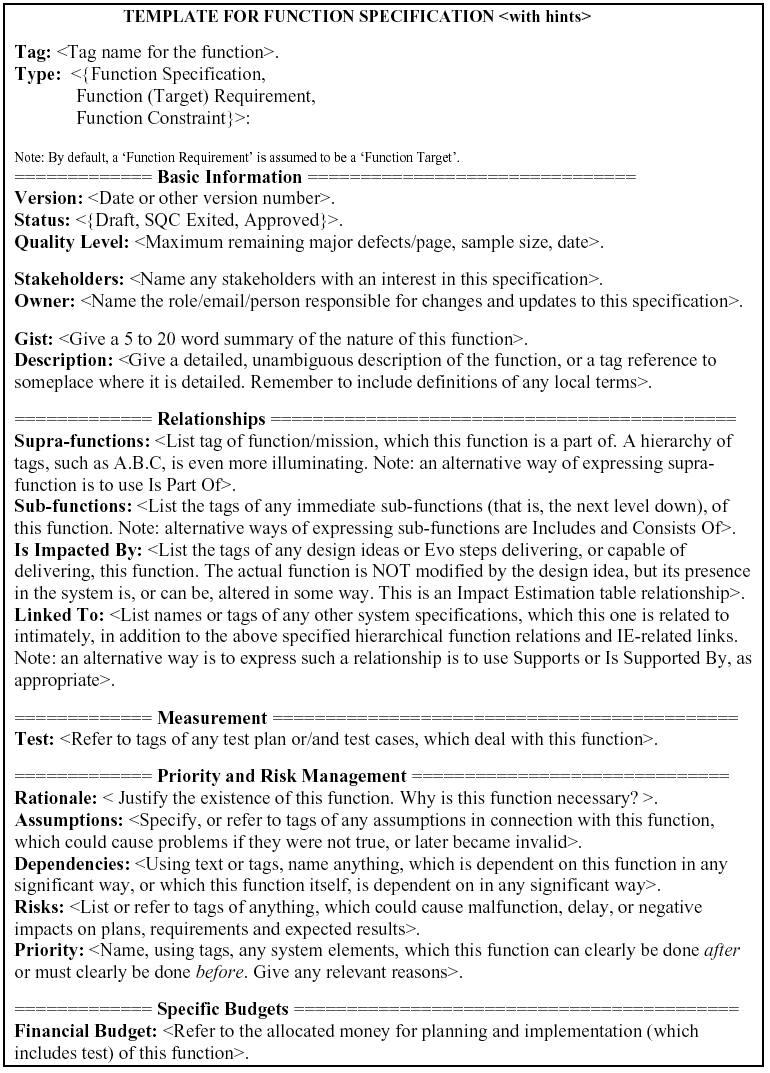
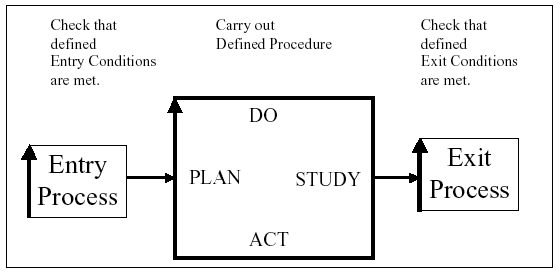
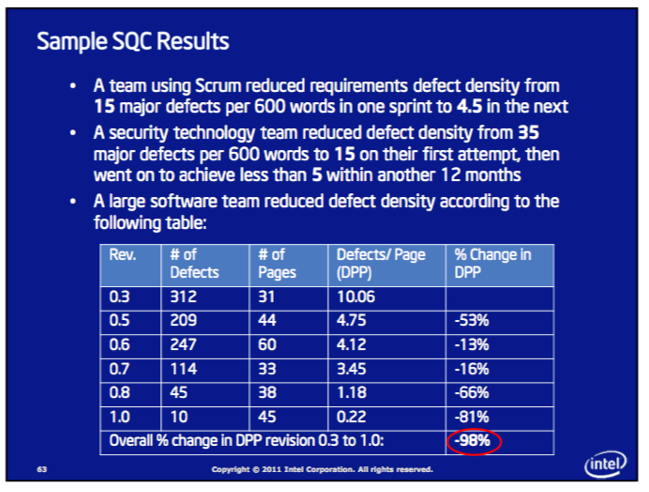
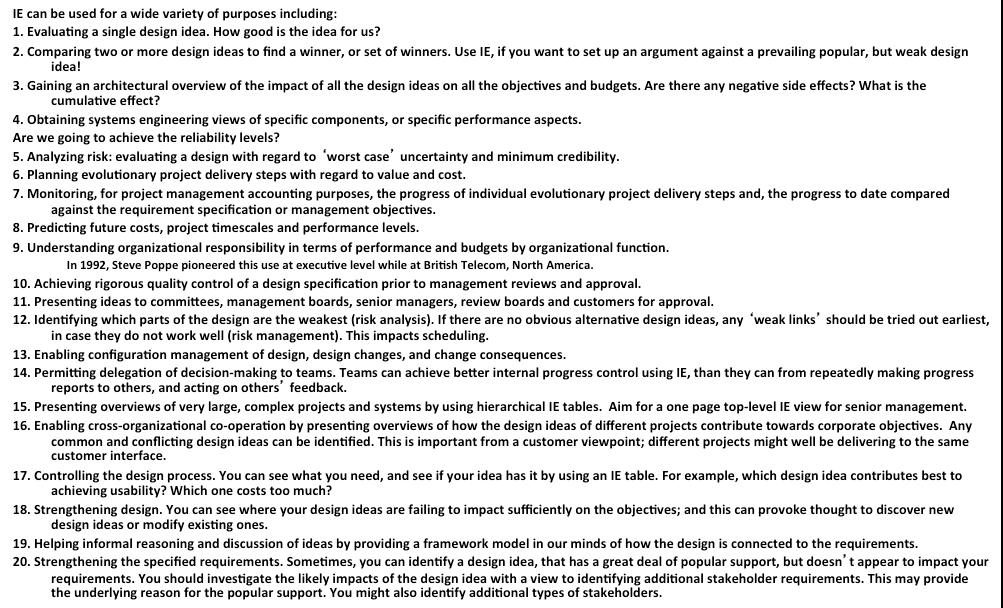
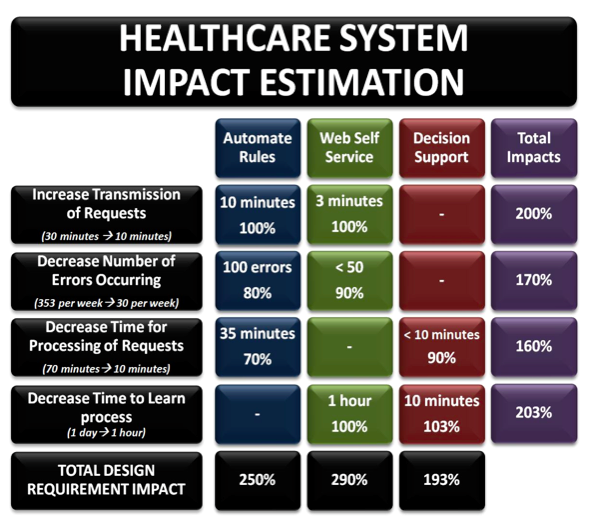
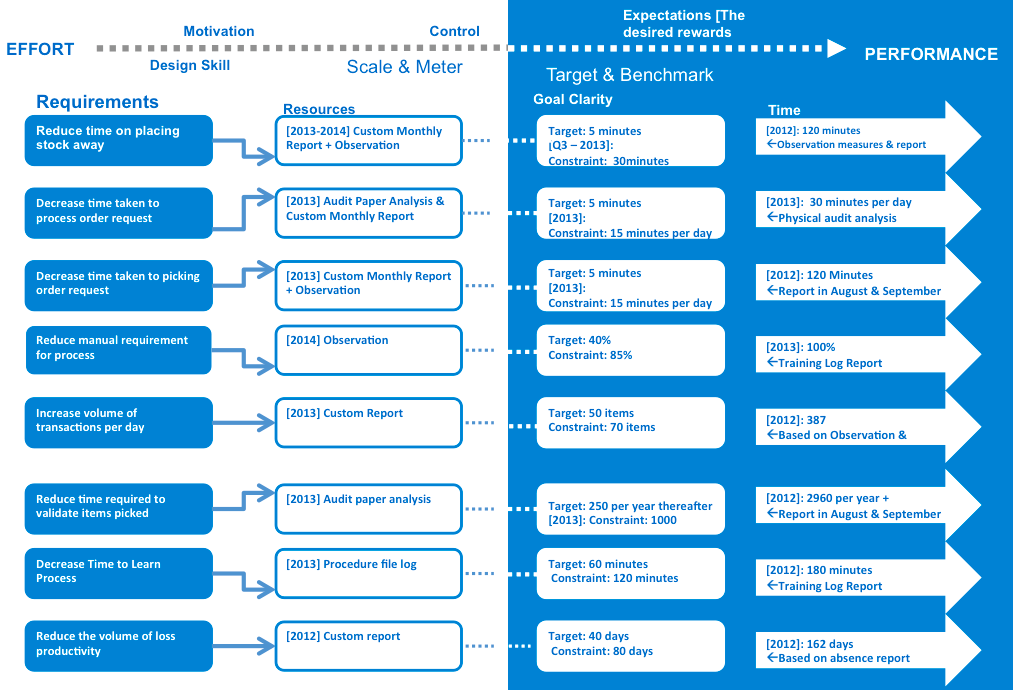
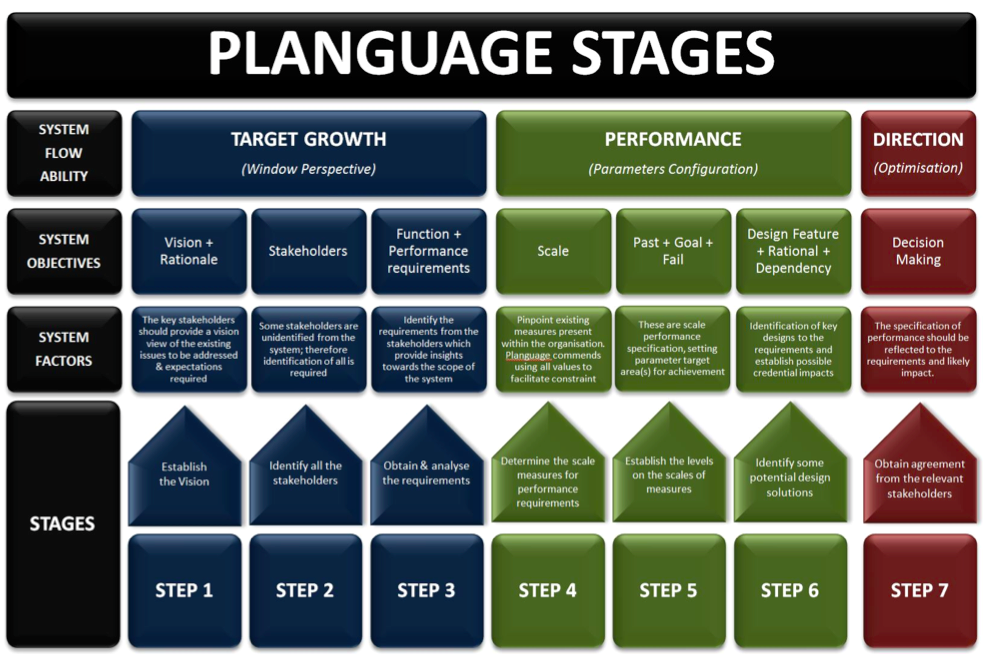
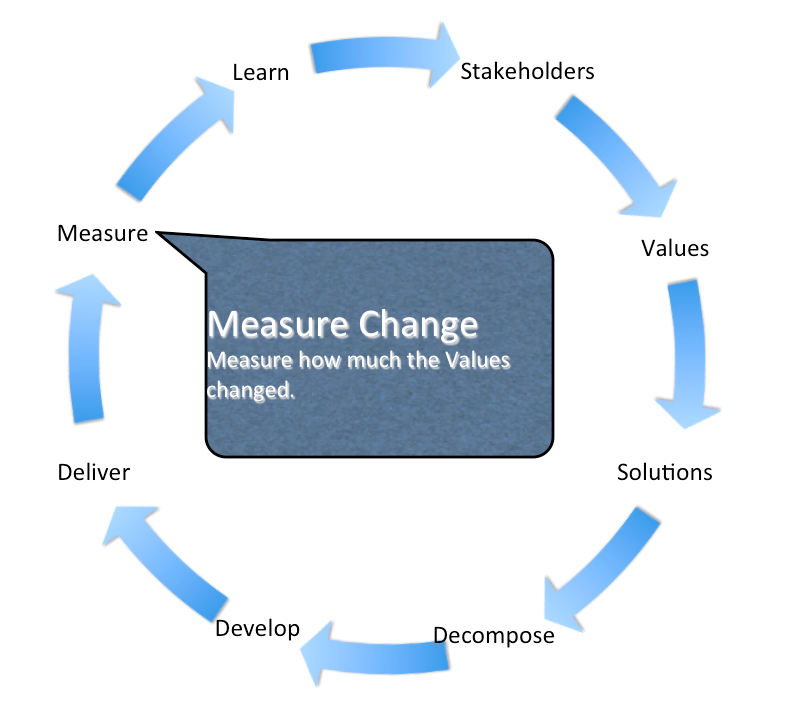
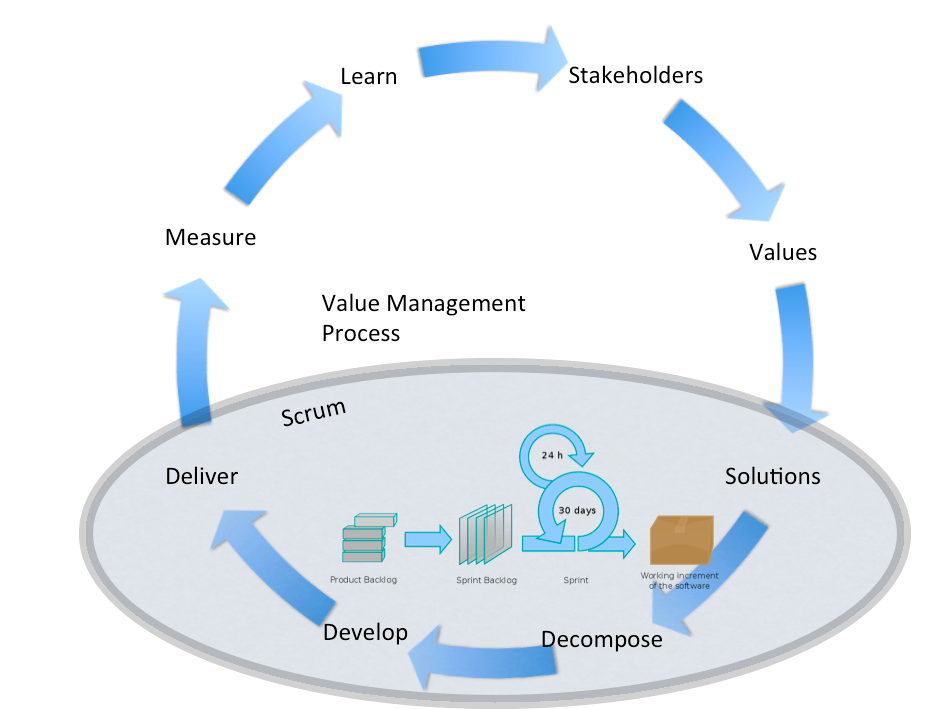
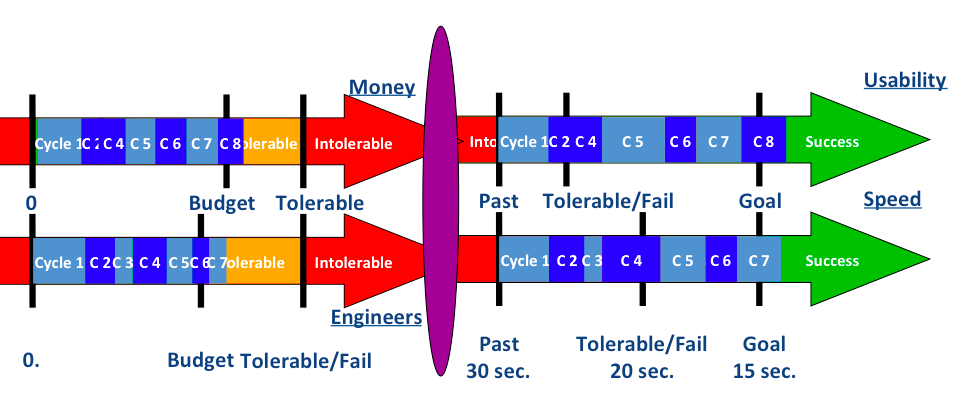
* **‘Planguage: an 'engineering' language and process for real software and Systems engineering - not 'programming'**
* By Tom Gilb
* **ABSTRACT**
* ‘Planguage’ is a general-purpose, systems engineering, planning language; for any system, including software systems. Planguage scope is requirements, design, project management, and quality control.
* It has been developed and practiced for decades (since 1960s). It is open source; anybody can use it for anything, in whole or part, freely.
* It is a large integrated ‘toolbox’, containing hundreds of distinct tools. Any set of these tools can be added to any other set of tools, or any framework. In particular, it is suitable as a set of ‘practices’ to evolve one’s own method within a stable Kernel.
* Planguage was designed to be interpreted by computers. The earliest automation was done by Prof. Lech Krzanik in 1978-9, on an Apple II in Forth, and published in his PhD Thesis, as well as in ‘Principles of Software Engineering’ (1988, ‘Aspect Engine’). Many automated tools have been made since, by Kai Gilb, and our clients, to support it’s use.
* Planguage was also designed to be ‘translated’ easily into any nation’s language. It includes a *graphical* representation language, as part of this.
* The *central* distinguishing characteristic of Planguage is it’s ability to directly integrate any quality (any ‘-ility’, not just reliability) statement quantitatively into the requirements, the designs, the project management, and the quality-control methods it contains.
* The *second* distinguishing characteristic of Planguage is that it allows and encourages very ‘rich’ planning specification of the background information for each individual requirement and design. This supports risk management, change management and dynamic prioritization.
* A *third* distinguishing characteristic is a systematic devotion to clarity and intelligibility of specification. Ambiguity, and lack of testable clarity is unacceptable. Even for ‘soft’ characteristics. Metrics, measurability and frequent numeric feedback about performance and costs is a primary notion.
* Planguage, with its project management component. ‘Evo’ (Evolutionary Value Delivery) is recognized as the ‘grandfather’ of Agile methods. Both in terms of earliest publication (1970s, 1980s [10]), and by Agile method developers. [10 B]
* The two largest scale adoptions of Planguage were at HP (from 1988), and is at Intel (over 17,000 engineers, over 10 years). A body of literature exists for this.
* Other noteworthy adoptions documentable, but often less than Corporate, sometime lasting only a few years include IBM (Corporate Quality Policy, CMM 4), ICL (1982, top management, sw product development), Boeing (1990, aircraft engineering QC, Process Error Prevention method), McDonnell Douglas (aircraft engineering, 1998-90), Citigroup (2003-2006), Credit Suisse, JP Morgan, Union Bank of Switzerland, Philips Medical Systems, Ericsson (ERA, 1990s), Nokia & Symbian, Microsoft (Test). There are many smaller and lesser known organizations and single projects for which we have published case studies.
* **BASIC DEFINITIONS**
* Here are my personal definitions, as defined in my book Competitive Engineering [2].
* Let me start by defining ‘engineering’, in general:
* **‘Engineering’** is
* • an Evolutionary Process, • using practical Principles, • in order to determine,• and identify the Means to deliver, • the best achievable Performance and Cost levels balance, • for optimal Stakeholder satisfaction, • in a complex risk-filled environment. [1, 2H1]
* **‘Software’**
* “Software refers to the ‘non-hardware’ aspects or components of a system. It specifically includes computer programs, data (computer readable files and databases), and software documentation and plans (any form of specification or plans made by people concerning software).” [2H1]
* **Software engineering** is the discipline of making software systems deliver the required value to all stakeholders. [2H2]
* **A System**:
* A system is any useful subset of the universe that we choose to specify. It can be conceptual or real. [2H2]
* In the *systems planning language* ‘Planguage’ [2], a system can be described fundamentally by a set of attributes. The attributes are of the following types
  + **function**: ‘what’ the system does
  + **performance**: ‘how good’ (quality, resource saving, workload  capacity)
  + **resource**: ‘at what cost’ (resource expenditure)
  + **design**: ‘by what means.’
* In addition, other factors describing various aspects of the system can be specified. These include:  requirements, dependencies, risks, and priorities. All these specifications (the attributes and the additional factors) need to be qualified by time, place and event conditions.
* **Systems Engineering** [2H2] is an engineering process
* encompassing and managing all relevant system stakeholders requirements, as well as all design solutions, and necessary technology, economic and political areas. The fundamental purposes of systems engineering are to:
  + optimize the system solution at the highest level of stakeholder concerns,
  + synchronize all contributing disciplines to contribute efficiently to the final system characteristics,
  + consider the entire system life cycle needs,
  + manage risks for the entire system and the entire system life.
* The big problem in software disciplines and current software culture, is total lack of engineering. It all seems to boil down to ‘programming’. I am also concerned that there is so much focus on ‘code’, that other essential elements of the software, such as data and planning (requirements, architecture), and user instruction (dialogue interfaces, training, handbooks), not to mention contracts and requests for proposals seem to disappear from the map.
* The fact that these same code-focused cultures, also show little to no interest in the wider system, of people, and hardware, is hardly surprising.
* **Value Centricity**
* I believe, a healthy departure, for discussion of these things, is to agree on the highest purposes of it all. I believe this can be summed up with a simple idea:’ Value’.
* I believe, all of us have one central purpose, or ‘meaning’ with our profession: **delivery of stakeholder values**.
* If software cannot contribute to values delivery: get rid of it.
* If systems cannot contribute to values delivery: get rid of them.
* It is not about ‘building’ software or systems: it is all about improving multiple value requirements, for multiple stakeholders, while using multiple resources wisely – ‘efficiently’ (‘value for money’).
* If we can deliver the necessary value streams to stakeholders without any ‘building’ whatsoever: then we should do so. If we cannot deliver value with software or systems that we ‘build’, then we should never have built them, and we should destroy them.
* There is of course a huge vested industrial interest in ‘building’ systems and software. We need to convert this interest into a new vested interest: *delivering stakeholder value efficiently.*
* This must be consumer and stakeholder led. The ‘builders’ have shown themselves incapable of such a transition, as long as the consumers have been willing to pay extravagantly for failed systems. Governments might conceivably lead the way [4]; and enlightened companies. [3]
* There is one fundamental ‘technical’ problem standing in the way. Defining the values measurably.
* The vast majority of managers, through lack of suitable training and culture, are completely incapable of articulating (quantitatively) their primary and critical ‘values’. [5]
* **Planguage: Articulation and integration of values**
* Our ‘Planning Language’ [2, 6], ‘Planguage’ (pronounced ‘Plan-guage’) is a direct practical solution to this problem, of lacking articulation of stakeholder values. It puts quantified and testable value requirements at the center of software/systems engineering. Everything else in Planguage is there to support giving the right values to the right stakeholders and the least resource use, for the entire system life cycle.
* In addition, there is a fundamental Planguage culture that these values need to be delivered early, frequently, incrementally, with measurement for learning and feedback, directing the ensuing project work.
* **Planguage as a set of optional planning tools**
* Planguage can be viewed as a fairly complete *planning* package for software and systems engineering. But in fact it consists of a very large number of components and sub-components. They are all systems engineering tools, and although they generally work best together, they can be used separately, and integrated into almost any other set of systems engineering tools.
* In practice that gradual adoption is the normal and best route to adoption. Evolutionarily. Making sure the ones you take into use work well, and are adopted warmly by professionals.
* **Some overviews of Planguage**
* ****
* Diagram 1: Planguage concepts. The \*nnn number refers to a formal definition of the concept in the Planguage Concept Glossary. [2H1]
* Planguage consists of a number of planning disciplines in the areas of requirements, design, and project management. These support multiple quantified value objectives, cost management, and risk management. They apply to software projects, IT projects, and any other systems engineering projects. They treat software, and all of its artifacts in essentially the same way they treat any other system components or systems. They focus on delivering multiple values to multiple stakeholders, while managing limited or budgeted resources to do so.
* They enable the system planners to deal with the entire life cycle of the system.
* That means Planguage can be equally well used to plan long term maintenance capability, as well a short term performance and security or usability.
* 
* Diagram 2: Planguage has a well defined and thought out set of concepts for requirements.
* In particular Planguage integrates quality requirements (-ilities, how well) with all other requirement types. It also allows and encourages thorough and rich requirement specification to support various system development disciplines, such as risk management, contracting, decision making, and value delivery. It makes a very clear distinction between real ‘ends’ (results, values), for defined stakeholders, and the ‘means’ to those ends (design, architecture, strategies).
* 
* Diagram 3. Planguage is a combination of defined languages (words, icons, numbers), and defined engineering processes.
* Planguage has a rich set of graphical symbols and over 676 formally defined concepts. It is based on well over 100 basic principles [2 G]. It is extendible, and modifiable, both at the corporate level, and the project level.
* 
* Diagram 4: Standards: Planguage is based on well-defined processes, and on well-defined ‘Rules’ for specification.
* These Planguage processes and rules all strongly support the management of quantified qualities and other values, in relation to budgeted resources. That is what we describe as an engineering process.
* 
* Diagram 5: Planguage has a wide variety of engineering standards.
* One interesting detail is that we have clearly distinguished between *clarity* of engineering specification and *content*. For example we have rules that values and qualities must be expressed quantitatively (clarity). But it also has rules that say these same quantified qualities should carefully distinguish between a target (value level to achieve), and a constraint (a minimum level for survival or partial payment) - content.
* Planguage also suggests a number of engineering management policies; such as estimation of the value and cost impacts of all architecture suggestions.
* 
* Diagram 6. An organized hierarchy of Planguage engineering standards. The \* number is a formally defined concept identifier, independent of the English terms used here.
* The Planguage Standards are terse, deep, powerful, and you can select, modify and improve them at will. They are also open source, and free.
* 
* Diagram 7: Planguage is tuned to the real-world complexity of many-to-many relationships.
* Planguage handles many levels of concern, multiple improvement objectives, for multiple stakeholders, multiple resource constraints, multiple functions, multiple designs, and multiple functions all in one integrated planning language. The language is designed for automated integration, and there are several automated support tools available. Many of these are simple variations on spreadsheets, often made locally by Planguage user companies themselves. None are commercially available.
* 
* Diagram 8: Planguage tailoring starts from the free, published, basics, and then can be tailored at the corporate level, for all projects, and then can be further tailored within a single project or a local component specification.
* 
* Diagram 9: Planguage supports a very large number of specific relationships between planning elements.
* This very pervasive use of pointers to related system components helps in change management, and risk management. You could go as far as saying that almost all Planguage statements express some kind of relationship to other components of the system planning.
* Here are some examples of relationship parameters:
* Authority
* Source
* Owner
* Author
* Implementer
* Impacts
* Supports
* Supported By
* Version
* Derived From
* Sub-component of
* Sub-components {list}
* Dependencies
* Contract
* Test Case
* Scenario
* Model
* And more!
* A sample of Planguage relationship parameters.
* 
* A Planguage ‘template with hints’ for Function specification, showing some relationship parameters.
* The hints are deleted electronically when content is filled in. We have found that ‘templates with hints’ are a good way to get engineers to specify necessary information required by the Rules and good practice. They are better than manuals, training courses and coaching to teach large numbers of engineers what is expected. They were initially designed by me, and trialed at McDonnell-Douglas (now Boeing) in 1988. They can feed into more integrated sets of planning documentation.
* 
* Diagram 10: Planguage engineering processes are based on strong gates in and out of each process, so that bad engineering does not flow downstream.
* Planguage is essentially ‘lean’ in the sense of focusing on getting it right upstream, and doing it right the first time.

Here are some applications of our quality control process, SQC, based on Software inspection technology, but now vastly simplified and reduced in cost.

* **- Reducing Time-to-Delivery**
* **- Measuring the Quality of a Document**
* **- Measuring the Quality of the Process producing the Document**
* **- Enabling Estimation of the Number of Remaining Defects**
* **- Identifying Defects**
* **- Removing Defects**
* **- Preventing additional ‘Downstream’ Defects being generated by removing existing Defects**
* **- Improving the Engineering Specification Process**
* **- Improving the SQC Process**
* **- On-the-Job Training for the Checkers**
* **- Training the SQC Team Leader**
* **- Certifying the SQC Team Leader**
* **- Peer Motivation**
* **- Motivating the Managers**
* **- Helping the Specs Writer**
* **- Reinforcing Conformance to Standards**
* **- Capturing and Re-using Expert Knowledge (by use of Rules and Checklists)**
* **- Reducing Costs**
* **- Team Building**
* **- Fun – a Social Occasion**
* Some uses of the ‘Specification Quality Control’ process
* All Planguage Process Outputs, all plans and specifications, are expected to undergo a simple, but numeric, quality control, in order to *measure* the degree to which specification rues have been followed in practice.
* This strongly motivates the rapid and practical adoption of Planguage. Standards get taken very seriously.
* 
* Case Study: Intel SQC, Source Erik Simmons slides, 2011, “21st Century Requirements Engineering” public talk.
* One of the best Corporate adoptions of Planguage is Intel (over 17,000 engineers using it and trained, over 10 years), and the Champion (teacher and coach) for that is Erik Simmons. Here is a sample of the results of using the Planguage SQC (Specification Quality Control) Process on requirements written in Planguage. The 3rd case cited 200-300% productivity improvement as a result [7].

Another major Planguage tool is the Impact Estimation Table. Which has a wide variety of engineering applications.

* 
* The Impact Estimation Table, in Planguage, can be applied to a very wide variety of systems engineering problems.
* 
* NHS Case A: A simple Impact Estimation table used in a successful UK Healthcare project, by Man-Chie Tse & Ravinder Singh Kahlon   
  {Man-Chie, [Ravi}@dkode.co](mailto:Ravi%7d@dkode.co) [8]
* The improvement values are on the left hand side, with an indicator of the formally defined values: the benchmark (example 1 day) and the target (example 1 hour). The design strategies are across the top. The estimates are both in real impacts on a defined scale of measure, and also the % of the way to the targets within deadlines. (100% = all the way on time).
* 
* NHS Case B: The Statement of Requirements for the Health System Case above.
* 
* NHS Case C: Planguage Stages as interpreted by Man-Chie Tse & Ravinder Singh Kahlon
* **The ‘Evo’, Evolutionary Project Management Process [2C]**
* The Planguage, project management process is based on the fundamental notion of continuous measurable feedback and learning what really works, in delivering planned value.
* As well as confirming that stakeholders are in fact really happy with the values we assumed at the outset they were saying they wanted. We could have misinterpreted. They could have changed their mind. New powers that be, like a new government, might have different priorities and earlier stakeholders.
* 
* Diagram 11. The ‘Value Delivery Cycle’: The Core of ‘Evo’.
* This cycle is not essentially different from the Deming Plan-Do.-Study-Act cycle. It is just more explicit about stakeholders and design aspects.
* Evo is focused on defining critical stakeholders, defining their critical needs (Values). Finding technical solutions to deliver those values. Decomposing those solutions into smaller deliverable part solutions, so we can get early, continuous and frequent delivery of value to stakeholders. Then we can measure the degree of delivery of values. And learn the truth of what works, and who really appreciates it.
* 
* Diagram 12: The value Delivery Cycle, overlaid with a Scrum Agile Process.
* A ‘Scrum’ process, and most other ‘software and agile’ processes, tends to be focused on *building code*. It should really be more open. Scrum, for ‘*systems* engineering’. For delivering and building anything necessary to deliver the stakeholder values.
* In any case - code or systems – we still need to consider the stakeholders, their real values (not user stories, the improvements at the top of their list!). The measurement of results, and the changes necessary as a result of these considerations. The upper part of the process above.
* These parts of Evo can be added to any other process such as Scrum, as was the case in Kai Gilb’s ‘Bring’ project in Norway. [9] Scrum alone, well done as Scrum, led to a failed system. When the real stakeholder values (not user stories, but speed and accuracy of customer access) were acknowledged, and designed for, it became a success.
* 
* DIAGRAM 13: Evo delivers multiple stakeholder values, measurably, while consuming budgeted resources.
* 
* DIAGRAM 14: the value delivery continues, cycle by cycle, until the Goal levels are reached; or until all budgeted resources are used up.
* Here is an example of practical experience with Planguage and Evo:
* Richard Smith 
* “However, (**our old** project management methodology) main failings were that
* it almost **totally missed the ability to track delivery of actual *value* improvements to a project's stakeholders**,
* and **the ability to react to changes**
* **in requirements and priority for the project's duration”**
* “The (old) toolset generated lots of charts and stats
* that provided **the illusion of risk control**.
* But actually provided very little help to the analysts, developers and testers actually doing the work at the coal face.”
* “The proof is in the pudding;
* I have **used Evo**
* *(albeit in disguise sometimes)*
* on two large, high-risk projects in front-office investment banking businesses, and several smaller tasks. “
* “On the largest critical project,
* the original ***business functions & performance objective* requirements document,**
* ***which included* no design,**
* essentially remained ***unchanged***
* over the **14 months** the project took to deliver,….”
* “… but the detailed **designs**
* **(of the GUI, business logic, performance characteristics)**
* **changed** many many times, guided by lessons learnt
* and **feedback** gained by delivering a succession of early deliveries
* to real users”
* “ In the end, the new system responsible for 10s of USD billions of notional risk, **successfully went live over one weekend**
* **for 800 users worldwide**, and **was seen as a big success**
* **by the sponsoring stakeholders.”**
* Case: Citigroup. Richard Smith, trained by us (2006), relates his experiences with Evo. In an Amazon Book Review of [2] & blog. [http://rsbatechnology.co.uk/blog:8]
* The Citigroup case makes several interesting points related to Planguage and Evo. Value delivery. Successful early cumulative value delivery. Stable top-level value requirements. Unstable ‘bad designs’, traditionally wrongly called ‘requirements’. Frequent measurement of the effect of the designs.
* **SUMMARY**
* Planguage is a rich set of systems engineering tools for integrated requirements engineering, design engineering and architecture, project management, and serious quality control of specifications and products.
* It has decades of practical experience in smaller projects and larger corporations; in systems product engineering, IT, and even in top management planning. It is most appreciated by serious engineering-culture product businesses, who are internationally competitive in quality and cost.
* It’s central practice is quantification of all qualities. This is integrated with functions, costs, designs, constraints, and other central system modeling components. Planguage’s central implementation practice is ‘Evo’, an incremental deployment and numeric feedback process, for learning, value delivery, and progress tracking.
* Planguage ideas and tools are free, and can be adopted or modified as practices in any method.
* **References**
* 1. Billy Koen
* My definition of Engineering is based completely on Koen’s excellent definition (1984), and only terminology is changed.
* Koen, Billy V. 1984. **Toward a Definition of the Engineering Method.** Proceedings of the ASEE-IEEE Frontiers in Education. 14th Annual Conference, Philadelphia, PA. 3-5. October 1984. Pages 544–549. The paper also appeared in Engineering Education. December 1984. Pages 150–155. Also in Spring 1985 in The Bent of Tau Beta Pi. Pages 28–33.
* Professional Website; (2014)
* <http://faculty.engr.utexas.edu/koen>
* Koen, Billy Vaughn. January 2003. **Discussion of the Method: Conducting the Engineer’s Approach to Problem Solving**. Oxford University Press. ISBN 0-195-15599-8. Pages 260. http://www.me.utexas.edu/faculty/people/ koen.shtml/.
* \*
* Video Presentations 2007 Illinois Lecture
* <http://faculty.engr.utexas.edu/koen/etc-lecture>
* 2. Tom **Gilb**: “**Competitive Engineering**” **2005**
* A. The book
* Gilb, Tom, Competitive Engineering, A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage, ISBN 0750665076, 2005, Publisher: Elsevier Butterworth-Heinemann. Sample chapters will be found at Gilb.com. [2 B, 2 C]
* B. Chapter 5: Scales of Measure:
* <http://www.gilb.com/tiki-download_file.php?fileId=26>
* C. Chapter 10: Evolutionary Project Management:
* <http://www.gilb.com//tiki-download_file.php?fileId=77>
* D. Chapter 8, Specification Quality Control
* E. For Extensive technical detail on Spec QC above [2 D] see Gilb & Graham, Software Inspection, 1993, Book.
* F. Planguage Rules Collection from all chapters
* <http://www.gilb.com/dl829>
* G. CE Principles
* **Gilb Principles collection**
* <http://www.gilb.com/tiki-download_file.php?fileId=352>
* **Competitive Engineering** **book Principles**. Gilb and Others 8MB.
* Extended Collection, **Undergraduate Basics (Paper)** Principles, Gilb’s Datamation **Laws of Unreliability**, ICL **Bill of Rights**, Demarco and Gilb’s Law of Measurability, Risk Principles, Clinical Risks Slides, GILB’S INTERPRETATION OF **ERICSSONS CORPORATE QUALITY POLICY**, **Decomposition Principles** (from CE 10), **12 Tough Questions**
* H. Planguage Glossary
* 2H1: The extended Planguage Glossary,Tom Gilb’s *personal* version
* ‘CE Full Glossary’, This is updated at Gilb.com periodically
* <http://www.gilb.com/dl830>
* Version 18 Oct 2014 is version as of writing this Chapter..
* 2H2. The ‘Competitive Engineering’ *book* Glossary, and the whole CE book, outside the book glossary, as a source of defining Planguage terms.This CE Glossary is 1/6 size, heavily edited, of the full Glossary [2H1]
* [**http://www.gilb.com//dl540**](http://www.gilb.com//dl540)
* **3. No Cure No Pay**
* Agile Contracting for Results The Next Level of Agile Project Management: Gilb's Mythodology Column Agilerecord August 2013
* <http://www.gilb.com//dl581>
* 4. Brian Wernham, Agile project Management for Government.
* http://www.amazon.com/Agile-Project-Management-Government-Wernham/dp/0957223404
* 5. Quantifying Management and Stakeholder Values
* **“Quantifying Management Bullshit”**
* <http://www.coremag.eu/fileadmin/Papers/Quantifying_Management_TGilb_core3.pdf>
* version 18 april 2011
* Quantifying Management Bullshit: forcing IT Stakeholders to reveal the value they really want from your IT Project.
* By Tom Gilb
* <http://www.requirementsnetwork.com/node/2820>
* RQNG 25 May 2011
* Quantifying Management Bull: Forcing IT stakeholders to reveal the value they really want from your IT project
* <http://www.gilb.com/dl465>
* CORE Version
* 6. T. Gilb “Value Planning”, Book Manuscript Fall 2014
* <http://tinyurl.com/valueplanning>
* For later info see gilb.com
* 7. **Intel Specification Quality Control Experience + Planguage**
* “The Impact of a Requirements Specification on Software Defects and Other Quality Indicators”
* John Terzakis
* Intel Corporation
* August 31, 2011 RE 11 Conference Trento, Italy
* <http://selab.fbk.eu/re11_download/industry/Terzakis.pdf>
* 8. **Tse and Kahlon: Planguage Case NHS**
* Source Oct 2013 Paper
* **How Planguage Measurement Metrics Shapes System Quality**
* **Man-Chie Tse**1,2 **& Ravinder Singh Kahlon** 1,2
* {Man-Chie, Ravi}@dkode.co
* Title of 2013 SQM paper:
* the principles and application of Planguage for Managing System Innovation
* <http://www.gilb.com/dl582>
* Great real pharmaceutical hospital case with real improvements.
* Slides <http://www.gilb.com/dl583>
* 9. **Bring Case** and more: Hierarchical Impact Estimation Tables
* <http://www.gilb.com/dl500>
* **A. "Value-Driven Development: Principles and Values."**
* Slides for , 50 minute talk, Software Passion Conference 20 March 2012 Gothenburg, Sweden
* **B. Value Management**
* **(Evo)**
* **with Scrum development**, March 2010 English Version , Kai Gilb
* [www.gilb.com/tiki-download\_file.php?fileId=277](http://www.gilb.com/tiki-download_file.php?fileId=277)
* **C. The Inmates are running the asylum, Construx Summit talk Oct 25 2011 Seattle**
* **Contains considerable Bring Case slides**
* [www.gilb.com/tiki-download\_file.php?fileId=488](http://www.gilb.com/tiki-download_file.php?fileId=488)
* 10. T. Gilb, **Principles of Software Engineering Managemen**t,
* 1988
* Internet Chapters (text only, no illustrations):
* A. pdf ‘Ch 14 POSEM Productivity’ gilb.com/dl560
* B. pdf ‘Ch 15 POSEM Deeper Perspectives on Evolutionary delivery gilb.com/dl561
* This includes a page extra of quotations from Agile Gurus crediting it as inspiration for them, and it being first.
* C. Chapter 13.4 (page 237-241) Open Ended Architecture
* D. Chapter 21 ICL Inspection Experiences
* 11. URL31 **HP Evo**
* **A. The Evolutionary Development Model for Software**
* by Elaine L. May and Barbara A. Zimmer
* August 1996 Hewlett-Packard Journal
* <http://www.gilb.com/tiki-download_file.php?fileId=67>
* **B. Evolutionary Fusion: A Customer- Oriented Incremental Life Cycle for Fusion**
* by Todd A
* <http://www.gilb.com/tiki-download_file.php?fileId=35>
* August 1996 Hewlett-Packard Journal
* **C. RAPID AND FLEXIBLE PRODUCT DEVELOPMENT: AN ANALYSIS OF SOFTWARE PROJECTS AT HEWLETT PACKARD AND AGILENT** (2001)
* by
* Sharma Upadhyayula
* <http://www.gilb.com/tiki-download_file.php?fileId=65>
* M.S., Computer Engineering University of South Carolina, 1991
* And
* Massachusetts Institute of Technology
* January 2001
* **D. Best Practices for Evolutionary Software Development**
* by
* Darren Bronson
* <http://www.gilb.com/dl825>
* 57 pages., 1999.
* URI: http//hdl.handle.net/1721.1/80490
* E. Note: these references focus on Evo part of Planguage. There is also a body of Literature for the HP Adoption of our Inspection method (alias Spec QC, in Planguage).
* Grady, R. B. and Van Slack, T., “Key Lessons in Achieving Widespread Inspection Use”, IEEE Software, V. 11, N. 4, Month, 1994, pp. 46-57
* <http://dl.acm.org/citation.cfm?id=140207> (paid download)
* 12. **Intel Planguage Experiences**
* A. **Intel Report on SQC** (Gilb methods used here <- E Simmons)
* The Impact of a Requirements Specification on Software Defects and Other Quality Indicators by john.terzakis@intel.com
* **http://selab.fbk.eu/re11\_download/industry/Terzakis.pdf**
* B. **Intel Experience with Planguage and SQC 2011**
* Erik Simmons, Intel, 2011, 21st -Century Requirements Engineering: A Pragmatic Guide to Best Practices, Erik Simmons PNSQC 2011 (Pacific Northwest Software Quality Conference)
* http://www.uploads.pnsqc.org/2011/slides/Simmons\_21st\_Century\_Requirements\_slides.pdf
* Tom Gilb is a freelance consultant (since 1960), teacher and author serving clients mainly in Europe, and the US. He joined IBM 1958.
* He has books in print: “Competitive Engineering”, “Principles of Software Engineering Management” and “Software Inspection”. A new book, tentatively titled ‘Value Planning’ (about Planguage) is drafted Summer 2014.
* He specializes in software engineering, systems engineering, and technical management. He resides in Norway and London.
* He is the inventor of ‘Planguage’: working on it for decades. Appearing in many books (like ‘Software Metrics’ 1976) and papers, as it got refined. The development has mainly been in connection with using it as a tool for client projects.
* He is an Honorary Fellow of the British Computer Society.
* Tom participated at the initial SEMAT, Zurich meeting, 2010.
* His most recent papers, book manuscripts and slides are available on [www.gilb.com](http://www.gilb.com)