Undergraduate Basics for Systems Engineering (SE), using The Principles, Measures, Concepts and Processes of Planguage.

Abstract.

There are some very basic things that systems engineers should be taught. These things are both fundamental and classic. They are *fundamental* because we can reuse them in a very wide variety of SE situations. They are *classic* in the sense that they have a very long usefulness half-life. They are probably useful for at least a career lifetime. When I was in my Twenties, I decided to collect, to learn and to develop these SE Basics. Now, in my Sixties, I am more than ever convinced that these fundamentals should be share with students. The fundamentals are: Principles (heuristics, laws), Measures (ways to quantify critical factors), Concepts (really useful definitions of fundamental SE ideas), and Processes (really useful SE processes). I have published these in several books and papers already. I would like to argue here why they need teaching in undergraduate systems engineering. I believe that their usefulness and longevity are demonstrated in my own work, are acknowledged by many professional colleagues and some academics, and are self-evident upon examination. Hopefully this paper can stimulate others to adopt at least the general idea, if not my exact artefacts.

Principles

Some Principles of Useful Knowledge

- 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 when 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.

The Notion of **Usefulness** of Principles:

A principle is a short statement that guides people to take certain decisions or action. It is condensed wisdom. Principles are useful if they remind or teach us to act in a better way than we otherwise would do.

For example:

"There is lots of uncertainty and risk of deviation from plans in any project.

You cannot eliminate risk. But, you can document it, plan and design for it, accept it, measure it and reduce it to acceptable levels. You may want to avoid risk, but it doesn't want to avoid you."

Source: CE, page 23.

This principle tries to warn about the inevitability of risk. It also is specific about what you can do about risk. It teaches that you cannot eliminate risk, but you can try to manage it in various ways.

From the departure point of this principle, the teacher can then be more specific on how to identify, specify and mitigate risks.

The Notion Of Half Life of Principles

If a principle became obsolete in a few years – perhaps because of new technology or new economics, then it would be less valuable to learn, and might even be dangerous to continue to practice beyond its true lifetime. So I prefer principles that we can imagine 'always were true', and we can so no clear reason why they 'will not be true for the foreseeable future".

It takes decades from when a principle is stated, until it becomes taught in any substantial way. The student has decades of their future in which to apply a principle. So it makes good sense that the principle is something we can rely on in the long term.

The Notion of **Fundamentality** of Principles

Principles should be *fundamental*. They should be basic tools for everyday use in planning, engineering, discussing, decision-making, and reasoning. We should be able to use them as the basis for all our more-detailed actions and thinking processes. For example:

"8.The Principle of 'Quality In, From the Beginning'

Quality needs to be designed into processes and products,

Cleaning up bad work is a loser, but cleaning early is better than late.

A stitch in time still saves nine,

But an ounce of prevention is still worth a pound of cure. "

Source CE, page 24.

The above principle applies to all engineering and management planning work. WE humans seems to have a strong natural tendency to clean up our faulty work when it is discovered, rather than to consciously discover how we can prevent the faults from getting into our work in the first place.

This principle is at the heart of CCI Level 5 (Defect Prevention).

This principle is fundamental. It is at the basis of all improvement efforts in a systems engineering process. It is the basis for a paradigm shift for many professionals I deal with; the shift from ‘fix problems’, to ‘prevent problems’. Students should be taught such profound principles before they waste years discovering them, if at all.

In Competitive Engineering I have offered 100 such principles. I have ‘brainstormed’ many more in other books and papers, including this one. I am sure my many systems engineering, and other discipline’s colleagues have and will continue to develop principles that deserve to be taught formally. My concern is that we place far too little emphasis on selecting and teaching these principles. My concern is that students do not even get a dozen good principles to base their professional work on. I think we need a course called something like “The Most Important Systems Engineering Principles”.

Measures

One single experience overshadows all others in my technological wanderings. Engineers do not seem to have been taught how to quantify most of the critical quality aspects of their systems.

Most real engineers have been taught how to deal with qualities like availability and reliability. But these are just two of hundreds of quality aspects we meet when engineering systems.

Dr. Hastings of MIT, in describing the changing face of systems engineering spoke of conventional SE with

"Focus on reliability, maintainability, and availability"

and referred to Expanded SE as having an

"Emphasis on expanded set of 'ilities' and designing in robustness, flexibility, adaptability in concept phase".

I agree. But we are not being trained to do so. The textbook literature is extremely sparse on the subject. Most all professional engineers I meet have never seen this done in an engineering manner, by defining the system requirements quantitatively. It is not sufficient to state slogans ('we need more robustness") and then throw in all the robustness technology we can think of at the moment. But that is an good description of what I see done in practice.

The problem is that we do not even teach basic patterns of defining these ilities measurably.

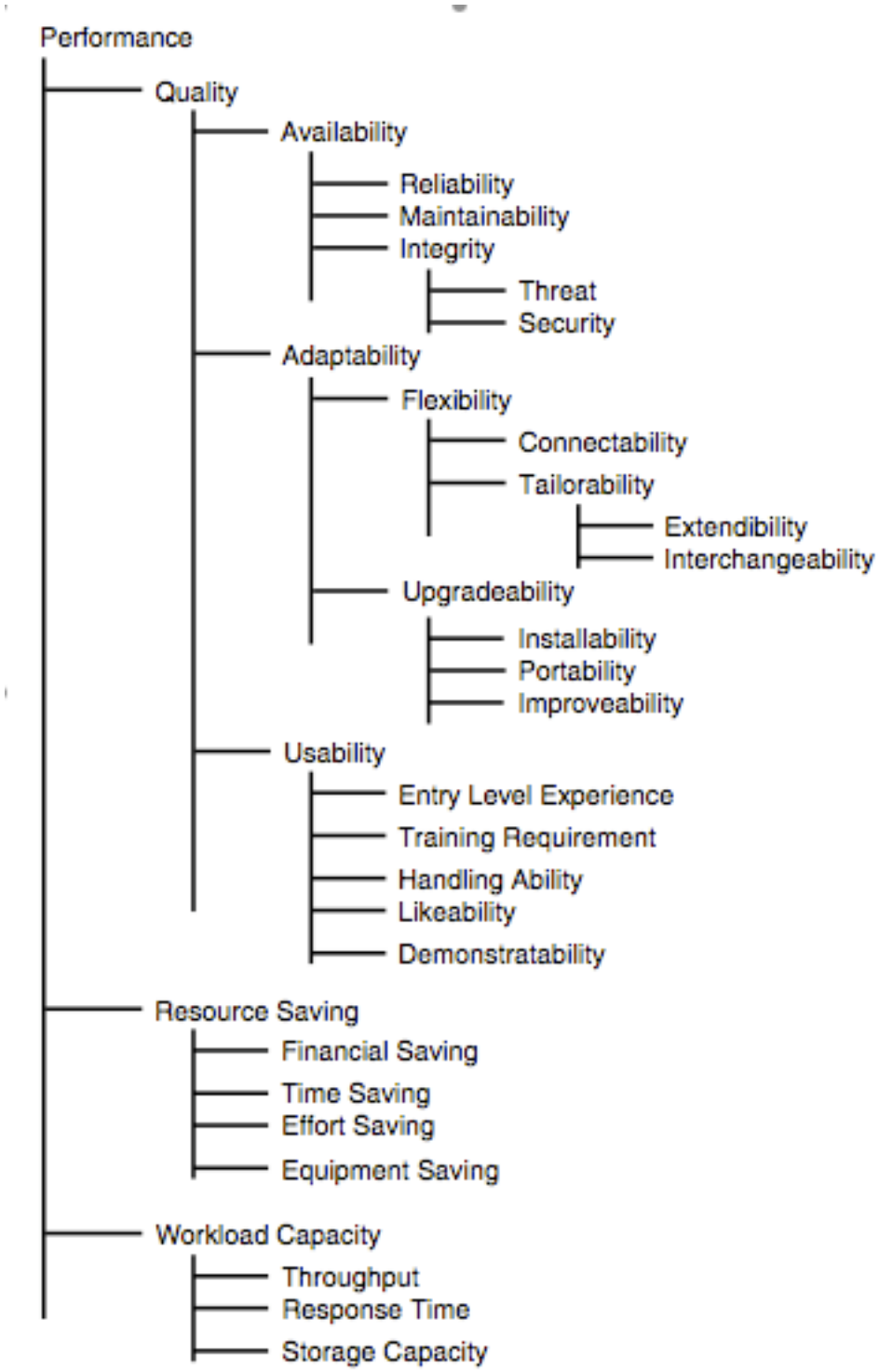

Illustration: Some quality concepts and their possible decompositions. Source CE, page 154.

Even more rarely are we taught specific examples of scales of measure for things like adaptability and usability. But people are constantly naming them as things they want to build into their systems.

Portability: 'The cost to move from location to location.'
Scale: The cost to transport a defined [System] from a defined [Initial Environment] to a defined [Target Environment] using defined [Means].
Type: Complex Quality Requirement.
Includes: {Data Portability, Logic Portability, Command Portability, Media Portability}.

Example of scale of measure template. Source CE, page 158.

The basic process for finding quantification of qualities are:

1. find a suitable template that you find usable
2. modify a reasonably close template to be more like what you need
3. search the web for what you need (you get 200,000 hits from 'portability metric')
4. decompose (Cartesian analysis) high level quality concepts – as in the diagram above, to more elementary and more directly measurable concepts.

A general process example, for '' is given in the process section below.

If we increase our ability to quantify qualities, then we also increase our ability to do a wide range of engineering activities better.

We can:

• Simplify requirements (if the top few requirements are quantified, there is less need for copious documentation as the developers are focused on a clearer, simpler 'message');
• Communicate quality goals much better to all parties (that is, users, customers, project management, developers, testers, and lawyers);
• Contract for results. Pay for results only (not effort expended). Reward teams for results achieved. This is possible as success is now measurable;
• Motivate technical people to focus on real business results;
• Evaluate solutions/designs/architectures against the quantified quality requirements;
• Measure evolutionary project progress towards quality goals and get early & continuous improved estimates for time to completion;
• Collect numeric historical data about designs, processes, organizational structures for future use. Use the data to obtain an understanding of your process efficiency, to bid for funding for improvements and to benchmark against similar organizations!

Summarizing Quality Metrics

We need to teach the quantification of all critical quality aspects of a system; both up front in the requirements, and also in the evaluation of designs and architectures. We need to integrate these quality quantification into all other performance and cost quantifications. All qualities are quantifiable! That enables us to think like engineers. That simple fact needs to be a part of our culture. It is not yet the case.

Concepts

In spite of many attempts by IEEE and INCOSE to standardize or define technical terminology, most engineers seem to

1. use a personal and seemingly accidental definition of a word
2. not be aware that almost all other project participants define the term differently
3. not have reasonably clear and useful concepts of the most fundamental systems engineering concepts such as requirement, design, architecture, measure.

I have personally given up hoping that people can agree on the meaning of words. It is not the words that are critical, it is the concept definitions that we need to learn. We can then assign words or symbols to the concepts, in order to reference them. We can declare the set of words we use to reference concepts, in a paper, a book or slides.

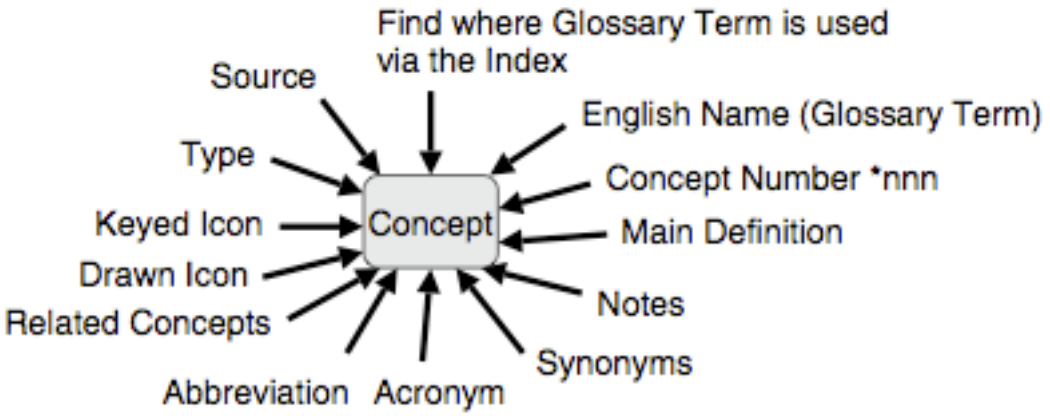

Illustration: I advocate detaching the concept definition from a variety of devices that can be used to reference the concept. Source, CE page 323

My primary concerns are that:

1. *we do not have a rich enough set of concepts: we need to distinguish between many types of requirements, many types of designs, many types of constraints – and much more.*
2. *We use words with no agreed meaning, as though others would know what we mean*

3. Our terms are not clearly aligned with our work processes

Goal	Concept *109

A goal is a primary numeric target level of performance. An implication of a Goal specification is that there is, or will be, a commitment to deliver the Goal level (something not true of a Stretch or Wish target specification). Any commitment is based on a trade-off process, against other targets, and considering any constraints. The specified Goal level may need to go through a series of changes, as circumstances alter and are taken into consideration.

A specified Goal level will reasonably satisfy stakeholders. Going beyond the goal, at the cost of additional resources, is not considered necessary or profitable – even though it may have some value to do so.

A Goal parameter is used to specify a performance target for a scalar attribute.

Example: Defining concept number 109, in the book referenced with the term 'Goal'. Source CE, page 366

Full personal Glossary:
http://www.gilb.com/tiki-download_file.php?fileId=25

I would suggest that we need to define our key concepts much better. Then we need to teach a decent set of the concepts systematically. We need to teach them in the context of practical processes for systems engineering. We also need to teach people to know when they need to define their local use of terms in their specifications. We have consistently measured in requirements specifications over 50 unclear or ambiguous words per page!

Processes

Defined processes can encapsulate experience and wisdom, perhaps good practices. They are one way to store the wisdom and to share it with others.

I would like to imagine that there would be some way of evaluating the properties of processes, so that we could distinguish useful processes from less useful processes.

We don't seem to be there yet. "Further research seems necessary".

Procedure

P1: Ensure that you have derived an elementary attribute (from a complex requirement), and that you are not trying to use a complex requirement, which needs decomposition into its elementary attributes. (Trying to find a single Scale for a complex (multi-Scale) requirement doesn't work well. It is usually the cause of trouble when people fail to find a suitable Scale.)

If you find you do indeed have a complex requirement, then decompose it and try to find Scales for its components. You might well find that further (second-level and more) decomposition is required!

P2: Ensure that the elementary attribute that you are developing a Scale for has a suitable tag and a Gist or Ambition

parameter that adequately describes the concept in outline terms.

P3: Using the Gist or Ambition, analyze how a 'change' of degree in the scalar attribute level would be expressed. What would a user experience or perceive? For some examples, see Table 5.1, 'Examples of Scales of Measure'.

Sometimes you can keep things simple, and 'make do', by controlling the details at a higher level of abstraction:

. by deciding to use one dominant Scale only, and consciously ignoring the potential other scales.

. by aggregating several scales of measure to express one summary scale of measure.

. by defining a complex attribute as the 'set' of other Scales and definitions.

P4: Specify the critical [time, place, event] qualifiers to express different benchmarks, constraints and target levels.

P5: If there is no appropriate standard Meter (or test), start working on a Meter. Try to imagine a practical way to measure things along the Scale, or at least sketch one. Try thinking about any measures that are currently being carried out (this could even help you start developing ideas for scales of measure). Also, think about whether any current system could be modified, or have its settings changed, to perform additional measurement.

P6: Try out the Scale. Define some reference points from the past (benchmarks) and then, on the basis of benchmarks, specify future requirements (targets and constraints).

P7: Repeat this process until you are satisfied with the result. Try to get approval for your Scale from some of the stakeholders. Does it quantify what they really care about?

P8: Consider putting embedded parameters into the Scale definition.

Rationale: To enable a Scale to be reused both within a project and in other projects.

EXAMPLE Scale: Time needed to do defined [Task] by defined [User] in defined [Environment].

Goal [Task

Get Number, User <Novice>, Environment <Noisy>]: 10 minutes.

P9: Once you have developed a useful Scale, ensure it is made available for others to use (on your intranet, or a web site, or in course materials, or your 'personal' glossary of Scales2). Offer the Scale to the owner of the 'Scales' library within your organization.

Process Example: A general process for defining scales of measure for quality variables. Source CE page 150-1.

References

CE: Gilb, Tom, Competitive Engineering, A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage, ISBN 0750665076, 2005, Publisher: Elsevier Butterworth-Heinemann.
http://homepage.mac.com/tomgilb/filechute/%20%20Gilb%20Competitive%20Engineering%20Book%20copy.pdf (only for SEMAT activists!)
http://www.gilb.com/tiki-list_file_gallery.php?galleryId=16 (2 freely downloadable sample chapters)

One hundred principles are explicitly stated in CE. Over 650 Concepts are defined in the book (180) and on our website (all 650+).
http://www.gilb.com/tiki-download_file.php?fileId=352 (a collection of these principles

Gilb.com: www.gilb.com. our website has a large number of free supporting papers (with many references of course), slides, book manuscripts, case studies and other artifacts which would help the reader go into more depth. The site contains a concept glossary of over 650 concepts.
Downloads: http://www.gilb.com/downloads

Gilb88: Tom Gilb, "Principles of Software Engineering Management", Addison-Wesley, 1988. About 124 principles are explicitly stated.

Jevons: W. Stanley Jevons (1835-1882) "The principles of science : a treatise on logic and scientific method", Published 1879. Macmillan and Dover Edition (1960s), 786 pages.

MIT05:
cipd2.mit.edu/public/MIT%20ES_Council_Fall2005.ppt
MIT Engineering Systems: Research and Education in Systems

Presented By Dr. Daniel Hastings *Massachusetts Institute of Technology*
December, 2005. Referenced in 2005 INCOSE Orlando Keynote.

BIOGRAPHY

Tom Gilb is an international consultant, teacher and author. His 9th book is '**Competitive Engineering**: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage' (January 2005 Publication, Elsevier) which is a definition of the planning language 'Planguage'.

He works with major multinationals such as Bosch, Qualcomm, HP, IBM, Nokia, Ericsson, Motorola, US DOD, UK MOD, Symbian, Philips, BAe, Intel, Citigroup and many others. See www.Gilb.com for much more detail, and free publications on Planguage.

His **Planguage** is a major innovation in systems engineering planning. It has been **adopted** by several major multinationals

Contact: Tom @ Gilb . c o m

3. Undergraduate Basics MASTER (131.07 Kb)

You can download this file using: http://www.gilb.com/community/tiki-download_file.php?fileId=98

05/12/2009 01:00