System Engineering, for a Cognitive Sciences Approach

Thomas Peugeot, CSEP

Airbus Defence & Space
Metapole,
1, Boulevard Jean Moulin, 78990 Elancourt Cedex

Abstract. This article proposes to throw some light on the strong relationship between Cognitive Science and System Engineering. In the same manner that Herbert Simon justified hierarchical organizations with the argument that individuals had "bounded rationality", this article tries to show that most of our System Engineering processes are techniques to compensate for our cognitive boundaries. Practical implications for SE justification and SE education are discussed.

1 Introduction

Be prepared for the truth: despite 6 million years of fast paced growth, our brain is still a limited engine. The brain cannot hold more than 50 000 patterns; the brain cannot ingest more than 15 concepts per days; the brain cannot manipulate more than 7 ideas simultaneously and the brain cannot stop from flying tactically to the first solution instead of first considering strategically all available venues.

Those 4 cognitive rules were proposed by Alan Newell in "Unified theories of cognition" (UTC, ref. 1). This article proposes to explain some System Engineering process with Newell's rules. In the same manner that Herbert Simon justified hierarchical organizations with the argument that individuals had "bounded rationality", this article tries to show that most of our System Engineering processes are best practices both to compensate for our individual cognitive boundaries and to scale, with the number of talents involved, our ability to develop systems.

2 Unified Theories of Cognition

UTC was published by Newell in 1991 (notice that Newell was knowledgeable about system engineering since he was a fellow of Herbert Simon, see ref 9). In UTC, Newell proposes a fruitful theory of cognition. For instance, at Airbus Defence & Space, our human behavior simulations are influenced by Newell's models (see ref 4 for an example).

Here, we are not interested in Newell's response functions. We are interested in the 4 rules that Newell reminds at the beginning of the book. Those rules might not be aligned with the latest cognitive sciences theories but they are enough for our argument.

Max processing

7 concepts

Rule 1: “7 +/- 2”. This rule is the most well known. 7 +/- 2 is the amount of concepts that can be manipulated simultaneously by an individual. For instance, it is known that a map should not have more than 7 different colors.

Max storage

50 000 concepts

Rule 2: 50 000 concepts per individuals is the amount of concepts an individual has in his brain store. For instance, when you ask a software engineer a question about his program, it is a rule of thumb that he can answer immediately if his program is less than 50 kSLOC (thousands lines of source codes). Above this threshold, he might have to go back to his design documentation or dive back into the code.

Max intake

15 concepts / day

Rule 3: 15 concepts can be ingested per day. Newell derives this quantity by measuring the learning process of a chess player. On average, a chess player can learn 15 "positions" per day. This is somehow related to the 50 000 limits because it takes on average 10 years of training to become a chess master (15 concepts * 360 days * 10 years is 54 000 concepts).
Rule 4: Individuals search "depth first". This is explained later.

3 System Engineering process revisited

This section revisits some process of the INCOSE handbook (ref 5). For each process, one or more Newell's rules are suggested.

3.1 CONOPS and FBS

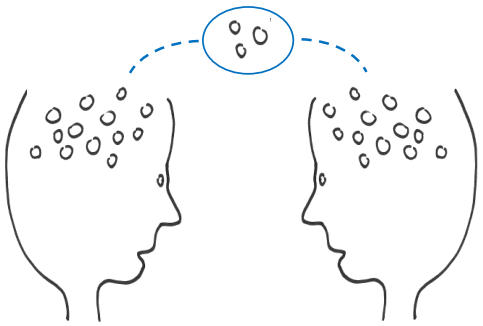
When System Engineers draft CONOPS, it is, through stories, to describe the system bit by bit. A rule of thumb is that a good CONOPS should be as interesting to read as a novel. With cognitive sciences words, a CONOPS should be an ergonomic device for pouring concepts into the mind of the reader.

From a completed CONOPS, System Engineers inventory all the functions elicited in the CONOPS and assemble them in an abstract hierarchical tree named the functional breakdown structure (FBS). In order to do that, System Engineers create abstract nodes that assemble functions and abstract nodes that assemble other abstract nodes. The top abstract nodes are the main functions of the system.

This abstract hierarchy makes it easier to understand the system in a top down view. The cognitive science justification is that our brain cannot manipulate more than 7 ideas simultaneously. Vertical hierarchies are fit for our limited intellectual span.

3.2 Requirement elicitation

When eliciting requirements for a system, two individuals coming from different disciplines try to contract on shared abstractions. One individual is the stakeholders agent. He represents the discipline that operates the system. The other individual is the designer. He represents the discipline that develops the system.



Concepts overlap

From 0 to 50 000

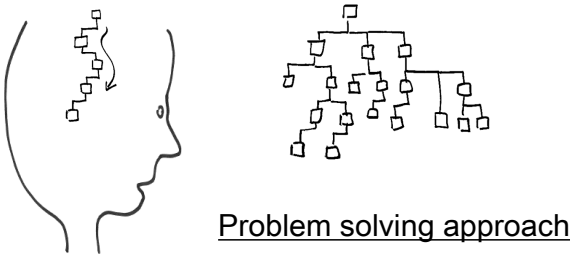
What effort shall we expect? Let us say we open the brain of both individuals, map both sets of 50 000 concepts and measure the overlap between both maps. This overlap may be large or thin. If it is large, a small effort is needed by the two individuals to elicit requirements (a lot will be implicit). If it is thin, misunderstanding might happen. The thinner, the harder both individuals have to explicit common concepts to create requirements.

3.3 Architecture process

In some cases, finding the best architecture is done with the help of mathematical tools such as operational research. Here, we are interested in cases when the architecture decision is driven by human cognition alone.

Since architecture is the most creative and diverse part of the System Engineering processes, it is preposterous to propose generalities. Yet, the article tries to relate one rule of thumb for system architects (the “no fly to solution” rule) to one rule elicited by Newell (the “depth first” rule).

According to Newell, when a brain is looking for a solution, it balances between two perpendicular directions, the vertical “depth first” direction and the lateral direction.



Problem solving approach

Depth first

"Depth first", which is the most natural tendency, derives of Newell's observation of chess players. For every move, a chess player anticipates was will be the opponent response. From this hypothetical answer, he devises his own answering move. This problem is usually formalized as a tree exploration. Some chess master can explore up to 10 moves down the tree.

Newell notices that good players tend to explore the tree in "depth first" rather than by layers. A chess master explores many branches of the tree in sequence. He then creates a global view of the tree to choose the branch of the tree with the most interesting outcome. In other words, he explores tactically before elaborating a strategic picture.

In System Engineering, selecting the first available architecture is not recommended (the "no fly to solution" rule). INCOSE commends choosing choice criteria, then construe alternatives and then evaluate all alternatives against all criteria.

Cognitive sciences might suggest a different perspective. It is necessary to go deep to have insights into the architecture strategy. In other words, some selection criteria will not be present at the start, they will surface along the design exploration (we see later how recent trends in technical management are more aware of this fact).

3.4 Interface design

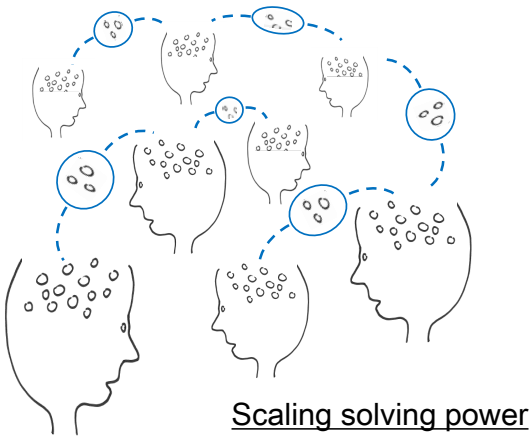
Architecture design decomposes a system into sub-systems with interfaces. When engineers design interfaces, they mitigate the integration risk.

Let us take two different specialty engineers who have to design an interface. Both need to share some of their 50 000 concepts. With requirement elicitation, we have already seen this situation where individuals need to share concepts. Instead of requirement elicitation, interface design is not an exercise in abstraction. Indeed, an interface has to be as detailed as possible to mitigate the integration risk between components.

However, in software engineering, a "good" interface design is an interface that aims at making the system across the interface the most abstract possible. A good interface limits the complexity for the interface users (there brain have to consume less concepts). For instance, the High Level Architecture (HLA) standard used in distributed simulation exhibits a "double abstraction barrier" to forbid participants to known too much about the other participants (ref 2).

3.5 Weaving abstract hierarchies together

Thanks to requirement elicitation, architecture and interface design, System Engineers can create Product Breakdown Structures (PBS). A PBS enables different teams to work in parallel on different part of the system. A PBS scales the engineering power with the number and diversity of contributing talents.



When System Engineers create PBS, it is to reflect the integration strategy. A PBS is one amongst many ways of putting system elements into abstract hierarchies, for instances:

- An operational expert categorizes people. He ends up describing an organizational breakdown structure (OBS).
- A roll-out/ILS team is interested in how to deploy and maintain the system. It devises a logistical breakdown structure (LBS).
- A Manager is interested in how the work will be done. He formalizes work breakdown structures (WBS).
- A security expert devises red area, black area, DMZ, trusted areas. He construes security breakdown structures (SBS).
- A financier relates work and procurement expenditures to his cost breakdown structure (CBS).

System Engineers trade is to weave those abstract representations of the system. A System Engineering Management Plan (SEMP) shall describe ways for teams to reach consistent breakdown structures with fluidity. For instance, at Airbus Defense & Space, our Model Based System Engineering (MBSE) tools are tailored to deal with overlapping hierarchies of data, including scheduling data.

4 Alternative approaches to technical management

In architecture & decision processes, INCOSE states that you should first gather all necessary information and then make the design/decision (see above).

Some other architecture/decision processes that have received attentions in recent years are a bit different, and perhaps more aware of our cognitive constraints. Toyota's Set Based Concurrent Engineering (SBCE, ref 3) and Lean Startup (ref 6, 7) are such exercises in architecture/decision process. Both surmise that some pieces of information necessary for decision are not present upfront, they come along the development process.

4.1 Set Based Concurrent Engineering

Toyota's SBCE decision process entertains more than one engineering solution down the development cycle. As stated by Sobek:

> In developing a vehicle's styling, Toyota makes more […] models than most competitors do. […] Simultaneous with the development of the two to three full-scale models, Toyota engineers develop structural plans for multiple styling design ideas and analyze them for manufacturability.

Sure, SBCE is more work upfront but less rework is to be expected downward. Indeed, SBCE is more robust to late problems discovery because more solutions are kept alive. Sobek contrasts SBCE with point based concurrent engineering where one solution is selected early and finding problems late leads to rework up the value chain. For instance at Airbus, at Preliminary Design Review (PDR), the usage is to have only one solution to be reviewed.

4.2 Lean startup

Another example is the "lean startup" agile approach. In SE classical view, people either plan or execute the plan. If something goes wrong, it is either the plan or the execution that was wrong. Lean startup suggests that when a system interacts with an environment that is complex and not controlled, the plan/execute model maybe not optimal.

When designing social web applications (the focus of Ries's work), the system interacts with thousands of individuals/clients. Then a simple *configuration management* issue kicks in. It is impossible to inventory and control the 50 000 concepts of the thousands of individuals that will use the system.

To fix the "brain configuration control" issue, Ries proposes to perform more but shorter "plan/execute" cycles. In a "lean startup", the business concept under study is declined in assumptions (let us say 20 assumptions for a typical concept). At the beginning of each cycle, a set of those assumptions is selected (let us say 5 assumptions). The objective of the cycle is to develop parts of the concept that enable the team to measure if those assumptions are true. If, at the end of the cycle, some as-

sumptions are wrong, the lean startup must be prepared to “pivot” and elicit a modified concept with new assumptions (that again need to be rigorously tested).

According to Ries, a Lean Startup measures progress not in units of work achieved but in units of verified learning.

5 Practical conclusions

5.1 Design of requirement documents

As System Engineers, we are interested in SMART requirements.

However, cognitive science hints that there is also a challenge elsewhere, the design of requirement documents in complex systems.

First, a requirement is a shared concept between two brains. A requirement is like a prosthesis between the brain of the stakeholder agent and the brain of the designer. Like any prosthesis, a requirement needs to be “adapted” to its users.

Second, a requirement is not alone. It is formalized in a document containing a set of requirements. Ideally, this document shall be approved by two persons only: the stakeholder agent and the designer. Therefore, a requirement document shall fit both the brain of the stakeholder agent and the brain of the designer.

In some System of Systems design, such as Air Traffic Management, stakeholders (operational experts) and designers (system architects “at system level”) can share some requirements but it is difficult to group requirements into documents that span more than one system. Indeed, if a system is complex, a system designer can only elicit requirements with a span of one or two systems at best (cf. the “50 000 concepts” rule).

How to elicit requirements that have a span across all the systems into a single document ?

The solution to this cognitive constraint is to design requirements in two layers: a “system of systems” top layer and a “system” layer. The “system of systems” requirements are more abstract than “system” requirements. The “system of systems” requirements cannot be used directly for the design of system but they can fit into one document.

The design challenge is to find system engineers that can work sandwiched between both layers. Seen from operational experts, they are designers who can elicit requirements in a document with a span of all systems. Seen from the system architects “at system level”, they are stakeholders who can elicit requirements, in the “system” layer, for each systems. The design challenge is a logistic challenge: to find people with a brain that have a good balance of 50 000 concepts, for instance 25 000 concepts shared with operational experts and 25 000 concepts shared with system designer.

5.2 Evaluating SE needed overhead

Another problem is to assess the need for System Engineering artifacts. Do we need to formalize an interface? Is a simulation needed to demonstrate the functional behavior of the system?

Let us take simulations. They can be expensive but they provide visual cues, the path with the highest throughput to the concept store in our brain. The case for simulation shall sometimes be a cognitive case. Are the stakeholder already experts in the operational view (weak case) or are they novice (strong case).

5.3 Teaching the INCOSE handbook

To pass an INCOSE certification, one rote learns 25 different processes. The global coherence is difficult to grasp. Clearly, outputs of some process are inputs to other process. Yet, it can be rather difficult to derive a general underlying argument to justify all processes.

The INCOSE backdrop philosophy is that it is good to put 10 to 15 % SE overhead to have a complex project succeed. This philosophy is pragmatic, it is based on regular observation of patterns in past complex projects but it is not a generative argument. This article suggests that Cognitive Science could be mentioned in SE handbooks introduction. It could answers “why do we need SE?” before going to the “what is SE?”.

An interdisciplinary point of view toward system engineering can be insightful. In 1984, Perrow, a sociologist, published his “normal accident theory” (ref 10) that challenges the traditional risk approach to complex system engineering. The book second chapter is titled: “Nuclear Power as a High-Risk system: Why we have not had more Three Miles Islands – but will soon”. It is, in retrospect, an insightful point of view.

6 Conclusion

If cognition is the mother of all constraints in System Engineering, it is time to measure cognition constraints consistently. In many occurrences, System Engineers rely on their experience and their “soft skills” to assess people constraints.

Between cognitive science and system engineering, an interdisciplinary academic effort could provide us with bespoke tools to assist system engineers in their daily work. For instance, we could evaluate more consistently the cognitive overlap between two individuals.

Academics sometimes testify that teaching SE to students can be challenging. As stated by Simon himself, the subject is abstract and a bit confusing (“cooky booky”). Simon compares here SE to other engineering discipline based on hard sciences (ref 8). If cognitive scientists could provide system engineers with quantitative tools, SE could become a more “normal” discipline.

7 Aknowledgment

The author acknowledges helpful and good humored discussions with Dominique Ernadote, Jean-Victor Noël-Betemps, Bruno Carron and Jean-Luc Wippler.

8 Bibliography

1. Newell, A. (1994). *Unified theories of cognition*. Harvard University Press.
2. Dahman, J., Ponikvar, D. R., & Lutz, R. (1996, March). HLA federation development and execution process. In *15th Workshop on Standards for the Interoperability of Distributed Simulations* (pp. 327-335).
3. Sobek, D. K., Ward, A. C., & Liker, J. K. (1999). Toyota's principles of set-based concurrent engineering. *Sloan management review*, *40*(2), 67-84.
4. Gautreau & al. (2013). Lessons learned from NMSG-085 CIG Land Operation demonstration. *Spring Simulation Interoperability Workshop,* paper 13S-SIW-031.
5. INCOSE System Engineering Handbook V3.2: A Guide for System Life Cycle Processes and Activities.
6. Ries, E. (2011). The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses. Random House LLC.
7. Blank, S. (2013). Why the lean start-up changes everything. *Harvard Business Review*, *91*(5), 63-72.
8. Simon, H. A. (1996). *The sciences of the artificial*. MIT press.
9. Newell, A., & Simon, H. A. (1972). *Human problem solving* (Vol. 104, No. 9). Englewood Cliffs, NJ: Prentice-Hall.
10. Perrow, C. (2011). Normal Accidents: Living with High Risk Technologies (Updated). Princeton University Press.