

Notes on *The Sciences of the Artificial*

Adapted from a shorter document written for course 17-652 (Deciding What to Design)¹

Ali Almassawi
December 29, 2005

1 Introduction

The Sciences of the Artificial is an enlightening work, despite its age, that contains within it Herbert Simon's observations on the nature of things in the natural and artificial worlds². Simon, whose writing style is uniquely engaging, analytically discusses topics such as complexity and symbolic systems, where they lie with respect to the two previously mentioned worlds and how they are affected by the surrounding environment.

This short document discusses a limited subset of Simon's models and processes that I found to be the most interesting. An attempt is made to keep it as focused as possible on things that, if incorporated into software design, can immediately provide benefits to a project. The document is by no means prescriptive or all-encompassing.

2 Reducing redundancy in complex systems with a hierarchic model

Any system that is made up of parts is naturally complex. Complexity may be perceived, superficially at least, as being chaotic³. In order to organize this complexity, Simon proposes using a hierarchic model in which a complex system is recursively decomposed into subsystems until an elementary level is reached. An elementary level is a level where a subsystem can no longer be reduced. The criterion for reduction is functionality; subsystems are decomposed functionally, with similarities being collapsed together to achieve hierarchic levels.

Take the human body for example. The human body has ten systems within it that function simultaneously as a set of organs; each organ can be decomposed into tissues and each tissue can be decomposed into cells. Here, the structure of a complex and apparently chaotic system is organized and described nicely and succinctly using a hierarchic model. The progression from organs to cells is naturally one of high abstraction, or highly complex, to low abstraction, or simple.

A theme that runs through Simon's discussion of the hierarchic model is that a complex system can be organized and made understandable by identifying and then reducing its redundant elements. In fact a dominant idea in decomposition is to collapse similarities together.

3 The property of near decomposability

Near decomposability is according to Simon a property of complex systems. The idea of near decomposability is two-fold. First, it says that the interactions *between* subsystems in a complex system are weaker than the interactions *within* them. From Science, one can think of how molecular forces are weaker than nuclear forces, and from the corporate world, one can think of how interactions between employees in a department are much more frequent than interactions between employees of different departments. Second, it says that each subsystem in a decomposed system is *almost* autonomous, meaning that each is independently functional and useful, but still provides value to the overall system by

¹ Course 17-652 is taught by Mary Shaw, Jim Herbsleb and Dave Root at Carnegie Mellon.

² The artificial world is the world encompassing all things that are man-made.

³ In distinguishing between engineering and science, Simon says that scientists choose to look through chaos and extract from it patterns that allow them to make sense of things. Similarly, he says: "Complexity, correctly viewed, is only a mask for simplicity; [natural science finds] pattern in apparent chaos".

maintaining a weak connection with it. In the object-oriented world, the idea of near decomposability can be observed in efficient OO-models where loose coupling, encapsulation and methods are effectively used to achieve modularity.

Simon generalizes the concept of near decomposability into a hypothesis he calls the empty world hypothesis. He says that a seemingly empty world can be accurately described by describing it only at the desired level of abstraction, without saying anything about its subsystems down to the elementary level.

Consider the following example: a monkey is given a paint brush, some paint and pieces of paper and is left to do as he⁴ pleases while a psychologist observes from afar. What are the things that the psychologist may do to understand the monkey's behavior? Perhaps change the paint colors; change the size and shape of the brush; position a painter next to the monkey and have the former paint in front of him. These are all things that the psychologist varies in order to probe the monkey's internal system and better understand his behavior. Yet notice that our hypothetical psychologist at no point analyzes the neurological structure of the monkey's brain (not that he couldn't) or tries to understand how his anatomical components and their interactions affect his behavior. The tests are simply observations of how the monkey deals with his environment. The monkey's behavior can be accurately described without having to go to the trouble of analyzing it at a molecular level.

This example shows the empty world hypothesis in action. By picking a particular level of abstraction, i.e. by considering the monkey to be a simple behavioral machine⁵, one can learn all sorts of interesting facts about it without having to dig deep into its internal structure.

4 Yes, abstraction is a wonderful concept

Abstraction is a wonderful thing that humans use all the time. A lot of the time it is used subconsciously, perhaps because that is how the brain is organized [Simon]. A smart person learns to use abstraction effectively in order to get things done more efficiently. A not-so-smart person either doesn't use it at all, or uses it incorrectly. Mary Shaw nicely sums up these two cases: "Abstraction is hiding detail. Good abstraction is hiding the right detail".

Consider the following example of two students who have been given a problem and asked to submit their solutions by the next day: The good student looks at the problem and identifies its parts. He then looks at those parts and if there is any value in breaking any of them down further, does so. Otherwise, he starts tackling each of them separately and finally makes a claim about the overall problem in the form of a solution. The bad student looks at the problem and breaks it into parts, each of which he also breaks into parts, etc. He drills each part down so deep that at the end of the process he looks up from under his helmet to see a pile of different-sized fragments around him. Where should he start? By the time he finishes tackling all those parts, either the deadline has passed or the descriptions of the parts are so superficial or unnecessarily detailed that an accurate solution can't be arrived at.

The good engineer does the same thing when designing an artifact. Going from the abstract to the concrete gives him more flexibility in making design decisions and, more importantly, affords completeness and accuracy. Abstraction is a key concept in computer science.

⁴ Masculine pronouns, with the exception of those referring to Simon, are meant to encompass both men and women throughout the document

⁵ In his chapter on the psychology of thinking, Simon states the following hypothesis: "{An ant, a man}, viewed as a behaving system, is quite simple. The apparent complexity of its behavior over time is largely a reflection of the complexity of the environment in which it finds itself."

5 The sciences of design

To begin with, what is a science of anything? A science is a set of activities and principles that one uses to describe phenomena. A definition that Simon gives when talking about it in the context of design is that it's a set of tools that allow one to change existing situations into preferred ones. Take the field of computer science for example. If one were to ask: "What do I need to know to be a computer scientist?" the answer would be "artificial intelligence, data structures, algorithms, operating systems, kernels, etc." Those are the sciences of the field.

Elsewhere, Simon defines the sciences of design as simply the proper study of mankind. He then lays out a curriculum containing activities, principles and tools that, if followed, can turn a student into a designer. Some of those activities, principles and tools are briefly described below:

- Understanding the user's psychology, cognitive behavior and problem-solving patterns. Accurately understanding the end-user and being able to describe their behavior in a user model is an effective way of determining how the program model will end up being designed. Recall that one of the responsibilities of a designer is to bring the user model and the program model in line. These sciences relate to some of the things that Donald [Norman] talks about in his elaborate discussions of mental models, affordances and the psychology of thinking, as well as contextual design's idea of co-designing with the end-user [Beyer].
- Determining the optimal design decision. Because resources are scarce, the designer needs to have tools at hand that allow him to decide between different designs. Business concepts such as utility theory and cost/benefit analysis are useful tools that can be used to quantify different design alternatives based on their values to different stakeholders.
- Decomposition. As discussed in previous sections, functional decomposition is a process that allows a complex system to be split up into chunks. Each chunk can then be independently designed, ranked and delegated.
- Representation. Different types of problems require that they be represented differently in order to provide value. Simon's criterion for effective representation is that the latter should make the solution transparent. He defines a taxonomy for deciding on the right representation: "if the problems relate to physical objects, they (or their solutions) can be represented by floor plans, engineering drawings, renderings, or three-dimensional models. Problems that have to do with actions can be attacked with flow charts and programs." Thankfully, popular languages like UML support both structural and behavioral diagrams.

6 How these sciences of design are applicable to software projects

All four of the sciences mentioned above are useful in today's software projects and can add value to their design process.

Understanding the end-user is a concept that has been emphasized with great enthusiasm in areas like contextual design. It is crucial that the team be familiar with the psychology and behavior of their end-user, or at least the portion of it that is significant to the team. One can look at case studies such as the one on the Microsoft Word 4.0 team and appreciate the dangers of not considering the end-user or involving them in the design. As contextual design suggests, a successful design requires involving the user in the design process to tease out subtle aspects of the end-user's behavior that might not be consciously apparent.

At some point, a development team will very likely have to decide between alternative designs. The ability to justify each decision is crucial and might in fact be requested by the paying customer. Simply relying on human judgment may not be a sensible thing. Among other things, the integrity of a design may be impacted if a design decision is made with no quantitative justification. Utility theory and conversion functions [Poladian] are two very practical quantification methods.

Accurate decomposition and representation of a complex system is crucial. Breaking a whole into parts and then analyzing the parts individually helps the development team at numerous levels. The increased granularity allows the team to understand the full extent of the problem more precisely. It also allows them to guarantee that quality attribute requirements hold in the final product. This is achieved because of the natural progression that is made from subsystems in a decomposed representation to modules in the code.

Here is a somewhat trivial decomposition of a software system that I'm currently working on:

- User-interface
- Graphics engine
- Rule-based and case-based reasoning engines
- Learning engine
- Databases

Considering each of these areas in near isolation yields a more understandable problem domain⁶. The engines represent different types of artificial logic, the databases define the software system's data model and the user-interface represents its presentation components. Each of these can still be broken down further. Once each subsystem is broken down to an elementary level, the project schedule becomes much more realistic and the task of delegating "work units" to team members becomes less risky.

7 Final remarks

All in all, the models and processes that Simon discusses are quite useful and practical despite the theoretical fashion in which they are presented. The fact that they are applicable to today's software projects is testimony to their significance and timelessness. Software developers will greatly benefit from at least adding Simon's set of sciences to their engineering toolkit and using them wisely when appropriate.

As for hierarchic decomposition, I think it is still a dominant way of designing software. Nevertheless, other design methods, like Michael Jackson's problem frames and UML's use-case diagrams, take a different approach towards decomposition. There it is more like parallel decomposition rather than hierarchic. I'd be interested in learning more about whether or not these approaches can be consolidated with Simon's hypotheses.

⁶ Note how this decomposition can be nicely fit into a Model-View-Controller architecture. Such paradigms rely on separating concerns at an architectural level, as does hierarchic decomposition.

8 References

- **Books**

[Beyer] Hugh Beyer and Karen Holtzblatt, Contextual Design: A Customer-Centered Approach to Systems Designs, Morgan Kaufmann, 1998.

[Norman] Donald Norman, The Design of Everyday Things, Basic Books, 2002.

[Simon] Herbert A. Simon, The Sciences of the Artificial 2nd edition, MIT Press, 1981.

- **Papers**

[Poladian] Vahe Poladian, Shawn Butler, Mary Shaw and David Garlan, Time is Not Money: The case for multi-dimensional accounting in value-based software engineering, 2003.