Abstract – We present a simple conceptual model of what constitutes complexity and simplicity in design engineering. At the core of the model are the three concepts of 1) scale (how many things are there), 2) diversity (how many different kinds of things are there), and 3) connectivity (how many relationships are there between things). The model distinguishes essential from accidental complexity (i.e., the complexity that we, engineers, add while designing), and intrinsic versus perceived complexity. The model also articulates the complexity of the thing (or system) we design or observe versus the complexity of the community around the system: its users, designers, manufacturers, sellers, other systems, etc.

This model is then used to articulate a set of heuristics to address complexity: reduce, hide, shrink, organize, explain, expose. Finally we open the toolkits of engineers in various disciplines to identify strategies, methods, or tools that they can use to address complexity: design principles, guidelines, design methods, patterns, tactics, frameworks, etc. Approaches such as modeling, abstraction, partitioning can then be described in terms of our key concepts and heuristics; e.g., “abstraction reduces perceived complexity”.

This conceptual model helps engineering students to better reflect on their practices of design, and how these practices vary across disciplines. It also provides a more systematic approach to answering the never ending question: “how can you make this simpler?”

Keywords: Complexity; simplicity; design; system engineering; complex systems; patterns; tactics; conceptual model; engineering education.

“Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better.” — E.W. Dijkstra

1. INTRODUCTION

Gather ten engineers around a table or a whiteboard, and while they may agree that a given system “is complex”, they may not come very rapidly to a consensus as to what makes it complex, or simple. They will likely paraphrase U.S. Justice Potter Stewart on pornography: “Complexity: I may not be able to define it, but I know it when I see it.”

If they are experienced engineers, they will likely know a few ways to manage or even reduce this complexity; but they may not necessarily be conscious of why and how these ways succeed in reducing complexity. And more often than not, even experienced designers tend to equate complex with “bad”, or confuse intrinsic complexity with perceived or apparent complexity.

With engineering students, the situation is worse, as they often lack exposure to sufficiently complex systems to be able to articulate any difference. The systems they are most often faced with in labs and projects are actually pretty simple. But the instructors often leave them with some hand-waving response when challenged to define complexity, or simplicity.

In this paper, we’ll attempt to give (simple?) definitions of both complexity and simplicity, and build a small conceptual model of complexity and simplicity. We will then articulate a few heuristics to make systems simpler. We then look into the toolkit of the system’s engineers for some age old recipes used to tackle complexity, and re-examine them in the light of this conceptual model.

There are many forms of complex systems, especially biological or social ones. However in this paper we will look at the complex systems that humans have designed and built, the kinds of systems that Herbert Simon spoke about in “the sciences of the artificial” [1]. Typical systems in that category that are really complex are: the world-wide telephone system, the Internet, the power grid. Some of these systems are actually “systems of systems”, or “ultra-large scale (ULS) systems” [2]. We also will not look at system theory, control theory and the vast and fascinating domain of complex adaptive systems (CAS).
2. COMPLEXITY AND SIMPLICITY

2.1. Complexity

Complexity is characterized by three main attributes: scale, diversity, and connectivity [3]. Scale: how many “things” are there in the system? Diversity: how many different kinds of “things” are there? Connectivity: how many different relationships are there between “things”.

When approaching a system, how many elements or parts we can identify is the first thing that we give us a sense of its complexity. But rapidly we realize that the mere number of elements is not so determinant, as the diversity of these elements. A system made of 5,000 elements all of the same type will feel much simple than a system made of 100 elements of 20 different kinds. While in the first case, going from 5,000 elements to 1 million will not feel much more complex, going fro 20 to 30 different kinds will visibly increase complexity. Digging further, we soon realize the real complexity comes when we realize that these elements or parts are not just sitting there together, but are interconnected: they influence each other in many different ways, some immediate and visible, some deferred, some completely invisible [4]. Here also the profusion of different kinds of interconnection, often driven by the number of different kinds of elements will increase complexity.

One crucial kind of interconnections between elements is what constitute feedback loops, which in turn is the basis for much of the complex systems behavior, behind evolution, and adaptation.

2.2. Simplicity

Conversely, if we assume that simplicity is the exact opposite of complexity, then it may be characterized by parsimony, homogeneity, uniformity, and regularity of organization.

2.3. Essential vs. accidental complexity

Another important consideration is essential complexity versus accidental complexity. Often many of the elements or types of elements or relationships between elements are not an essential part of the system; they have been introduced by the designers for a wide range of reasons. If you were to open a telephone switch, you’d find that 90% of the elements in there have little to do with putting you in communication with Grandma.

2.4. Intrinsic vs. perceived complexity

Finally, we should distinguish between intrinsic complexity and perceived complexity. Perceived complexity varies greatly from the perspective of the stakeholders, both of the actual system or the construction of the system. The perceived complexity is often related to the fact that many relationships in the system (or in the development organization) are hidden from the stakeholder or observer.

3. HEURISTICS TO ADDRESS COMPLEXITY

“Perfection is achieved not when there is nothing more to add, but rather when there is nothing more to take away.” —Antoine de Saint-Exupéry [5]

3.1. Laws of simplicity

An obvious approach to simplicity is to define it as the exact counterpoint to complexity: a system that handles few things, few types of things, or where there are few relationships between elements will immediately feel much simpler. John Maeda has expressed this well in his “laws of simplicity” [6].

In this (very simple and short) book, written when he was at the MIT Media Lab, he has proposed a set of simple heuristics for designing simple systems: reduce, shrink, hide, organize, explain. Reduce directly addresses our three key metrics of complexity, though when hit by the limit of essential, intrinsic complexity we have to resort to hide and shrink, while organize may provide an operational approach to reducing and hiding.

3.2. Reduce

“The simplest way to achieve simplicity is through thoughtful reduction” (Maeda’s Law #1). Reducing the number of things is rarely an option, but limiting the profusion of different kinds of things has been achieved by various means, such as class hierarchies. For example: on a map, let us not represent the rivers and the power lines if they do not bring any value. Or let us not distinguish streets, roads, and highways. Or let us not describe the topographical relief. “Some things can never be made simple” (Maeda’s Law #9) is the observation that there is an intrinsic lower limit of essential complexity that we cannot cross. From this boundary, designers are left to tackle accidental and perceived complexity. When reducing is no longer possible, they can hide or shrink or explain.

3.3. Hide

Hiding elements does not eliminate them, it only removes them in some select viewpoints, for some stakeholders. Hiding has been exploited in various ways for a few decades by software designers, in the form of encapsulation [7].

3.4. Shrink

Shrinking is not quite the same as hiding; it occurs when you expose a simplified view of an element or relationship. For example, using a circle on a map to
represent a city. Shrinking allows to control perceived complexity by guiding the attention of the observer of a system on fewer elements, or balancing various elements.

### 3.5. Organize

“Organization makes a system of many appear fewer” (Maeda’s Law #2). When there is nothing that we can eliminate or hide, organizing introduces a regularity that allows the observer to understand the system and to infer some of its properties. This works when the organizing mechanism is easily recognizable—a pattern, for example.

### 3.6. Explain, expose

Finally the perceived complexity may often be reduced by explaining or exposing some aspects of the system, for example making some hidden relationships visible; “knowledge makes everything simpler” (Maeda’s Law #4). Expose may sounds to contradict Reduce (3.2) or Hide (3.3), but actually it addresses the reduction of perceived complexity, where the actual intrinsic complexity cannot be mitigated: if you cannot eliminate the elements, the connections, it is sometimes better to make them visible.

“Fools ignore complexity; pragmatists suffer it; experts avoid it; geniuses remove it.” —Alan Perlis

### 4. AN EXAMPLE: THE INTERNET

The Internet as we know it today exhibits many of the aspects described above. The Internet is one of the most complex man-made system ever created. It contains:

- very large numbers of things (scale),
- large numbers of type of things (diversity): servers, routers, nodes, addresses, protocols, etc., and
- large numbers of relationships (connectivity): names to nodes, nodes to protocols, ...

How has this complexity been managed, to render the Internet usable? Let us consider the Internet in light of the heuristics just presented.

**Reduce:** there has been no attempt to reduce the total number of things. However, there has been strict control on the number of types of things: The W3C is very conservative in the introduction of new elements that may affect some property of the Internet. This constraint inevitably leads to limits on the number of relationships between things. And at the same time is increases uniformity in the system, and in turn less perceived complexity.

**Hide:** in the Internet, many lower level protocols are hidden, and most of the physical infrastructure is hidden. As users and developers, we deal with higher-level abstractions—nodes, servers, etc.

**Shrink:** this occurs through abstraction and hierarchy in various parts of the Internet: for example, google.com is a shrunk representation of many thousands of IP addresses and hundreds of thousands of computing nodes.

**Organize:** the Internet has employed patterns extensively to tame complexity. Two of the most common patterns used are peer-to-peer (P2P) and client-server. Each of these allows us to comprehend groups of nodes (that may number in the millions) in a simple and consistent way.

The Internet is also an example of the duality between the system (the Internet itself) and its community: all the users, suppliers, regulators,... with very complex connectivity between elements of the system, and entities in the community.

### 5. THE ENGINEER’S TOOLKIT

As engineers, we have several sorts of tools to address complexity. Although they vary considerably depending on our respective engineering discipline (mine is that of software-intensive systems), they more or less all fall in one of the following four categories.

#### 5.1. Approaches to handle knowledge

The designer has many different ways to manage knowledge, either generic knowledge of her domain, or specific knowledge associated to a certain system: design methods, modeling techniques, using mathematical or physical avenues, drafting, CAD systems, simulation tools, ontologies, etc. Here also fit guidelines, handbooks, and standards of our discipline.

#### 5.2. Design principles

Principles capture, in a synthetic form, laws or guidelines that the designer should follow to achieve an effect, or to avoid a type of failure. They constitute condensed design knowledge. For example, “separation of concerns” guides one to reduce the number of relationships. “Abstraction” supports selectively hiding without reducing. The “minimal surprise” principle encourages the use of regular means of organizing domain concepts.

#### 5.3. Design patterns

“Humans use patterns to order the world and make sense of things in complex situations.” [8] Pioneered by Christopher Alexander [9], design patterns have been very useful to capture small and large elements of solutions. A pattern establishes a relationship between:

- a context: a recurring, common situation in the world that gives rise to a problem,
- a problem, appropriately generalized, that arises in the given context, and
Tactic: Nodes may be heterogeneous, sharing only a common communication protocol.

Tactic: Nodes may be nested (i.e., a hierarchy). In fact, nodes exhibit self-similar (fractal) structure.

Standard: Comply with both IETF standards for Reactive MANET Protocol (RMP) and Proactive MANET Protocol (PMP)

Framework: use the Common Manet Framework

6. THE SYSTEM AND ITS COMMUNITY

There is a whole facet of complex system design, which we have left aside until now. We have only considered the complexity of the system itself, the “system-under-design” (SUD). There is also the complexity of the community around the system: all the users, builders, designers, and maintainers of the system, what in some circles are called “the stakeholders”, parties that are actively involved in the system, or whose interests are affected by the system. This community also constitutes a complex system, which exhibits the same aspects of complexity as the system itself: this community may be large, made of many different kinds of stakeholders, with many kinds of relationships. Also, these two—the system and the community—are not independent: there are many relationships between elements and kinds in the system, and elements and kinds in the community.

The complexity of an engineering project is thus the combination of the system complexity and the community complexity, as described by Walker Royce in [14] (p. 210-211).

Here again we strive to reduce complexity by looking at ways to limit these relationships with the help of governance rules, access rights, and congruence between organization of the system and organization of the community, i.e., the degree of alignment between the structure of the system and the structure of its community [15]. Often the structure of a system reflects the structure of the organization that built it (Conway’s law).

The simultaneous growth of a system and its community is seen in IT technology with the emerging concept of ‘software ecosystems’ [16, 17] around certain tools, technologies or companies, such as the Amazon ecosystem, or the Facebook ecosystem.

6. TEACHING AND COMPLEXITY

The conceptual model presented here can be used very systematically when working with students on their design projects, in particular in capstone projects. The question is not “does it work?” but “how can we make this simpler?”

The canvass I use for reasoning about complexity and simplicity is the following:
7.1. Function, purpose

We start by identifying what are the functions or purposes of the system? Whom are these functions for? If more than one or two, let us rank them by order of decreasing importance.

This step is important because often students will list functions that are simple “nice-to-have”, or “me too”, or just there to show off their literacy.

7.2. Scale, Diversity and Connectivity

What are the elements the system is made of? What are the kinds of elements? Are they visible from outside of the system? How are they related to the functions? How do they contribute to each function? What are the connections or relationships between these elements? What is the impact of time on the connections? Does the system behaves based on history, on a stored internal state? What form does this internal state take? Data?

7.3. Essential vs. Accidental Complexity

What are the design elements that directly contribute to the functions you identified? What are the elements that you had to include to support your design, or to palliate to possible deficiencies or your design?

Does the end-user really want an error log? A tracing feature? Another login name and password?

7.4. Perceived vs. Intrinsic Complexity

What would make your system design look complex for other people? (Students are obviously usually very familiar with their own design, and it is somewhat hard to make them look at it from someone else’s perspective.) Would your grand-mother understand how to use it?

7.5. The System and its Community

Who are the stakeholders? Start from the primary users (identified earlier in step 7.1), extending to the designers, builders, maintainers, manufacturers, … to legislators, competitors,…. criminals (!) Who has something at stake? Who is affected by the system? Looking at this community as a system, we can apply the “scale, diversity, connectivity” step (7.2), but we have in general less control on the other attributes (7.3, 7.4).

We can now establish connections between elements of the system, and entities of the community. Who cares about what?

7.6. Looking through our toolkit

Once we have this complexity inventory, we can try to apply some of the heuristics: Reduce, Hide, Shrink, Organize, Explain/Expose to explicitly address the issue of complexity

We can in turn pull:
- design principles: should we increase modularity?
- or patterns and tactics: can we reduce complexity by layering the system?

We can take elements and ask: are they all necessary? how can we dispense of them? Is the distinction between these two kinds of elements really necessary? Shouldn’t we merge them, treat them alike? Reuse one for the other? Do we need both this file and a database? Can’t we store this information in the database?

For elements or for relationships: can we hide them, can we shrink them? We can exploit the connections between the system and its community: which stakeholders need to know about what aspects? Maybe the hiding can be selective? Or the abstraction could be different for beginner users, expert users, and maintenance personnel, for example.

Can we make the system easier to understand and use by exposing some relationships, making them explicit?

The criteria for picking and applying this or that principle, tactic, pattern, etc. are:
- does it reduce one of our core metrics: scale, diversity and connectivity (without increasing another one)
- does it lower perceived complexity? for whom?
- does it reduce accidental complexity? (akin to Value Analysis)

The process sketched here in steps 7.1 to 7.6 is iterative and incremental. It is based on whatever design material the students have produced so far: design dossier, blueprints, sketches, prototypes, etc. It may be seen as an instance of a “reflective conversation with the materials of a design situation” described by Donald Schön in The Reflective Practitioner [18, 19].

At the University of British Columbia (UBC), the model introduced in sections 2 and 3 is introduced in a final course on project management, in the segment on managing complexity [19, 20]. The process sketched in section 7 is used in interdisciplinary capstone projects, in particular the course offered jointly with the Sauder School of Business: New Venture Design [21].

8. CONCLUSION

To paraphrase Grady Booch in [20], “the primary challenge of every system designer is to engineer the illusion of simplicity in the face of essential complexity.”

The overall strategy to reducing complexity in design systems, both intrinsic and perceived complexity, can be introduced systematically while teaching design. The conceptual model of complexity and simplicity we’ve introduced, together with its heuristics and toolkit (appropriately specialized by domain) allow to define simple steps:

1) choosing the right set of design primitives, to cover all of the required system functionality, while keeping the number of such primitives as small as possible,
2) finding regular, systematic ways of assembling the primitives into larger, more complex aggregates, and
3) minimizing the number of different forms of interaction between the primitives, or aggregates, and keeping these interactions flexible and adaptable.

Design principles, guidelines, standards, patterns, tactics, methods and tools… are articulated in terms of the impact they have on a few core metrics of complexity, reducing profusion to achieve simplicity. But remember: “Everything should be made as simple as possible, but not simpler.” —Albert Einstein

Acknowledgements

My work as an educator in design at UBC is directly supported by an NSERC Chair in Design Engineering.

Many thanks to my colleagues from the IFIP WG2.10 on Software Architecture for their feedback on the original presentation on this topic, which led to interesting discussions with Prof. Rick Kazman (U. of Hawai‘i) and ultimately to a conference paper with him [13] related to the complexity of software-intensive systems.

References


[20] P. Kruchten, "The frog and the octopus-Experience teaching software project management to undergraduate engineering students," in Proc. of the 2nd annual conference of the Canadian Engineering Education Association, St. John's, NF, Canada, 2011, CEEA.