Toward a System Design Science

Joseph Sifakis

RiSD Laboratory, EPFL, Lausanne, Switzerland
joseph.sifakis@epfl.ch

1 About Design

Design is a universal concept. It links the immaterial world of concepts to the physical world. It is an essential area of human experience, expertise, and knowledge, which deals with our ability to mold our environment to satisfy material and spiritual needs.

Design has two different connotations. One is simply a plan or a pattern for assembling objects constituting a given artifact. The other is the creative process for devising plans or patterns and carrying them out to produce an artifact. For this paper we focus on the second interpretation. We are ultimately interested in putting design on a more scientific basis. Toward this end, we focus here on articulating a new structure for the design process, which we believe will support this goal.

We consider that design is the process that leads to an artifact meeting given requirements. The requirements include functional requirements describing the functionality provided by the artifact and extra-functional requirements dealing with the way in which resources are used for implementation and throughout the artifact's lifecycle.

Designers deal with two often antagonistic demands: 1) productivity, meaning cost-effectiveness; 2) correctness, meaning compliance to requirements. In pursuit of these demands, the design process moves through three stages. The first, requirements specification, describes the artifact's expected behavior and any applicable techno-economic constraints. The second, proceduralization, generates an executable description for realizing the anticipated behavior by executing sequences of elementary functions. The third, materialization, produces an artifact by following the procedure using the available physical resources. Design is an essential component of any engineering activity. By its nature, it is a "problem-solving process".

As a rule, requirements are declarative. They are usually expressed in natural languages. For some application areas, they can be formalized by using logics. When requirements are expressed by logical specifications, they can be treated as axioms; proofs that the artifact meets them can start from there. Proceduralization can be considered as a synthesis problem: procedures are executable models meeting the specifications. Unfortunately, model synthesis from logical requirements often runs into serious technical limitations such as non-computability or intrinsically high complexity.

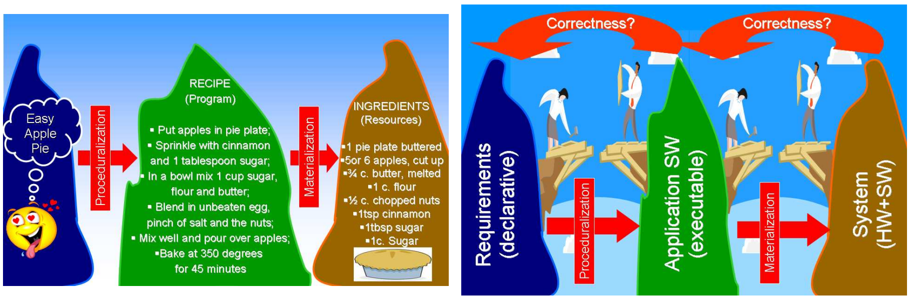
Easy Apple Pie
Proceduralization
RECIPE (Program)
▪ Put apples in pie plate;
▪ Sprinkle with cinnamon and 1 tablespoon sugar;
▪ In a bowl mix 1 cup sugar, flour and butter;
▪ Blend in unbeaten egg, pinch of salt and the nuts;
▪ Mix well and pour over apples;
▪Bake at 350 degrees for 45 minutes
Materialization
INGREDIENTS (Resources)
▪1 pie plate buttered
▪5or 6 apples, cut up
▪¾ c. butter, melted
▪1 c. flour
▪½ c. chopped nuts
▪1tsp cinnamon
▪1tbsp sugar
▪1c. Sugar
Correctness?
Correctness?
Requirements (declarative)
Proceduralization
Application SW (executable)
Materialization
System (HW+SW)

Fig. 1. Design is a universal concept applicable from cooking to computing systems

What happens when requirements are uncertain? To deal with uncertainty, engineers developed empirical approaches including requirements capture, incremental prototyping, incremental testing, etc. Some of these are done with solid science protocols focused on experimental design, data analysis, and hypothesis testing. Others are much less formal and involve many rules of thumb based on engineering experience. A design science would clarify the role of such empirical methods, and it would also address the issue of designing against uncertain requirements.

Design formalization raises a multitude of deep theoretical problems related to the conceptualization of needs in a given area and their effective transformation into correct artifacts. So far, it has attracted little attention from theoreticians. One reason is the predilection of the academic world for simple and elegant theories. Another reason is that design is by nature multi-disciplinary. Its formalization requires consistent integration of heterogeneous models supporting different levels of abstraction including logics, algorithms, programs, physical system models, risk models, statements about user practices and statements about esthetics.

Despite the challenges, providing systematic and well-founded design techniques is of paramount importance for two reasons. The first is that we need to construct artifacts of guaranteed quality and performance based on scientific evidence. This is the case for airplanes, cars, critical resource management systems as well as for critical computing and communication infrastructure. The second reason is the need to master as much as possible through automation the complexity and development costs of increasingly sophisticated artifacts. This need can be illustrated by numerous manufacturing setbacks experienced by the aircraft industry e.g. the A380 delivery delay or recent safety concerns with Boeing's Dreamliner.

Ideas for "scientizing" design emerged in the beginning of the 60's [1, 2]. There exists today an abundant literature on design science e.g. [3] and on design science and computing, in particular [4–7]. In this paper, we present our view about design science and propose a vision determining its scope and perimeter.

2 Bringing Science to Design

Science is a disciplined and systematic method for building, organizing and using knowledge about the world. We consider that scientific investigation intimately combines two interdependent processes. The first process is descriptive and intended to develop theory connecting some observed reality through abstractions to the world of concepts and mathematics. The second process is prescriptive and consists in applying a theory in order to assess its explicability and predictability as well as to invent things that do not yet exist. Interaction and cross-fertilization between these two processes is key to the progress of scientific knowledge. Today, more than ever, the two processes are involved in an accelerating virtuous cycle for the advancement of scientific knowledge.

The starting point in the scientific investigation cycle need not be observation. The theory of relativity was motivated by a series of thought experiments rather than direct observation. The development of computing as a scientific discipline started from prior knowledge about computation based on mathematics and logic.

We consider that design science studies design as a process for developing artifacts meeting given requirements for trust, function, safety, reliability, esthetics, cost containment, etc. Our definition significantly differs from others in the literature because it emphasizes the modeling-prediction cycle to support the application of technical means to aid the design processes. Design science is also concerned with deriving from the applied knowledge of the natural sciences appropriate information in a form suitable for the designer's use.

Inherent technical difficulties and limitations aggravate these problems. Nonetheless, we believe that their analysis and formalization in a flow leading from requirements to their materialization, can bring interesting insights about the very nature of artifact creation. For many aspects of design it will be impossible to achieve full automation. The main benefit from a scientific approach to design is rigor. Design can be sketched out as an iterative process consisting of steps supported by a methodology. While guaranteeing the correctness of designs may be an unattainable goal, a guarantee of accountability is realistic; accountability means that at each design step it should be possible to know which requirements hold, which ones do not hold and why.

3 Principles and Problems

We proposed a characterization of design science as a formal process encompassing the three stages of requirements expression, proceduralization and materialization. Each stage corresponds to a main type of problem to be solved by designers. Next, we discuss four principles that should drive the definition of a design process.

3.1 Four Driving Principles

Separation of Concerns. Design consistently integrates a sequence of three stages: requirements specification, proceduralization and materialization.

- Requirements express, usually in some declarative language, why we build an artifact. They express intention and needs motivating the design.
- Proceduralization consists in discovering what functionality should be ensured by the designed artifact to meet functional requirements; then it provides a procedure for composing atomic components, each component providing some elementary function or service.
- Materialization defines how function components can be implemented by using physical components. Implementation choices are driven by extra-functional requirements which are mainly trade-offs between cost and performance.

This three-stage decomposition is essential from a methodological point of view. It allows complexity to be tamed as it clearly separates concerns (why, what, how) by separately addressing three difficult problems. Furthermore, the distinction between proceduralization and materialization allows artifacts to be built providing the same functionality under different techno-economic constraints. This makes possible the development of families of artifacts with identical functional features and different performance and cost characteristics.

Each stage of a design process may be further decomposed into steps. At each step, the designed artifact is described at a certain level of abstraction by using an adequate modeling language. Each step progressively reduces abstraction by replacing conceptual constructs and primitives by more concrete ones. The final model is a blueprint for building the physical implementation.

Separation of concerns should be supported by an adequate design methodology. A design methodology identifies designer activities that can be supported by state-of-the-art tools to automate tedious and error-prone tasks. It also precisely determines where human intervention and ingenuity are needed to resolve design choices through requirements analysis and confrontation with experimental results. Identifying adequate design parameters and channeling the designers creativity are essential in this enterprise. The interaction between designer and supporting tools may involve iterations to eliminate design errors and determine optimized solutions.

Semantic Coherency. Designers use a variety of languages for the description of the behavior of the designed artifact at different abstraction levels. These include declarative languages for expressing requirements, and procedural languages for modeling, simulation, and performance analysis. Designers use, above all, domain-specific languages for example, for buildings, mechanical systems, electric systems, control-based systems, hardware description languages, and web-based systems.

Frequently, these languages only have informal semantics, which make it difficult not only to verify that the requirements capture their intended meanings, but also to reconcile different models that are brought together in a design process. This may be a source of design errors. To achieve semantic coherency, and minimize those errors, all these languages must be rooted in a common semantic model. The choice of the semantic model depends on the type of designed artifact for example, geometric model, differential equations, or abstract machines.

Semantic coherency enforcement may be completely transparent for the designer. It can be handled by translation tools such as compilers, interpreters, and model transformers.

A common semantic model is essential for rigor of design. It characterizes correctness through a semantic equivalence relation between artifact descriptions in the different languages that appear in each design stage. An essential requirement for a common semantic model is that it directly encompasses primitives and constructs of the hosted languages to avoid combinatorial explosion of the translation [8].

Component-Based Construction. Building larger structures from smaller components enhances productivity and correctness, and is essential in any design process. Components hide behavioral details behind interfaces that highlight their interactions with their environment. They can be assembled into composite components by partially composing their interfaces. Their semantics are defined by stating the rules of the composite component in terms of the behaviors of its constituent components. Component composition can use a large variety of mechanisms expressing how the behaviors of the composed components are restricted through mutual interaction.

Designers need a unified composition paradigm for describing and analyzing coordination between components in terms of tangible, well-founded, and well-organized concepts.

Correctness-by-Construction. Correctness means that a designed artifact meets its requirements specifications. Many designers consider it an ideal to establish correctness by checking that a design, once completed, meets its specifications. This ideal is usually impossible because automatic verification entails intractable computations. The size of state space to be examined by a verification method explodes exponentially with the number of components in the artifact. The best we can do is limit automatic verification to small or medium size models and to specific properties. Some researchers have investigated compositional verification techniques, which aim to decompose a global requirement for a composite artifact into sets of requirements for its constituent components. So far, compositional verification approaches have failed to make any significant breakthrough [9].

An alternative approach is to establish correctness-by-construction incrementally, as you go along the design process, through the combined application of three principles: property enforcement, property composability, and property preservation.

Property enforcement: Property enforcement is very common in engineering. Engineers extensively use principles for building complex artifacts from components so as to meet given properties. These principles can be embodied in patterns for buildings design or for software design, in communication protocols, in distributed algorithms, in hardware or system architectures. They are solutions to particular problems. For example, a communication protocol ensures reliable message transmission despite packet losses. A token-ring algorithm ensures mutual exclusion in a distributed system. Client-server architectures ensure atomicity of transactions and fault-tolerance. All these component coordination mechanisms can be reused provided they are adequately formalized. They allow correctness almost for free.

Notice that property enforcement enables designers to ensure that compositions of components meet a specific global requirement. In contrast to compositional verification, it does not require breaking up the global requirement into sub-requirements to be met by components. For example, we do not have general compositionality theory for deadlock-freedom preservation. Nonetheless, specific protocols or architectures may be used to build deadlock-free systems from deadlock-free components.

Property composability: A key issue in this approach is maintaining coherence while combining multiple existing solutions to specific problems. For example, a database programmer might apply several instances of a lock algorithm, but their multiple application may contain a deadlock. How does the designer know that the combination of correct solutions might be unsafe?

Another illustration of this problem comes from fault-tolerant computing. Fault-tolerant systems combine multiple methods for protection against invalid actions, including: 1) triple modular redundancy mechanisms ensuring continuous operation in case of single component failure; 2) hardware checks to validate that programs use data only in their defined regions of memory; 3) default to least privilege (least sharing) to enforce file protection; 4) checkpoints that permit backing up to, and restarting from, a prior valid system state in case of a failure. If we combine all these methods, how can we be sure there are no unwanted interactions that make the system prone to new faults?

Guaranteeing non-interaction of features is essential for correct-by-construction design. Violations of this principle invariably cause trouble. For example, features of telecommunication systems frequently interact, causing user confusion and misuse. Interference among web services and among features in aspect programming are additional examples.

Property preservation: When a requirement holds for an artifact description at some design step, it is essential that it remains valid at all subsequent steps. This allows establishing correctness incrementally. Each new modeling step must not invalidate correctness of previous steps. Artifact models are progressively built by first ensuring validity of each functional requirement and then models are refined to satisfy additional extra-functional requirements. Model refinement can be characterized as a preorder relation between models. It can be implemented through a set of model transformation rules. The demand for property preservation means that these rules preserve the semantic equivalence of models.

3.2 Three Basic Problems

The three stages of the design process correspond to three types of basic problems. Their solution is aggravated by many factors including undecidability, overwhelming algorithmic complexity, conceptual ambiguity, and physical uncertainty. The objective is not to tackle these problems in their full generality but rather to identify avenues for their partial solution in specific application contexts by supporting the designer's ingenuity with automation.

Formalizing Requirements. Many design processes begin with an expression in a rigorous language that declares the needs to be met by an artifact and the associated techno-economic constraints. Several difficulties obstruct full formalization of requirements. Requirements are by their nature declarative; thus logic is, in principle, an adequate framework for their expression. However, requirements are initially expressed in natural languages, which usually allow ambiguities. Ambiguities inhibit translation into a formal language, limiting the designer's ability to be systematic and rigorous. In addition, many requirements are meant to describe the behavior of the artifact in context of its environment including its potential users. Formalization of an artifact's environment is no easy task it must be done at the right abstraction level, accounting for all the relevant behavioral properties. Today we lack theoretical approaches for tackling this problem.

The concept of correctness conjoins two types of requirements: 1) trustworthiness requirements ensuring that nothing bad could happen; 2) optimization requirements for performance, cost-effectiveness, and tradeoffs between them. Trustworthiness characterizes qualitative correctness. It means that the artifact can be trusted, and that it will behave as expected. It accounts for non-vulnerability to hazards such as: 1) design errors; 2) physical failures and defects; 3) interaction with potential users including erroneous use and threats; and 4) interaction with the physical environment including disturbances and unpredictable events.

Optimization requirements deal with the optimization of functions subject to constraints involving resources used for implementing and using the artifact. They deal with: 1) requirements on performance metrics such as throughput and response time, which characterize how well the artifact does with respect to user-defined criteria; 2) cost-effectiveness, which characterizes how well resources are used with respect economic criteria; 3) tradeoffs between performance and cost-effectiveness.

Trustworthiness and optimization requirements can be difficult to reconcile. As a rule, improving trustworthiness causes wasted resources. Conversely, resource optimization may jeopardize trustworthiness. Designers try to balance trustworthiness and optimization.

There is a limit to how far we can push a formal logic approach to requirements. The biggest problem is that users themselves often cannot articulate their deep concerns about trust and performance. How can we formalize what the customer cannot say? For example, with computer security, how can we be exhaustive and precise about threats from unseen or unsuspected adversaries? Here there is a real possibility that empirical approaches can help. We can build prototypes of systems and ask users to try them out and tell us about good points and problems. By systematically iterating between prototypes and customer assessments, we can converge on a set of requirements that earn their trust. The big challenge is to develop sound scientific methods for this process and reconcile the experimental results with the mathematical models.

Proceduralization. Proceduralization is a synthesis problem: find a procedure that builds functionality meeting given requirements from a set of predefined atomic components of known functional characteristics. Unfortunately, most non-trivial instances of this problem are computationally intractable for example, program synthesis from logical specifications.

A pragmatic approach for tackling this problem is to strive to bridge the gap between declarative and procedural languages by working in two directions.

One direction is to raise the abstraction of languages to get them as close as possible to the declarative style. This would simplify reasoning and relegate procedure generation to tools. Many approaches for enhanced abstraction proposing logical, constraint-based, and functional description languages already exist. They are equipped with interpreters or compilers that allow automatic synthesis of procedural descriptions.

The other direction is to develop adequate domain-specific languages allowing ease of description as well as enhanced safety and productivity. Examples include Matlab/Simulink, HTML, Logo, SQL, BPEL and hardware description languages.

Materialization. Materialization consists in exploring cost-performance tradeoffs among all the possible physical implementations of the desired functionality. It involves extensive empirical evaluation and hypotheses testing to determine designs better fitting cost-performance requirements. The exploration can be performed on an adequate model obtained from the procedural description by assigning to its elements models of functionally equivalent physical components. In addition to their functionality, this transformation should take into account physical characteristics, such as execution times, latency, and power consumption. The obtained model should faithfully describe the dynamic behavior of the artifact, including both the provided functionality and its global physical properties. A key issue to consider when building such models is their predictability that is the degree to which their quantitative properties can be asserted. For example, the materials laid down in layers during a 3-D print may not satisfy the continuity assumptions of a mathematical model; experimental validation of stress-bearing properties of those materials is essential. Building predictable models raises deep theoretical problems [10] and requires a marriage of formal methods and scientific validation methods.

Design space exploration techniques are intended to determine an optimal assignment of physical components that fits the user-defined cost-performance requirements. Currently, they are mostly ad hoc and consist in evaluating the impact of design parameters on the requirements. The challenge for designs science is to make the formal and experimental sides of design mutually compatible and reinforcing.

Design space exploration allows estimating combinations of parameters that better fit the requirements. The main challenge is the complexity of exploring the design space. State-of-the-art techniques for overcoming complexity combine symbolic representation of the design space and theoretical results for accelerating the exploration process [11].

4 Toward a Design Science

Even though forty years have passed since the first seminal ideas about design science, very little progress has been made toward defining its technical goals and clarifying its scope and limits. It is time to develop technical work contributing to the advancement of our knowledge about design as a universal paradigm amenable to both scientific analysis and rigorous practice. In this essay we have argued that design is a rigorous process involving the successive solution of three types of problems. We proposed principles and associated scientific challenges for putting design into practice.

We believe that achieving this goal is not only a matter of writing down a theory of design, it will require hard work with semantic models, empirical methods for dealing with uncertainty in requirements, flexibility for dealing with changing environments, testing, automated materialization, and the maturing of correct-by-construction principles. This vision is both intellectually challenging and culturally enlightening. It is at least of equal importance as the quest for scientific discovery in natural sciences.

Endowing design with scientific foundations is a huge intellectual challenge that would meet an urgent demand for cost-effectively building complex, trustworthy artifacts. Failure in this endeavor, would seriously limit our capability to master the techno-structure and its further development intended to address urgent global challenges for optimal resource management and enhanced services. It would also mean that designing is a definitely a-scientific activity [12] driven by predominant subjective factors that make for ineffectual rational treatment.

Meeting this challenge would significantly enhance our capability to build trustworthy artifacts, and would confirm that design is definitely a scientific activity.

Acknowledgments. Peter Denning and Richard Snodgrass contributed to significantly improving the paper through constructive comments and criticism.

References

1. Simon, H.A.: The Sciences of the Artificial, 3rd edn. MIT Press, Cambridge (1996)
2. Alexander, C.: Notes on the synthesis of form. Harvard University Press, Cambridge (1964); Autres tirages: 1968, 1971
3. Cross, N.: Designerly ways of knowing: Design discipline versus design science. Design Issues 17(3), 49–55 (2001)
4. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. 28(1), 75–105 (2004)
5. Winter, R., Zhao, J.L., Aier, S. (eds.): DESRIST 2010. LNCS, vol. 6105. Springer, Heidelberg (2010)
6. Peffers, K., Rothenberger, M., Kuechler, B. (eds.): DESRIST 2012. LNCS, vol. 7286. Springer, Heidelberg (2012)
7. Henzinger, T.A., Sifakis, J.: The discipline of embedded systems design. Computer 40(10), 32–40 (2007)

8. Bliudze, S., Sifakis, J.: A notion of glue expressiveness for component-based systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 508–522. Springer, Heidelberg (2008)
9. Cobleigh, J.M., Avrunin, G.S., Clarke, L.A.: Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. ACM Trans. Softw. Eng. Methodol. 17(2), 7:1–7:52 (2008)
10. Thiele, L., Wilhelm, R.: Design for timing predictability. Real-Time Syst. 28(2-3), 157–177 (2004)
11. Mohanty, S., Prasanna, V.K., Neema, S., Davis, J.: Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In: Proceedings of the Joint Conference on Languages, Compilers and Tools for Embedded Systems: Software and Compilers for Embedded Systems, LCTES/SCOPES 2002, pp. 18–27. ACM, New York (2002)
12. Grant, D.: Design methodology and design methods. Design Methods and Theories 13(1) (1979)