Chapter 6
A Science of Design for Software-Intensive Systems

There is something fascinating about science. One gets such wholesale returns of conjecture out of such a trifling investment of fact.

Mark Twain, Life on the Mississippi, 1883

6.1 Science of Design Challenges

Future complex software-intensive systems (SIS) will be vastly different from the software systems that run today's world. Revolutionary advances in hardware, networking, information, and human interface technologies will require entirely new ways of thinking about how software-intensive systems are conceptualized, built, and evaluated. As we envision the future of tera[1]-computing and even peta[2]-computing environments, new science of design principles are needed to provide the foundations for managing issues of complexity, composition, quality, cost, and control of software-intensive systems.

Evidence suggests that software-intensive systems development has already reached the limits of technologies developed in the first 60 years of computing. New, innovative principles, practices, and tools will be needed to move software development into the next generation of computing environments. Manual methods of software and systems engineering must be replaced by computational automation that will transform the field into a true scientific and engineering discipline. Other science/engineering fields have made this transformation to their everlasting benefit. Computational theories, models, and tools of subject matter dominate mature disciplines, such as electrical engineering and aeronautical engineering. Analogous

[1]Tera = 10^{12} or 1 trillion. Tera-computing environments support trillion-line software programs running on networks connecting trillions of computers at terahertz bandwidth speeds.
[2]Peta = 10^{15} or 1000 trillion.

A. Hevner, S. Chatterjee, *Design Research in Information Systems*, Integrated Series in Information Systems 22, DOI 10.1007/978-1-4419-5653-8_6,
© Springer Science+Business Media, LLC 2010

computational models for software are now just emerging and must be incubated with focused research and development (R&D) and supportive demonstration environments. While much of the research focus during first 60 years of computing was on correct syntax-directed computation of details for computer execution, the focus of the next 60 years will shift to semantics-directed computation of correct abstractions for human understanding and manipulation.

The challenges of building large-scale software-intensive systems are unique and very different from the challenges of building large physical systems. Wulf (2006) identifies three principal reasons for the unique challenges of software-intensive systems:

1. Software-intensive systems are more complex than physical systems. Emergent properties are difficult to predict. We do not understand the science and first-class properties of software design.
2. Software has fewer constraints than physical systems. Thus, there are many more design options. The design space is enormous. It is very difficult to understand, model, and make effective design trade-offs for software-intensive systems.
3. The mathematics describing software-intensive systems lacks continuity. Discrete mathematics does not support efficient testing and analysis of software. It is impossible to exhaustively test a software-intensive system based on the discontinuities of the underlying mathematics.

A new vision of science of design research for SIS must achieve the following essential objectives:

- Intellectual amplification: Research must extend the human capabilities (cognitive and social) of designers to imagine and realize large-scale, complex software-intensive systems.
- Span of control: Research must revolutionize techniques for the management and control of complex software-intensive systems through development, operations, and adaptation.
- Value generation: Research must create value and have broad impacts for human society via the science and engineering of complex software-intensive systems and technologies.

The goal of this chapter is to present a vision of science of design research directions and to propose a framework for achieving this vision. The content of this chapter has benefitted greatly from my experiences at the National Science Foundation (NSF) during 2006–2008 and draws from many discussions with colleagues at NSF which I gratefully acknowledge here.

6.2 Software-Intensive Systems

A difficulty faced when discussing research in the field of software-intensive systems is the lack of common terminology for key concepts. The field is teeming with terms that are overloaded with meanings (e.g., system, design) or varied terms for the same basic concept (e.g., object, component, module). The goal of the following discussion is not to propose a new ontology but to simply define the terms used in this chapter.

A *system* can be defined generally as a collection of elements that work together to form a coherent whole. *Software-intensive systems (SIS)*, then, are systems in which some, but not necessarily all, of the component elements are realized in software. Figure 6.1 illustrates three layers of any software-intensive system – the human layer, the software layer, and the platform layer. Two critical interfaces are shown – the human–software interface and the software–platform interface.

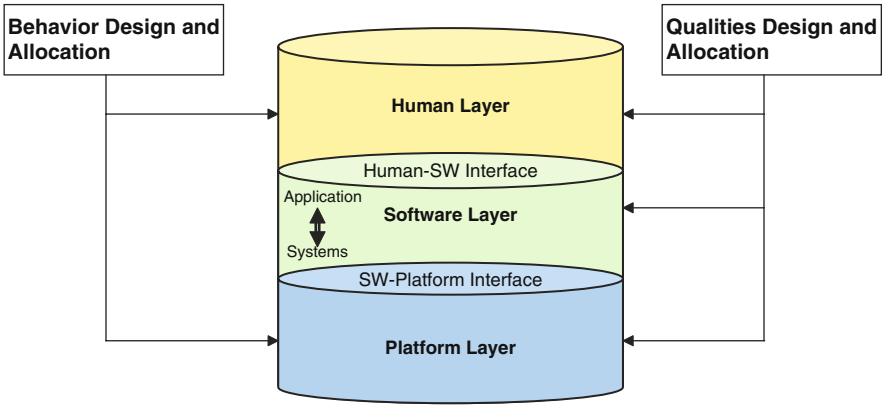

Fig. 6.1 Software-intensive system layers

The development of a SIS entails many important decisions such as the design and allocation of system behaviors (e.g., functions, actions) and system qualities (e.g., performance, security, reliability) to the different layers. For example, a particular system activity could be realized in hardware (platform), via a service call (software), by human behavior (human), or some combination of activities across all three layers. Likewise, a performance requirement (e.g., response time) for a SIS transaction could be divided and allocated as performance requirements in each of the layers.

Figure 6.2 shows the growth of the software layer, in size and percentage of the overall system, as a future trend. The role of software will become dominant in nearly all complex systems. Thus, research and development in SIS must actively address the challenges of using software as the primary building material in future complex systems.

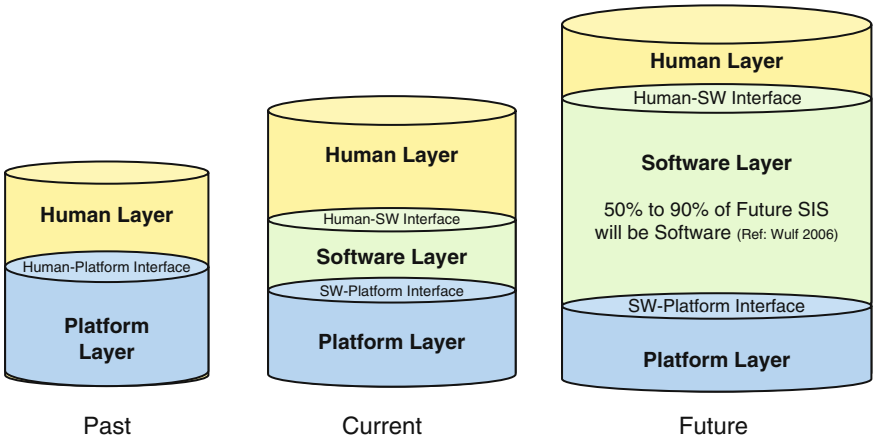

Fig. 6.2 Software-intensive system trends

Beyond individual, self-contained SIS, nearly all future systems will be connected to environmental resources and other systems via network connections. These connections lead to complex systems-of-systems architectures for providing behaviors and qualities. Figure 6.3 demonstrates that there are identifiable networks across all three SIS layers. Physical networks support the transmission of digital and analog data among system platforms. Software networks provide the middleware layers and protocols that transform the transmitted data into information that is shared among the information processing systems. Social networks provide a means of interaction and community among the human participants of the complex system (Fiadeiro 2007). Humans use the system information to make decisions, execute actions on the environment, and build application domain knowledge bases.

Figure 6.4 zooms in on the software layer to show its makeup of software code, information, and control within the context of an application domain. The overlaps among these three concepts support varying methods and techniques of understanding and building the software layer of systems. For example, software architectures define structures for integrating the concepts of code, information, and control for a particular application domain system. The message is that the software layer in a SIS is a challenging and fertile field of research opportunities.

6.3 Science of Design Principles

The science that provides foundations for the engineering of complex software-intensive system must be predicated on a set of fundamental principles. A principle is a clear statement of truth that guides or constraints action. A principle can also be formed as a rule or a standard of conduct. It is this search and discovery of fundamental principles that underlie the research agenda of a SIS research program.

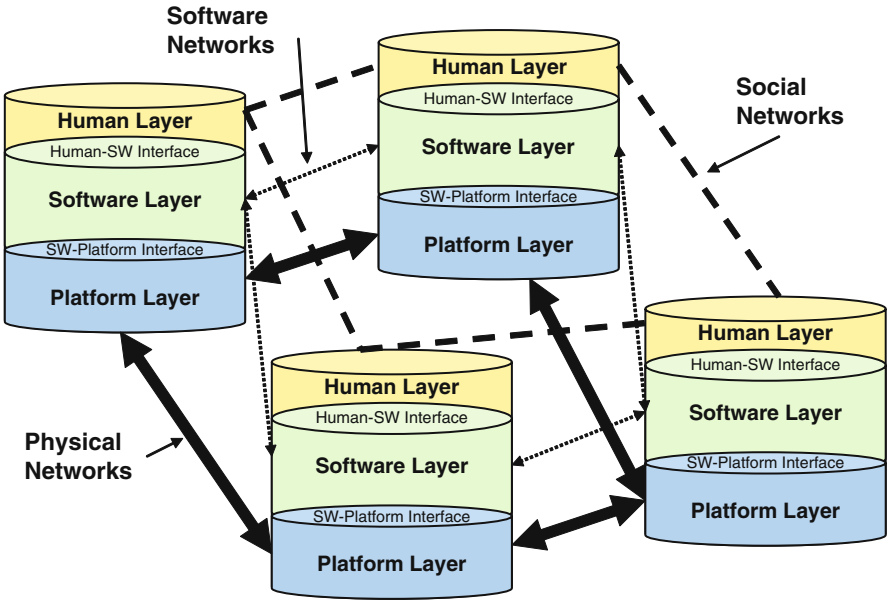

Fig. 6.3 Physical, software, and social networks of software-intensive systems

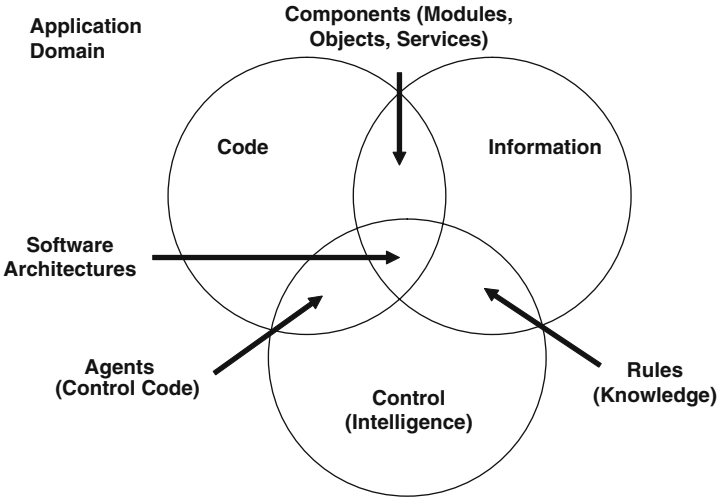

Fig. 6.4 Concepts in the software layer

Table 6.1 Current CSIS development principles

Principle	Related practices	References
System Abstraction	Hierarchical decomposition Systems architecting	Simon (1996); Maier and Rechtin (2000)
Levels	Protocol layering	Dijkstra (1968)
Information hiding	Objects Object-oriented languages	Parnas (1972); Kay (1984)
Intellectual control	Computer architectures Chief programmer teams	Brooks (1995); Mills (1983)
Computational thinking	Relational database Systems and languages	Codd (1970)
Form and function	Design patterns	Alexander (1979)
Economics of systems	Software economics	Boehm (1981)

This vision for a set of science of design principles aligns with Peter Denning's project to identify a framework for the fundamental principles for the field of computing (Denning 2003, 2005).[3] Whereas Denning's goal is to demonstrate the central role of computing as a true scientific field in relationship to other sciences; the goal of a SIS research program will be to discover, articulate, and use these principles to guide the effective and efficient development of future complex software-intensive systems.

A quick look back at the computer science field shows only a handful of truly fundamental principles that have guided the current development of complex software-intensive systems over the past 60 years. Table 6.1 summarizes several of these key principles.

The research and development projects that led to the principles and related practices found in Table 6.1 were transformative in providing breakthrough ideas for developing complex software-intensive systems. New transformative ideas are needed to move the field forward to build and manage SIS for the 21st century.

6.4 Categories of Software-Intensive Systems Principles

The future challenge is to bring researchers from multiple disciplines to discover and define fundamental principles and practices upon which future complex software-intensive systems will be imagined, architected, designed, built, and operated. As exemplars of critically important categories of fundamental principles that should be addressed in science of design research, we propose the following:

[3] Great Principles of Computing web site: http://cs.gmu.edu/cne/pjd/GP

- Computational principles: Computational thinking underlies true scientific and engineering fields (Wing 2006; Denning 2007). The field must better identify its rigorous mathematical and computational foundations to support the more effective and efficient SIS representations, models, analyses/manipulations, development methods/tools, and system instantiations
- Scalability principles: Scalability of system concepts is absolutely essential in order to build ultra-large-scale systems (Northrop et al. 2006). Effective ideas must apply equally well to the development and operations of small systems and of massive, complex systems.
- Creative principles: We solicit fundamental principles that enhance human cognitive abilities and support the creative process in the development of complex software-intensive systems. Effective human–computer interfaces for both development environments and application systems will embody these principles.
- Adaptability principles: In the development of future SIS, it will be impossible to specify or predict a priori all of the behaviors or qualities of the system. Runtime composition of system components will result in unknown, emergent behaviors and qualities during operation. Thus, key principles of adaptability must be discovered and applied to manage the evolution of the system as it adapts to its environment and transaction load.
- Ethical principles: The development of a SIS implies an ethical responsibility of how the system shapes its environment. The consequences of the system are formed by the ethical principles on which it was designed and built. Ethical principles would help us understand what values inform the development of the SIS and whose interests are served by the system.
- Economic principles: Cost goes hand-in-hand with complexity. Complex systems cost significantly more to develop, produce, and operate. Under current economics, the coordination costs of SIS rise exponentially with increases in size and complexity. A deeper understanding of the economic principles of complex systems is required in order to evaluate the feasibility and market impacts of SIS.
- Decidability principles: Consideration of a wide range of fundamental principles must eventually lead to decisions on how to imagine, architect, design, build, and operate a desired complex software-intensive system. What are the decidability principles that underlie the construction of such SIS decision models? How, for example, would we evaluate trade-offs among ethical principles and economic principles in a SIS decision model?

Among others, these are several of the key categories of principles that must be studied in a comprehensive science of design research agenda.

6.5 A Proposed Research Vision

A research vision for the science of design of software-intensive systems is presented in Figs. 6.5 and 6.6. As seen in Fig. 6.5, the intellectual merit of this research must be drawn from scientific theories originating from several disciplines

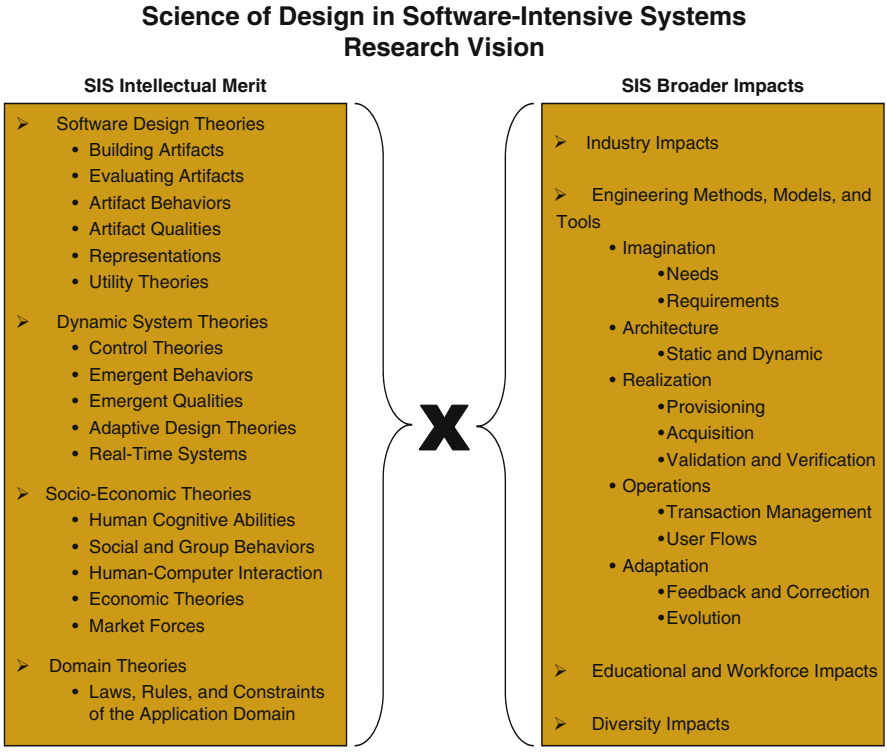

Fig. 6.5 Science of design in SIS research vision

including computer science, software engineering, systems engineering, socio-economic fields, and the application domain. The broader impacts of the research must be felt in the scientific community, industry, government, academia, and human society.

6.6 SIS Scientific Theories

The scientific theories for fundamental science of design research in SIS are identified in the following categories and briefly discussed. A full description of all these grounding theories is beyond the scope of this chapter but can be readily found via a literature review on the listed topics.

6.6.1 Software Design Theories

- Building artifacts
- Evaluating artifacts

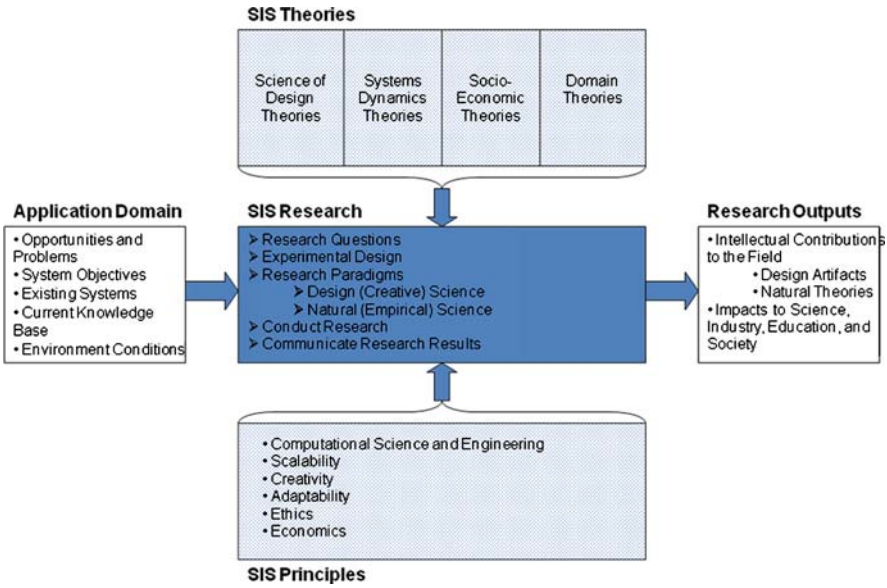

Fig. 6.6 Research on science of design in software-intensive systems

- Artifact behaviors
- Artifact qualities
- Representations
- Utility theories

New ideas in software design research will necessarily draw from the classic works of software theory by Simon (1996), Mills (1983), Parnas (1972), Dijkstra (1968), Brooks (1995), Freeman (1987), Boehm (1981), and many other thought leaders in the software field. These time-tested software principles are weighed alongside the latest ideas in the field to find the right balance of pure and applied research directions.

6.6.2 Dynamic System Theories

- Control theories
- Emergent behaviors
- Emergent qualities
- Adaptive design theories
- Real-time systems

Dynamic system theories provide the bases for understanding the dynamic behaviors of complex systems (Forrester 1961; Randers 1980). The essence is the

recognition that the structure of any system — the circular, interlocking, temporal, spatial, and sometimes non-deterministic relationships among its components — is often just as important in determining its behavior as the individual components themselves. There are often properties-of-the-whole which result in dynamic, emergent behaviors and qualities which cannot be explained in terms of the behaviors and qualities of the parts.

6.6.3 Socio-economic Theories

- Human cognitive abilities
- Social and group behaviors
- Human–computer interaction
- Economic theories
- Market forces

Socio-economic theories will play a major role toward understanding the factors leading to success or failure in the development and use of software-intensive systems. For example, the design of software system architectures and software components must support economic business cases for utility, marketability, usability, and other system features important for successful deployment. Initial research on economic models for software design, such as Baldwin and Clark's (2000) study of design in the computing industry, is an important first step in this research area. Another research area of great interest is the open-source models of software development, operations, and evolution.

6.6.4 Domain Theories

Each and every software-intensive system is embedded within an application domain from which it draws its relevance and utility. Domain theories provide essential laws, rules, and constraints that ground the development and use of all systems in that domain. For example, complex software-intensive systems for airplanes must be developed in full awareness of the science and engineering of aeronautics. Similarly, a banking system must be grounded in the regulations and policies of the international and national financial systems.

6.7 SIS Engineering Activities

The effective engineering of software-intensive systems of any size and complexity consists of five primary activities as performed by skilled development teams:

1. Imagination: All system stakeholders participate in imagining the needs and requirements of a desired system. New and better methods and models for capturing and specifying system requirements are greatly needed. In particular, the search for rigorous specification models that are efficiently usable by software developers and effectively understandable by system stakeholders remains an important research effort.
2. Architecture: The need to architect complex systems has been recognized by the scientific and engineering communities in all domain fields of design. Theories of systems architecture are commonplace in fields ranging from buildings and landscaping (Alexander 1979) to nanotechnologies and even to an evolving understanding of the architecture of the human brain and body. Research on software architectures is relatively new (Shaw and Garlan 1996; Bass et al. 2003) and many of the underlying principles and theories are yet to be discovered (Maier and Rechtin 2000). Architecting involves both the art and science of designing and building systems. The architecture of a software system can be envisioned as the structures, protocols, standards, and aesthetics that provide the required system behaviors, state, qualities, and, even beauty (Gelernter 1998).
3. Realization: The actual construction of the system can be realized in many different ways. Software components are composed in an integrated application system. A software component provides a unit of behavior in the software system (Brown 2000). The component can be realized in forms such as software modules, services, objects, routines, and functions depending on the development environment. Artifacts of component development include behavioral specifications, designs, program code, test cases for unit, integration, and system testing, and documentation for system operators and system users. The state of a software component is represented in its data structures – variables, files, and databases. The designs for service behavior and state go hand-in-hand to achieve the greatest service utility and quality.
4. Operations: The deployment and effective operation of a system in the application environment are key engineering challenges. The interactions of software components and key user and environmental interfaces during system execution make up the real-time behaviors of the software system as a whole. The goals of management and control of software dynamics leads to a number of interesting research topics. User transactions can be described and formalized as flows of control, data, and qualities among software components. As an example of this line of thinking, consider a user transaction in a software system as an identifiable flow with requested behaviors and qualities. This flow is presented to the software architecture in a dynamic environment which can determine at that point in time whether the software system can provide the behaviors and qualities requested. If the flow is allowed to execute on the architecture, its instantiation will draw behaviors and qualities from a dynamic composition of components in the software system. While some research and development has been performed in the areas of workflows and business process flows (e.g., Aalst and Hee, 2002), more work is needed on the analysis and design of complex, network-centric systems to support and optimize user flows.

5. Adaptation: It is impossible to specify or predict a priori all of the behaviors or dynamic properties of a complex system while it is operating in unpredictable, possibly adverse, dynamic environments. Runtime composition of systems will result in unknown, emergent behaviors and qualities during operation. Thus, dynamic composition methods and key principles of adaptability must be discovered and applied to manage the evolution of the system as it adapts to its environment and transaction load. Complex systems often operate in complex environments on which they have little control and to which they must react quickly and reliably. The flexible nature of software and its inherent malleability provide the potential for systems to adapt autonomously to environmental conditions.

6.8 SIS Research Project Framework

A science of design research project, as shown in Fig. 6.5, is a cross-product of the grounding scientific theories and broader impacts of the research to include the new contributions to an engineering phase of the software-intensive system life cycle. Figure 6.6 provides a SIS research project framework. The inputs to the research are the SIS theories, principles, and application domain and the output of the research is the contributions to the application environment (relevance) and the scientific knowledge base (rigor).

Examples of challenging science of design for SIS research questions include the following:

- How can we design and evaluate SIS architectures for future computing environments to achieve the greatest understandability, utility, and quality?
- Knowing that designs of complex systems emerge throughout the development process and operations, how do we build flexibility into processes, methods, and models?
- How do we analyze and perform trade-offs between information design, control design, and software design?
- How can complex systems be designed in environments where the component parts are developed and controlled by multiple, independent entities?
- How will new physical platforms (hardware, communications) be integrated most effectively into SIS? What new interfaces and systems software are needed?
- What economic and social trade-offs are needed to best describe and understand the dynamics of SIS and the impacts of those systems on industrial, governmental, and societal infrastructures?
- How do we produce software system designs leading to systems that have the capacity to respond to surprise in operational environments?
- How do we best achieve human in the loop for SIS to enable and enhance human capabilities and values? What new human–computer interfaces are needed?

6.9 Intellectual Drivers for Science of Design in SIS Research

To conclude this chapter, we will focus on three key intellectual drivers for science of design research in software-intensive systems. One potentially radical approach for rethinking SIS foundations is to start from a small set of intellectual drivers of systems thinking and then apply an in-depth understanding of these drivers to real-world problems via science of design SIS research. The following three system concepts provide the most basic challenges and opportunities for transformative research: complexity, composition, and control.

Managing *complexity* (technical, human, and societal) in the development, operation, and evolution of software-intensive systems is an overriding challenge. Research to rethink IS complexity can be inspired by models in other scientific fields, both physical sciences and social sciences. For example, consider the development of IS artifacts that have the same robustness in the presence of complexity as biological organisms. Designing models and methods for managing complexity will require creative ideas for new information technology (IT) abstractions, representations, and languages.

Rethinking complexity will necessarily lead to changes in the way the *qualities* of IT artifacts are viewed. Current thinking assumes that if an accurate system specification can be produced up front then a system that fits stakeholder needs will naturally follow. Such an assumption is wrong when systems become complex enough to result in unexpected, emergent behaviors and properties in unstable operational environments. Software-intensive IS are subject to multiple stakeholders' inconsistent, contradictory, and partially understood objectives for behaviors and properties, such as performance, reliability, security, usability, and sustainability. While model-checking technologies have provided some useful forms of systems assurance, new ways of understanding and conceptualizing how IS qualities can be measured and evaluated are desired.

The essence of SIS design and evolution is *composition* of the system from component parts that may be developed by different parties in different languages and to different specifications. Mashups are examples of innovative approaches for composing disparate components of software and information. A composed system must interact properly with complex, uncertain environments, and the aggregate must be trusted. This concept requires that IS implementations respect the concerns of the domain, the intended usage, and the technology substrate (hardware and software) upon which systems execute. Successful identification of useful properties of IS must draw upon the relevant disciplines. We need new theories of abstraction, structuring, behavior and configuration as well as new logics for representing and reasoning about large systems in support of efficient and sustainable component-oriented engineering approaches. New theories of complexity and composition are needed to predict and reason about scalability in ways that can be empirically verified. A key challenge will be to identify perspicuous, useful, end-to-end properties and models that span hardware and software technology platforms, the problem domain, user interaction, and context of use.

Control of SIS has become increasingly challenging in situations of diverse software and data provenance, such as open-source communities and dynamic supply chains. In such settings, requirements for dynamic composition have both human and automation aspects. Human cognition imposes limits on our abilities to design complex artifacts. New techniques to augment human intellectual control and coordination of the design, development, and use of complex software-intensive systems are desired. For example, autonomic control of large-scale, distributed software-intensive systems can reduce or remove the requirement for human attention during runtime while still satisfying the needs of human users. Concepts of software system self-awareness and human–computer partnerships can lead to optimum system performance, negotiated access to resources, and novel IS configurations suitable to a particular situation. Research projects in this field might be inspired by emerging ideas in collective intelligence (e.g., wisdom of the crowds), virtual organizations (e.g., open-source user communities), and cognitive theories of abstraction, decomposition, and synthesis.

As we enter a future world of pervasive computing and ubiquitous cyber-physical devices it is essential that IT artifacts and the integrated systems containing these artifacts are reliable, adaptable, and sustainable. Science of design for SIS research must draw its foundations from multiple research disciplines and paradigms in order to effectively address a wide range of system challenges. Three of the most important intellectual drivers of future science of design in SIS research will be dealing with complexity, composition, and control. Consideration of these drivers must be the basis for the design of innovative artifacts and the development of rigorous theories to rethink the development, evolution, and adaptation of future information systems.

References

van der Aalst, W. and K. van Hee (2002) *Workflow Management: Models, Methods, and Systems*, The MIT Press, Cambridge, MA.

Alexander, C. (1979) *The Timeless Way of Building*, Oxford University Press, Oxford.

Baldwin, C. and K. Clark (2000) *Design Rules: The Power of Modularity*, The MIT Press, Cambridge, MA.

Bass, L., P. Clements, and R. Kazman (2003) *Software Architecture in Practice*, 2nd edn, Addison-Wesley, Boston, MA.

Boehm, B. (1981) *Software Engineering Economics*, Prentice-Hall, Upper Saddle River, NJ.

Brooks, F. (1995) *The Mythical Man-Month: Essays on Software Engineering*, 2nd edn, Addison-Wesley, Reading, MA.

Brown, A. (2000) *Large-Scale Component Based Development*, Prentice-Hall, Upper Saddle River, NJ.

Codd, E. (1970) A relational model of data for large shared databanks, *Communications of the ACM* 13 (6), pp. 380–387.

Denning, P. (2003) Great principles of computing, *Communications of the ACM* 46 (11), pp. 15–20.

Denning, P. (2005) Is computer science science? *Communications of the ACM* 48 (4), pp. 27–31.

Denning, P. (2007) Computing is a natural science, *Communications of the ACM* 50 (7), pp. 13–18.

Dijkstra, E. (1968) The structure of the 'T.H.E.' multiprogramming system, *Communications of the ACM* 11 (5), pp. 341–346.

Fiadeiro, J. (2007) Designing for software's social complexity," *IEEE Computer*, 40 (1), pp. 34–39.
Forrester, J. (1961) *Industrial Dynamics*. Pegasus Communications, Waltham, MA.
Freeman, P. (1987) *Software Perspectives: The System is the Message*, Addison-Wesley, Reading, MA.
Gelernter, D. (1998) *Machine Beauty: Elegance and the Heart of Technology*, Basic Books, New York.
Kay, A. (1984) Computer software, *Scientific American*, 250, pp. 41–47.
Maier M. and E. Rechtin (2000) *The Art of Systems Architecting*, 2nd edn, CRC Press, Boca Raton, FL.
Mills, H. (1983) *Software Productivity*, Little, Brown, and Co., Boston, MA.
Northrop, L. et al. (2006) *Ultra-Large-Scale Systems: The Software Challenges of the Future*, Software Engineering Institute Report at http://www.sei.cmu.edu/uls/files/ULS_Book2006.pdf, Carnegie-Mellon University.
Parnas, D. (1972) On the criteria for decomposing systems into modules, *Communications of the ACM* 15 (12), pp. 1053–1058.
Randers, J. (1980) *Elements of the System Dynamics Method*, MIT Press, Cambridge, MA.
Shaw, M. and D. Garlan (1996) *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, Englewood Cliffs, NJ.
Simon, H. (1996) *The Sciences of the Artificial*, 3rd edn, The MIT Press, Cambridge, MA.
Wing, J. (2006) Computational thinking, *Communications of the ACM* 49 (3), pp. 33–35.
Wulf, W. (2006) Keynote Presentation to USC Center for Software & Systems Engineering Symposium, Los Angeles, CA.