01 Jan 2022

Information Systems Analysis and Design: Past Revolutions, Present Challenges, and Future Research Directions

Keng Siau
Missouri University of Science and Technology, siauk@mst.edu

Carson Woo

Veda C. Storey

Roger H.L. Chiang

et. al. For a complete list of authors, see https://scholarsmine.mst.edu/bio_inftec_facwork/444

Recommended Citation

Siau, K., Woo, C., Storey, V. C., Chiang, R. H., Chua, C. E., & Beard, J. W. (2022). Information Systems Analysis and Design: Past Revolutions, Present Challenges, and Future Research Directions. *Communications of the Association for Information Systems, 50*(1), pp. 835-856. Association for Information Systems.
The definitive version is available at https://doi.org/10.17705/1CAIS.05037

Communications of the Association for Information Systems

Volume 50 Article 33

6-27-2022

Information Systems Analysis and Design: Past Revolutions, Present Challenges, and Future Research Directions

Keng Siau
City University of Hong Kong

Carson Woo
University of British Columbia

Veda C. Storey
Georgia State University

Roger H. L. Chiang
University of Cincinnati

Cecil E. H. Chua
Missouri University of Science and Technology

See next page for additional authors

Recommended Citation

Siau, K., Woo, C., Storey, V. C., Chiang, R. H., Chua, C. E., & Beard, J. W. (2022). Information Systems Analysis and Design: Past Revolutions, Present Challenges, and Future Research Directions. Communications of the Association for Information Systems, 50, pp-pp. https://doi.org/10.17705/1CAIS.05037

Information Systems Analysis and Design: Past Revolutions, Present Challenges, and Future Research Directions

Authors

Keng Siau, Carson Woo, Veda C. Storey, Roger H. L. Chiang, Cecil E. H. Chua, and Jon W. Beard

Communications of the Association for Information Systems

Special Section DOI: 10.17705/1CAIS.05037 ISSN: 1529-3181

Information Systems Analysis and Design: Past Revolutions, Present Challenges, and Future Research Directions

Keng Siau
City University of Hong Kong

Carson Woo
University of British Columbia

Veda C. Storey
Georgia State University

Roger H.L. Chiang
University of Cincinnati

Cecil E.H. Chua
Missouri University of Science and Technology

Jon W. Beard
Iowa State University

Abstract:

Systems Analysis and Design (SAND) is undoubtedly a pillar in the field of Information Systems (IS). Some researchers have even claimed that SAND is the field that defines the Information Systems discipline and is the core of information systems. The past decades have seen the development of Structured SAND methodologies and Object-Oriented Methodologies. In the early 1990s, key players in the field collaborated to develop the Unified Modeling Language and the Unified Process. Agile approaches followed, as did other dynamic methods. These approaches remain heavily employed in the development of contemporary information systems. At the same time, new approaches such as DevOps and DevSecOps continue to emerge. This paper curates these trends in SAND. It reviews past and present SAND research, discusses current challenges, and provides insights that can assist SAND researchers in identifying future SAND research streams and important future research directions.

Keywords: Abstraction, Agile, Conceptual Modeling, DevOps, Information Systems Research, Internet of Things, Modeling Methods, Systems Analysis and Design.

This manuscript underwent editorial review. It was received 9/20/2020 and was with the authors for ten months for two revisions. Jacob Tsai served as Associate Editor.

1 Introduction

Systems Analysis and Design (SAND) consists of methods for studying and understanding aspects of the real world that should be captured and represented in an information system. SAND is an indispensable foundation for information systems development (Bajaj et al., 2005) because it deals with the business-oriented and technical aspects of the problem environment and, when done well, successfully identifies and addresses relevant behavioral, cognitive, economic, organizational, and social issues. A variety of methods have evolved and developed in over six decades of computerization and automation, ranging from very structured approaches to more iterative, incremental, and adaptable techniques.

The successful implementation of information systems depends heavily on a thorough and well-executed SAND effort. However, this is much more difficult than anticipated! Information system (IS) failures have remained a common occurrence during the development of information systems (Sardjono & Retnowardhani, 2019; Odtadmin, 2019; Shen et al., 2018; Siau & Rossi, 2011; Siau et al., 2010; Avison & Fitzgerald, 2006; Siau & Tan, 2005c; Hardgrave et al., 2003; Schmidt, et al. 2001; Smith et al., 2001; Siau et al., 1997). The Standish Group, for example, reports that 83.9% of IT projects partially or completely fail, with the top factor in failed projects being incomplete requirements (Odtadmin, 2019).

Several major revolutions in SAND have occurred over the past decades. They include the emergence of concepts such as Flow Charts, Data Flow Diagrams (DFD), Entity Relationship Diagrams (ERD), Object-Oriented modeling (OO), Unified Modeling Language (UML), Unified Process (UP), Agile Modeling (AM), and DevOps. One revolution involves the movement from structured systems development to an object-oriented approach (Armstrong & Hardgrave, 2007) triggered by a more-or-less wholesale move towards object-oriented programming in the 1980s. Another revolution is forward engineering, whereby models developed can be automatically translated into programming code. While the goal of complete executable modeling remains tantalizingly elusive, progress has made it possible, in some cases, to make more than 40-50 percent of the modeled code executable. The application of techniques from artificial intelligence may increase the degree of automation. Advances in systems development have improved our ability to reuse codes rather than always writing new codes. Component-based software engineering is another step in the SAND revolution (Zhao & Siau, 2002). Web services or service-oriented architectures (SOA) (Erickson & Siau, 2008) support application development by combining services. In addition, new ideas and emphases such as Agile (Erickson et al., 2005), DevOps, extreme programming, agent-oriented approach, and cognitive modeling (Siau & Tan, 2005a, 2005b; Wei et al., 2006), within the context of SAND have assumed increasing importance for academics and practitioners. Continuing research in these areas, especially for large systems development projects and dynamic and fast-evolving web services, remains a high priority. The ultimate goal is to enhance systems development success.

This paper curates the revolutions within the area of systems analysis and design. In addition to providing the historical development of SAND, we chronicle and analyze the SAND methodologies. Doing so enables us to demonstrate how SAND methodologies have evolved to keep pace with advancing information technologies, and we expect this trend to continue. We also identify patterns that SAND methodologies exhibit: a higher-level abstraction of concepts; and business or ecosystem abstraction.

The remainder of the paper is organized as follows. First, the paper categorizes the different SAND revolution and evolution eras and presents the key SAND methodologies in each. Second, in reporting on the main developments within each era, the paper drills down to highlight the contextual circumstances that prompted and/or resulted in such revolutions and evolutions. Third, the paper discusses some of the current challenges facing SAND. Finally, the paper presents future research directions for SAND.

2 Classification Schemes for Curating and Research

To comprehend the revolutions and evolution of the concepts and methodologies of SAND, we organize and curate SAND development over six decades from several perspectives, as reviewed below.

2.1 Systems Development Concepts

Hirschheim and Klein (1989) presented the concept of systems development paradigmatic thinking by creating the "four paradigms of systems development." They describe the first paradigm as functionalism, in which systems development is driven from the outside, using formal and well-defined plans and tools. The elements of each system are viewed as physical entities, and the structured methodologies are

examples of this paradigm. Their second paradigm is "social relativism," which views systems development as happening from the inside. Entities and structures are considered to be changing, dynamic, or evolutionary. Various ethnographic systems development methodologies are examples of this paradigm. The third paradigm is radical structuralism, which emphasizes the need to transcend the limitations placed on existing social and organizational arrangements. This paradigm underlines the structure and analysis of economic power relationships. The last paradigm, neohumanism, seeks radical change, emancipation, and potentiality. This paradigm stresses the role that different social and organizational forces play in understanding change, as well as their importance in the information systems development process.

Hirschheim and Klein's (1989) concept of systems development paradigmatic thinking was further developed in Iivari, Hirschheim, and Klein's (2001) framework that consists of paradigms, approaches, methodologies, and techniques. In the social sciences, the term "paradigm" is usually reserved to describe the basic assumptions underlying coexistent theories (Burrell & Morgan, 1979). Within the context of SAND, examples of paradigms are Waterfall and Plug-and-Play. An approach can be considered in terms of the basic principles, goals, and concepts that anchor how systems development is understood and developed. Examples are the Structured Approach and Object-Oriented Approach. Methodologies, which are composed of specific procedures, are closely related to the more general and goal-driven approaches such as the Unified Process. Methodologies are used to guide information systems development. Finally, techniques can be considered as "well-defined sequence(s) of basic operations." Examples of techniques are a Class Diagram and Use Case Diagram. If the techniques are properly completed, they can lead to specific (and measurable) results. One issue with this framework is that it is often not easy to clearly distinguish between paradigms, approaches, methodologies, and techniques. Most researchers and practitioners use these definitions loosely.

2.2 Systems Development Methodologies

In this paper, we adopt a more commonly used definition of methodologies: a methodology is a systematic approach to information systems planning, analysis, and design (Rowley, 1993; Avison & Fitzgerald, 2006). Methodologies offer a series of phases and steps through which a project must proceed and a series of tools and techniques to assist in analysis and design (Rowley, 1993).

Another way to classify SAND research is to consider the underlying philosophies of the approaches, methodologies, or techniques. Hirschheim, Klein, and Lyytinen (1995) distinguished between ontological and epistemology perspectives. Ontologies are ways to classify the world in terms of its unchangeable, foundational, and universal structures. The world of ontologies can be further decomposed into realism and nominalism. Whereas realism proposes that a set of absolute laws and structures underlies the universe, the nominalism perspective posits that there is no absolute set of laws and structures and that those that exist are created by humans via social networks and structures. The Bunge-Wand-Weber (BWW) ontology is an example of philosophy in the SAND field (Wand & Weber, 1993, 2017; Burton-Jones et al., 2017; Lukyanenko et al., 2021).

The epistemology perspective of the world proposes to set a basis for what constitutes knowledge, how new knowledge is acquired, and what investigations into the world might be as well as how they should be conducted. The two endpoints of the epistemology dimension are Positivism and Interpretivism. Positivism proposes that the scientific method can be used to explain relationships between entities in terms of their causes and to discover the universal truth underlying the world. Interpretivism assumes that there are no absolute truths.

Avison and Fitzgerald (2006) classified systems development methodologies into different time-based eras, in which popular methodologies reflected the state of the art of systems development within the general time frame or era. They described the 1960s and 1970s as the Pre-Methodology Era. In the Pre-Methodology Era, attention was placed on technical issues and hardware limitations. Analyzing the business needs underlying the development effort was nearly always secondary during this era. The Early Methodology Era was the period between the late 1970s and early 1980s. The Software/Systems Development Life Cycle (SDLC) is a well-known approach of this time during which systems development shifted from hardware and technical constraints to the process itself. Although the process was receiving more attention, identifying business needs was still not receiving adequate emphasis and focus. The Methodology Era, encompassing the late-1980s through the late-1990s, saw a proliferation of methodologies in a variety of genres. The methodologies were (more or less) aimed at ameliorating the deficiencies of SAND approaches in the earlier eras. Finally, from the late-1990s to the present,

developers have gradually realized that strict adherence to any given methodology, no matter how efficacious it might have been in the portrayal of other projects, does not guarantee the success of the next project in which it is used. This is the start of the Post-Methodology Era.

2.3 Classification Scheme

Avison and Fitzgerald's (2006) time-based analysis of information systems analysis and design methodologies provides a good outline to understand the revolutions and evolutions of the methodologies over time. In this research, we use Avison and Fitzgerald's (2006) time-based eras as one dimension of our curation, with Iivari, Hirschheim, and Klein's (2001) framework consisting of paradigms, approaches, methodologies, and techniques as part of the second dimension. Several other important features are added to the second dimension to provide the Contextual/Situational factors.

Our classification is significantly different from Avison and Fitzgerald's (2006) in that we added a second dimension, the Contextual/Situational (i.e., paradigms, approaches, methodologies, techniques, IT focuses, IT impacts, IT scope, etc.) to their time-based dimension (Pre Methodology era, Early Methodology era, Methodology era, and Post Methodology era). Using Avison and Fitzgerald's (2006) time-based eras as one dimension and expanding on Iivari, Hirschheim, and Klein's (2001) framework as a second dimension helps to distinguish the progress of the SAND field based on various Contextual/Situational factors. The two-dimensional table enables us to document the accumulation of knowledge at a finer level and facilitates the identification of future research directions.

Further, instead of having one Methodology era, we divide the Methodology era into two separate eras – a Methodology Proliferation era and a Methodology Standardization era. In Avison and Fitzgerald's (2006) classifications, they mentioned that "Methodologies can be classified into several movements. The first are those methodologies designed to improve upon the traditional Waterfall model. The second movement is the proposal of new methodologies that are somewhat different to the traditional Waterfall model (and from each other)" The term "YAMA"—Yet Another Modeling Approach—was coined to describe the proliferation of modeling methodologies during the Methodology Proliferation era. Nevertheless, the Methodology era also saw the development of standards that synthesized and integrated many approaches, methodologies, and techniques developed before and during the era (e.g., Unified Modeling Language, Business Process Model and Notation). Because these standards integrate the ideas of the Methodology era, it is more meaningful to classify them as a separate era (i.e., the Methodology Standardization era) and differentiate it from the era where many different approaches, methodologies, and techniques were proposed (i.e., Methodology Proliferation era).

Finally, when Avison and Fitzgerald wrote their paper in 2006, they said, "it is not at all clear how [the Post Methodology era] will pan out" Sixteen years later, we have a lot more to consider and digest. In addition, newer technologies developed after 2006 (e.g., blockchain and Internet of Things) can potentially disrupt organizations and societies at a different magnitude than those technologies before 2006. However, the ad hoc and reactive approach to proposing methodologies (mentioned as a potential trend by Avison and Fitzgerald (2006)) will not be able to keep up with the pace of new technologies being adopted by organizations and societies. In this regard, the second dimension used in this paper to complement the methodological eras dimension by Avison and Fitzgerald becomes critical because it allows us to learn from the past eras in an organized and systematic manner to identify future research directions.

3 Evolutions of Systems Analysis and Design

This section highlights major benchmarks and trends in SAND. The needs and demands of SAND can be understood by using a historical timeline of the evolution of the challenges related to developing computer software and related hardware and networking capabilities,identifying user requirements, and satisfying information needs. The usage of information systems in organizations and the impact of information systems on organizational performances affect the revolutions and evolutions of SAND methodologies.

Table 1 summarizes the important methodological approaches that emerged during the different eras of SAND due to the advancement in technical capabilities and/or developers'experience. Table 1 also depicts the increased business focus of information systems in organizations as information systems extended their support from the operational level to the strategic management level and expanded from intra-organizational systems to inter-organizational systems to social media systems.

Table 1. Eras of Information Systems Analysis and Design Methodologies

	Pre-Methodology Era (1960s to 1970s)	**Early Methodology Era (1970s to early-1980s)**	**Methodology Proliferation Era (late-1980s to mid-1990s)**	**Methodology Standardization Era (mid-1990s to early-2000s)**	**Post Methodology Era (late-1990s to present)**
Paradigm	Computational problems or Calculations	Centralized, Waterfall	Decentralized, Plug-and-play	Decentralized, Plug-and-play	Dynamic, Flexibility, Adaptability
Approaches	Hardwired	Structured	Object-oriented	Object-oriented, Agile	Agile, DevOps, DevSecOps
Methodologies	Documentation, Algorithm	Software Development Life Cycle (SDLC)	Unified Process	Unified Process	Scrum, Extreme Programming
Data/File Structures	File System	Hierarchical, Network Database	Relational, Entity-Relationship, Object-oriented	Relational, Object-oriented	Relational, In memory, Vertical, Web-based, Data warehouse/Data lake, NoSQL
Techniques	Flowchart	ER Model, DFD	Object-oriented methodologies (e.g., OMT, OOSE, FUSION)	UML, BPMN	Coarse-grained conceptual model
Goals of IT Systems	Efficiency	Effectiveness	Effectiveness, Strategy	Effectiveness, Strategy, Value creation	Strategy, Value creation, Societal impacts
Information Technology Capability Driver	Mainframe "centralized" processing	Minicomputer, mostly centralized processing	Microcomputer, Client-server, Decentralized, and Distributed processing	Client-server, Decentralized, and Distributed processing	Internet, Internet of Things, Social Media, Cloud Computing, Off-the-shelf software
Applications and Information Focus	Application-specific	Data and User-driven	Business and Knowledge management driven	Business and Knowledge management driven	People, Relationship, Security, User experience, and Data exchange driven
IT Development Justification	Return on investment (ROI)	Increase productivity, Higher-quality decision	Competitive advantage, Strategic positioning	Competitive necessity, Strategic positioning	Value addition, Relationship Enhancement, Automated information interchanged, Internet as a platform
IT Development Scope	Organization	Organization, Group, Individual	Business Processes	Business Processes	Customer, Employee, Supplier ecosystem, Intelligent Devices, Inter-organizational

3.1 Pre-Methodology Era

The concept of SAND dates to the earliest days of computer hardware and software. SAND's origins emerged from several sources whose ideas were consolidated. The earliest incarnation of SAND emerged from engineering fields as computer engineering developed and evolved.

Early computer systems were large machines with centralized processing. Initially, these physically large computers were often special-purpose devices with limited computational capabilities that focused on specific types of problems, such as complex mathematical or engineering problems and census tabulation and processing. These computer systems dealt with computationally expensive types of problems with large numbers or many significant digits, or they focused on business-related types of calculations (Campbell-Kelly & Aspray, 1996).

During the Pre-Methodology Era, SAND required a detailed understanding of the algorithm. This requirement was necessary both to logically organize the steps required to solve the problem and also to 'hardwire' (cf. ENIAC) the program into the computer (i.e., physically rewiring the data paths and computational operations before the program was executed), which required its own set of detailed knowledge about the logical design of the computer. Once one program was completed, the wiring was reconfigured as appropriate to set up (i.e., program) for the next procedure, followed by program execution. Computer operations were essentially one program at a time, as opposed to a set of interrelated programs (an information system). Thus, flowcharting was a popular technique, and data were stored as file structures with the program. The areas of emphasis were efficiency for the specific problem type and the return on investment in terms of cost savings for an organization.

Technology improved rapidly. These types of task-specific machines were soon replaced by general-purpose computers that were flexible enough to be programmed for almost any type of computation or calculation. This general-purpose nature added another layer of complexity that required an additional understanding of the details of the types of problems to be executed.

The processing power of the computers increased as the capabilities of the hardware improved, including the addition of memory (initially to hold input data and output, and subsequently to also hold programs) (Campbell-Kelly & Aspray, 1996; Ceruzzi, 2003). Although not articulated until 1965, Moore's Law captured the essence of the rapid improvement in computing speed and capabilities (Moore, 1965). In these early years of computing, SAND remained a relatively straightforward process focused on individual programs for specific organizations, although these programs could be quite complex.

3.2 Early Methodology Era

By the early 1970s, computers were more powerful with increased computing power (processing speed and larger storage capacities), and minicomputers started to emerge. In response to these more capable, flexible machines (cf. the UNIVAC computer), more ambitious software projects were attempted with engineering-, scientific-, and business-focused software solutions. Procedural programming languages emerged (cf. FORTRAN, ALGOL, and COBOL, followed by PL/I in the 1960s). SAND became more formalized as the demands for the software grew more complex and sophisticated to capitalize on the expanding capabilities of the hardware. Data started to be centralized rather than specific to a particular program and network. Hierarchical and network databases emerged so that different groups, and even individuals, could share the data to enhance productivity and increase decision-making quality.

The mid-to-late 1970s saw the development of much larger software environments with interrelated software components, the expectation of data-sharing across multiple computers, and the emergence of real-time processing. For example, SAGE (Semi-Automatic Ground Environment) was an ambitious multi-year effort to develop a computer-based air-defense system that could integrate and synthesize radar data for tracking unknown (possible enemy) aircrafts entering the U.S. airspace. It was built from many design and development lessons learned with Whirlwind, an MIT-based project (Astrahan & Jacobs, 1983). While ultimately limited in its capabilities and not sufficiently advanced to track the nuclear-tipped ballistic missiles that became prevalent among the superpowers in the 1970s and 1980s, SAGE became an exemplar of a systematic approach to developing large-scale information systems (Hughes, 1998).

Bennington (1983) noted several factors that contributed to the general success of SAGE that has important implications for the development of SAND. First, the development team was comprised of engineers who organized their thinking and design with a strong formal and detailed structure. Many systems development efforts were in government or military procurement, which were some of the most

significant and demanding clients of the era. They developed expectations that all software projects proceed in the same way irrespective of whether one is manufacturing spacecraft or boots (Bennington, 1983, p. 352). Further, there was a consensus that programs needed to be "rationalized"-- meaning that requirements were clearly defined and tasks carefully planned; detailed documentation was created, produced, organized, and archived; human interfaces were designed; substantial testing tools were created and executed; and detailed record-keeping to trace problems and track fixes were generated. This structured and systematic approach was one part of the early foundation for what came to be known as the Waterfall Methodology, a term first used by Bell and Thayer (1976). During this period, early structured methodologies, such as Data Flow Diagram (DFD) (Gane & Sarson, 1977), Entity-Relationship Diagram (ERD) (Chen, 1976), and others (e.g., DeMarco, 1979; Yourdon, 1979, 1988; Page-Jones, 1988), were proposed to facilitate the system development process.

3.3 Methodology Proliferation Era

The Methodology era, as used by Avison and Fitzgerald (2006), saw a flurry of methodological innovations and activities. Numerous methodologies were invented or created during this period (in the spirit of "let a thousand flowers bloom!").

The late 80s and early 90s saw the shift from structured programming and approaches to object-oriented programming and thinking. This shift resulted in the creation and proliferation of object-oriented methodologies. A relevant subset of the object-oriented SAND methodologies is listed in Table 2. The object-orientated approach is different from the structured approach because object-orientation is based on the concept of "objects" that contain both code (i.e., procedures) and data (i.e., fields). The object-oriented movement also impacted the data/file structure. Although relational databases remained the prominent databases at that time, object-relationship, and object-oriented databases were discussed and proposed. Early object-oriented databases included Gemstone (GemStone Systems), Gbase (Graphael), and Vbase (Ontologic). The SAND methodologies in this era needed to account for technology development and incorporate business needs. Microcomputers, client-server computing, and distributed processing were emerging technologies. With microcomputers and personal computers, the development scope moved to the business-process level. Instead of focusing on efficiency and productivity as in the earlier eras, businesses in this era also expected the information systems to be effective and to address the strategic needs of the organizations.

Table 2. Examples of Object-Oriented Methodologies

Name	Reference
Stroustrup Methodology	Stroustrup (1988)
Colbert Methodology	Colbert (1989)
Wasserman Methodology	Wasserman et al. (1990)
Responsibility Driven Design (RDD) Methodology	Wirfs-Brock et al. (1990)
Coad/Yourdon Methodology	Coad et al. (1991)
Embley Methodology	Embley et al. (1991)
EVB Methodology	Jurik & Schemenaur (1992)
Business Object Notation (BON) Methodology	Nerson (1992)
Shlaer and Mellor Methodology	Shlaer & Mellor (1992)
Berard Methodology	Berard (1993)
Booch Methodology	Booch (1993)
de Champeaux Methodology	De Champeaux et al. (1993)
Object Oriented Software Engineering (OOSE) Methodology	Jacobson (1993)
FUSION Methodology	Coleman et al. (1994)
Martin/Odell Methodology	Martin & Odell (1994)
Object Modeling Technique (OMT) Methodology	Rumbaugh et al. (1994)
ROOM Methodology	Selic et al. (1994)
OOram Methodology	Reenskaug (1996)
Meyer Methodology	Meyer (1997)
HOOD Methodology	European Space Agency (2006)

3.4 Methodology Standardization Era

YAMA (Yet Another Modeling Approach) is a term used to describe the flurry of activities that happened in the Methodology Proliferation era, which resulted in numerous object-oriented methodologies. As a result of this methodology explosion, calls were put forth for a unified approach. UML (Unified Modeling Language) was jointly developed by Grady Booch, Ivar Jacobson, and James Rumbaugh at Rational Software (a subsidiary of IBM since 2003) in 1994–1995. UML is intended to be a general-purpose and developmental modeling language. There are three types of UML diagrams:

- Behavior diagrams – A group of diagrams that depicts behavioral and dynamic features of a system or business process. The diagrams in this group are activity, state machine, and use case diagrams, as well as the four interaction diagrams.
- Interaction diagrams – A subset of behavior diagrams that emphasize object interactions. This group includes communication, interaction overview, sequence, and timing diagrams.
- Structure diagrams – A type of diagrams that depicts the elements of a specification that are irrespective of time. This group consists of class, composite structure, component, deployment, object, and package diagrams.

UML has been extended several times over the years and has become the Object Management Group's (2020)(OMG) standard. UML 2.5 was released in June 2015.

In addition to UML, the Business Process Model and Notation (BPMN) approach is becoming the leader and de-facto standard in business process modeling. BPMN is a graphical representation for specifying business processes in a business process model. The main difference between UML and BPMN is that UML is object-oriented whereas BPMN takes a process-oriented approach that is more suitable within a business process domain. The latest version is BPMN 2.0.2 and was published in January 2014.

In 1999, Ivar Jacobson, Grady Booch, and James Rumbaugh also came up with the Rational Unified Process (RUP) (Jacobson et al., 1999). It was developed by Rational Software and served as a supportive process framework for contemporary software engineering. RUP, as an adaptable and iterative software development process, is not prescriptive. RUP's emerging process is tailorable with tools that help to automate the development process and services. RUP's features include iterative development, requirements management, component-based architecture, visually modeling, constant quality verification, and change management and control. The RUP was later termed the Unified Process (UP). UP divides a project into four phases: inception, elaboration, construction, and transition.

A unified approach has its advantages. Instead of developing more and more similar methodologies and naming them differently, the Methodology Standardization Era saw joint efforts at improving and extending the unified approaches. For example, the OpenUP and Agile UP are different versions of UP. The Open UP is a part of the Eclipse Process Framework (EPF), an open-source process framework developed within the Eclipse Foundation. Agile UP (AUP) is a simplified version of UP developed by Scott Ambler. The Methodology Standardization also has the benefit of focusing the organizations' effort on enhancing the effectiveness of the business processes rather than spending time determining the best SAND methodologies to choose and adopt.

3.5 Post Methodology Era

In the late 1990s and early 2000s, the growing popularity of the Internet and rapidly changing business environment meant that businesses had to focus on strategy, value creation/addition, and even societal impacts as they are increasingly connected to customers through information systems. These evolving needs pushed the scope of systems development to shift to an increasing emphasis on the customers and users of the system. Therefore, SAND methodologies needed to be responsive to these different and evolving needs. Dynamic and flexible approaches became prevalent and necessary. The concept of "agility" in information systems development emerged as represented by the Agile approaches (Conboy, 2009).

3.5.1 Agile Approaches

Boehm and Turner (2004) noted that SAND approaches could be placed on a continuum that ranges from "adaptive" to "predictive" in their methodologies and outcomes. Agile approaches fall at the adaptive end

of what some characterize as a 'rolling wave' of project planning and development. On the other hand, the Waterfall approach is conceived as being predictive.

During the 1990s, approaches were emerging in response to what many regarded as too detailed, over-regulated, excessively documented, and micro-managed approaches of the Waterfall or other heavyweight approaches that were common at the time. Instead, methodologies with an emphasis on iterative and incremental methods were expanding in use during the 1990s, including Rapid Application Development (RAD) (1991), the Unified Process (UP) (1994), and Extreme Programming (XP) (1996), Scrum, probably the best-known and currently among the most popular Agile approaches, originated in 1995, with the first recorded use of the term in 1986 (Schwaber & Sutherland, 2017). Similar changes toward iterative and incremental (i.e., Agile) development were underway in manufacturing and aerospace.

The Agile Manifesto (Beck et al., 2001), written by seventeen software 'anarchists,' consolidated the thinking behind the movement. Consisting of four core software development values and 12 development principles, Agile represented the movement toward more 'lightweight' methods of software development that emphasized dynamic, iterative, incremental, and non-deterministic thinking with more substantial involvement of the users. The four broad values emphasize individuals and interactions rather than processes and tools, the completion and delivery of working software even if the full project is not yet complete, significant collaboration with customers, and responding to evolving or emerging requirements instead of a predictive plan. Although formally codified in 2001 with the Agile Manifesto (Beck et al., 2001), elements of the Agile approach have been around since the late 1950s. For example, Weinberg (as quoted in Larman & Basili (2003)) noted the use of incremental and iterative approaches in programming work in which he was involved in Los Angeles in 1957. According to his recollections, most people on these projects "thought waterfalling of a huge project was rather stupid" and that mentality helped us to "realize that we were doing something else."

Also developed in the 1990s, Crystal is an extremely lightweight, adaptable approach to software development. Its features were derived from research on best practices of successful teams. A basic tenet of Crystal is that each project may require policies, practices, and processes that are tailored to its unique needs. Furthermore, teams also may have different needs based on team size, project criticality, and the urgency of the project. Other Agile frameworks often have a more fixed structure. Crystal consists of a family of agile methods, such as Crystal Clear, Crystal Orange, and Crystal Yellow. Some of the common features for Crystal are frequent delivery, reflective improvement, co-location (to support 'osmotic' or an easy flow in the communication), personal safety, a focus on work, access to appropriate tools, and access to subject-matter experts and users. By reviewing the most recent SAND-related articles published by AIS-sponsored conferences (AMCIS, ECIS, ICIS, and PACIS) in the past five years (2016-2020), Agile is a dominant topic that has been studied.

Extreme Programming (XP) is another framework for Agile software development and is the most specific about the use of appropriate engineering practices. Its emphasis is on producing higher-quality software and a higher quality of life for developers. Four specific characteristics are most important: working with dynamic (i.e., changing requirements); dealing with the risks of projects with fixed deadlines; using small, co-located teams; and using technology that supports automated unit and integration tests of the software. Five values focusing on the human side of development are also important – communication, simplicity, feedback, courage, and respect. XP's specificity on the engineering practices may make it a less desirable approach for some teams or projects. Its structure is in almost direct opposition to the approach used in Crystal.

Lean software development is the interpretation of lean manufacturing principles adapted from Toyota's Production System approach and applied to software development (Poppendieck & Poppendieck, 2003). It is a reliable and well-understood conceptual framework that is derived from experience with values, principles, and identified good practices. The Agile community has willingly adopted 22 Lean principles. The main principles of lean software development are: to eliminate waste, amplify learning, decide as late as possible, deliver as fast as possible, empower the team, build integrity into the product, and optimize the whole.

3.5.2 DevOps Approach

Two of the pillars of the Agile approach are an increase in "interactions" and "customer collaboration" during the development process between the developers and the users (Beck et al., 2001). However,

while the Agile approach supports continuous integration of components as they are developed, there is minimal emphasis on the final implementation (i.e., 'go-live' transition) and maintenance (or Operations and Maintenance) of the system (Ambler, 2009; Leite et al., 2019). Yet the maintenance phase of the Systems Development Life Cycle is typically the longest (Dennis et al., 2014). Many software-development approaches do not actively consider operational concerns (Roche, 2013). To address this deficiency, the concept of DevOps (Development + Operations) has emerged as an additional critical approach for Agile software development, where there is an added emphasis on implementation and the operations of high-quality software that can quickly contribute to business results (Bhat & Herschmann, 2020; DeGrandis, 2011; Spafford & Herschmann, 2019). Based on the definition of a methodology we use in this paper, DevOps, in its current form, is not a methodology but an expansion of the agile approach. Instead, DevOps is characterized as a(n) practice, paradigm, approach, method, discipline, mindset, or philosophy (Jabbari et al., 2016; Lwakatare et al., 2019). Others see it as an extension of agile and Scrum (Erich et al., 2017). Because of its importance in information systems development, we expect DevOps to evolve into a slightly more systematic approach (although not to the extent of traditional methodologies). We, therefore, include DevOps in this paper as one of the major movements in the Post Methodology era. Its focus on strategic emphasis, value creation, and value addition, as well as being people- and relationship-driven, also puts it in the Post Methodology era.

DevOps is often defined as "a set of practices intended to reduce the time between committing to a change to a system and the change being placed into normal production while ensuring high quality" results (Bass et al., 2015). The speed of delivery and integration—continuous integration, often occurring several times a day, requires designing for operations and maintenance from the earliest phase of product inception (Dennis et al., 2014; Svensson et al., 2015). At the macro level, successful DevOps requires an organizational shift in the development culture through the use of broad cross-functional teams that bridge all business (i.e., users) and IT roles. DevOps is a community approach (Erich et al., 2017; Hemon, et al., 2019; Sánchez-Gordón & Colomo-Palacios, 2020; Wiedemann et al., 2019). This change suggests opportunities to improve an organization's understanding of how the development community becomes aware of and understands problems, requirements, how work progresses, who possesses what knowledge, and whether and how people share knowledge. Mao et al. (2020) noted that "it is crucial that software teams have ownership and responsibility to deploy software changes" (p. 450).

The DevOps movement is evolving rapidly. As Tomas, Li, and Huang (2019) noted, a security culture also needs to be cultivated because security education is often lacking, security is too often ignored, and security assessments are often neglected. DevSecOps (Development + Security + Operations) is an extension of DevOps (Sánchez-Gordón & Colomo-Palacios, 2020), where security practices also become the development team's responsibility instead of being the sole responsibility of a testing-focused group. Including security issues in the earlier phases in the development process within the cross-functional teams allows the cooperation of business (i.e., users), information systems, and security groups (Myrbakken & Colomo-Palacios, 2017). The goal is to create a higher level of awareness of and proficiency in security issues for the rapidly changing software product.

In the Post Methodology era, the speed of software design and development is being driven by surging demand for digital capabilities to provide organizations with opportunities to accomplish strategic goals and better meet and create customer demand(s). The Agile approach to software development sought to create closer and more involved collaboration between the developers and users. Agile also worked toward more rapid delivery and continuous integration of the software, but this has often been hindered by the lack of involvement of those doing the implementation and maintenance work until product delivery. DevOps evolved to enhance the involvement of the developers with the operations specialists responsible for these end-phase activities to improve system design for continuous delivery into the large IT ecosystem. Further, DevSecOps emerged to extend these benefits by explicitly incorporating security into the earliest phases of design as software has become an increasingly important part of systems development and operation. Also, with the growing importance of user experience, especially in social media and online environments, DevUXOps may be the next extension. Agile, DevOps, DevSecOps, and DevUXOps aim to provide greater "agility" in developing software that is more rapid to build, more flexible to use, and takes into account issues that are currently mostly an afterthought (e.g., security, user experience).

4 Discussions

The field of SAND has developed over six decades, with several eras of revolutions and continuous evolutions. In this paper, we classified these eras into Pre-Methodology, Early Methodology, Methodology Proliferation, Methodology Standardization, and Post Methodology. Research results from the academic world have been adopted by practitioners. Practitioners, in turn, have developed their own methodologies, which were later studied by researchers. Some practitioners, for example, are starting to take the 'best' of both the Waterfall and Agile approaches; also known as 'Watergile' or 'Aquagile.' They often include the structure, documentation, and 'big-picture' view of Waterfall merged with the incremental, iterative, and rapid-delivery approach, especially for systems with ill-defined, evolving, or emerging requirements. There are, thus, many ways in which continued and further integration of academic research and practice is both important and challenging.

In this section, we examine the different SAND methodology eras as summarized in Table 1 to derive our insights into how the SAND field is moving forward. The two insights are (1) moving towards a higher level of abstraction and (2) moving towards a business-level (or even society-level) abstraction. We then use three examples to illustrate the various stages of this movement. We use the Work System Method (Alter, 2006) to illustrate how higher-level business level abstraction can be accomplished. We use the current popular DevOps approach to illustrate what can be done. Finally, we use the Internet of Things (IoT) to illustrate what might be forthcoming.

4.1 Moving Towards a Higher Level of Abstraction

From examining the SAND methodology eras, some common themes arise. As technology advances, its capability increases in magnitude, resulting in organizations wanting more of it. The technological advancements in each era are facilitated by abstraction. Abstraction provides a general representation of the details without sacrificing an overview of the system to be built and is a mechanism to manage complexity. Different eras provide different abstractions. For example, in the Pre Methodology era, abstraction is provided by algorithms. Designers are not concerned with how the programming is done but rather want an overview of the programming. In the Early Methodology era, the abstraction is on the flow of data, as captured in data flow diagrams, without being concerned with how the data is stored and how a program is going to be written. This is a higher-level abstraction than an algorithm because the software has more capabilities in that era. In the Methodology Standardization era, the abstraction is on the business process (e.g., BPMN), which focuses on the needs of businesses without having to be concerned with data. It should be clear that, as time progresses, technology can do more, and organizations expect more. Therefore, to manage the complexity of a larger system, a higher level of abstraction is needed.

Nevertheless, the introduction of new technology does not imply the need to replace existing systems. Many traditional applications continue to be important and well-used. These include airline reservation systems, database systems, enterprise systems, Internet-based systems, and AI applications, with a recent surge in the latter. As our dependence on these systems continues to rise in our society, well-designed and well-executed systems are critical for building a larger future ecosystem. Given the complexity of these future ecosystems, some forms of independence are needed. We, thus, predict the return of object-oriented analysis. It might not be the same as those in the 90s. It might even be called by a different name (e.g., agent-oriented analysis, role-oriented analysis, service-oriented analysis, or resource-oriented analysis). Nevertheless, the concept of autonomous entities operating in a decentralized ecosystem, interacting with each other to accomplish their own goals, will be needed.

4.2 Moving Towards a Business and Ecosystem Level Abstraction

When business environments change, the scope of the systems being considered as important for supporting business strategies and success also change. For example, in the Pre Methodology era, computational problems or calculations are the focus. In the Methodology eras (i.e., Methodology Proliferation and Methodology Standardization), the focus is on business (especially business processes and business operations). Progressing to the Post Methodology era, instead of focusing on clearly defined business processes, businesses want to be able to react quickly to competition, necessitating SAND to move away from the structured and slow business process thinking to understand strategy and value. Even within an era, considerations usually start from a smaller scope and move to a larger one, sometimes following the business or the ecosystem's needs. For example, in the Post Methodology era,

the move to DevOps is not unique in information systems. For example, marketing research has often considered the customer life-cycle or customer lifetime value. Analogously, we can view DevOps as a customer lifetime value where the customer is the user of information systems, and the lifetime value is providing what these users want from existing information systems, via operations and maintenance, when business changes (which is the dynamic nature of the business ecosystem in this Post Methodology era). Using the same deduction, we predict that the scope in the future will be expanded to include resolving societal issues. We are beginning to see this trend. To reduce greenhouse emissions, we research smart cities. To sustain healthcare, we analyze how to integrate their interacting components such as hospitals, labs, pharmacies, and other patient care organizations. This is being done at a time when new technologies, such as the Internet of Things (IoT), are becoming more popular and can support such initiatives.

The object-oriented analysis resulted from the attempt to introduce more efficiency into systems development efforts. One area where object-oriented analysis can be enhanced is to address the different conceptualizations needed between the efficiency of development and the effectiveness of using systems in organizations. For example, a flowchart is an abstraction for systems development, whereas business process modeling is an abstraction at the business process and operation level. Research is needed to develop the equivalent of this abstraction at the business level. Further, there is the need to map, align, or convert the business level abstraction to the systems development level abstraction.

The recognition of these two levels of abstractions is important as new technologies are being developed. For example, Blockchain and the Internet of Things both have the systems-level abstraction but lack the business-level abstraction. SAND, given its history, is positioned to be the perfect field to develop this business-level abstraction and its mapping to the systems-level abstraction. This is because SAND can consider factors beyond technological feasibility when using technology to develop a new business application.

4.3 Example of a Higher-Level Business Abstraction – Work System Method

The work system method is a semi-formal method for analyzing and understanding IT-enabled systems in organizations (Alter, 2003a, 2003b). The work system method is based on three components of work system theory: the definition of work system, the work system framework, and a work system life-cycle model (Alter, 2006, 2013). A work system is a system in which human participants and/or machines perform work (processes and activities) using information, technology, and other resources to produce specific products and/or services for specific internal or external customers. Information systems are special cases of work systems. An information system is a work system whose processes and activities are devoted to processing information -- capturing, transmitting, storing, retrieving, manipulating, and displaying information.

In the work system framework, nine elements are identified as part of a rudimentary understanding of a work system. Four of these elements (processes and activities, participants, information, and technologies) constitute the core of the work system. The other five elements (environment, infrastructure, strategies, customers, and products and services) fill out a basic understanding of the situation. The work system life cycle model consists of four main phases (initiation, development, implementation, and operation and maintenance).

The major differentiation between the work system framework and traditional systems analysis frameworks is the higher level of abstraction at the business level in a larger scope of consideration. At the business level, for example, products and services need to be linked to processes and activities. And according to Alter (2006), it is necessary to distinguish between customers and participants and not combine them into a single concept called "users" because customers belong to the business level, whereas participants belong to the systems level. Traditional systems analysis focuses mainly on the information systems themselves. The work system extends this consideration outside the information systems. For example, a solution to a problem that originated from an information system can now be found outside the information system (e.g., in products and services).

Alter has continued his work on the Work System Method to develop a Work System Theory (WST) (Alter, 2013). He argues that "WST is the core of an integrated body of theory that emerged from a long-term research project to develop a systems analysis and design method for business professionals called the Work System Method (WSM)."

4.4 DevOps Approach: A Potential Higher-Level Business Abstraction

DevOps expands the scope of development to include operations and maintenance. This includes help desk support, automation of as much of the development process as possible, and the management of the information system. The management part requires an understanding of higher-level business management. There is a large body of research literature on maintenance and management (e.g., work by E. Burton Swanson in the 1980s and 1990s (see, Lientz & Swanson, 1980; Swanson, 1994), industry standards and best practices (e.g., ITIL), help desk and support services, literature on workflow and business process modeling (automation still needs coordination), and literature in product management. We can learn from product management about which features to release and in what sequence should they be released as there are usually a lot of features to maintain and add. This requires a great deal of integration and, obviously, is not a simple task. Abstraction of important concepts in these fields for DevOps needs to be researched, developed, and explored.

Similarly, the move to DevSecOps means the need to include an understanding of how security impacts the higher-level business abstraction needed for development and operations. DevSecOps triggers other similar considerations, such as user experience (UX) that, similar to security, is usually an afterthought. Perhaps there should be DevUXOps or DevSecUXOps?

4.5 Example of Recent Technology that Needs Abstraction at Ecosystem Level – Internet of Things

Another rapidly emerging challenge for SAND is to support the analysis, design, and development of advanced technologies such as the Internet of Things (IoT) systems. In this paper, we use IoT as an example to illustrate the need for abstraction at the ecosystem level. For IoT, the abstractions may have to begin from the technical side. Several concepts are either unique to or more salient in IoT systems analysis and design than in other forms of systems analysis and design. Here, we discuss two main ones: (1) designing for automatons and (2) stream semantics.

Designing for Automatons. SAND has principally focused on managing human-driven systems. As an example, consider the concept of normalization (Codd, 1970; Codd, 1971; Codd, 1974; Fagin, 1977; Fagin, 1981). First Normal Form exists principally because humans have difficulty manipulating the concept of "many." But modern autonomous devices do not have this problem—they handle dynamic arrays well. Indeed, normalizing data in an IoT environment could be argued to be a poor design because it increases the risk of programmer error by making programming more difficult but does not facilitate the detection or management of errors caused by the autonomous device.

Humans tend to make more random errors, whereas autonomous devices tend to make more systematic errors. SAND problems in IoT often arise from difficulties modeling the physical world in the digital representation, i.e., digital twin (Bolton et al., 2018; El-Saddik, 2018). For example, we might model the location of the user as the geolocation of the user's mobile phone. However, the user and mobile phone are two distinct objects, e.g., it is possible for the user to have left the phone at the office and arrive home or for someone to have stolen the user's mobile phone.

In IoT systems, small differences between the physical world and the digital twin can have profound consequences, especially when those small differences trigger or fail to trigger actions in the digital twin. For example, Ibrahim Diallo lost his job when his manager was dismissed and failed to update Diallo's electronic paperwork (Wakefield, 2018). Because the paperwork had not been filed, a computer program then determined that Diallo's contract had not been renewed, canceled his security card, ordered security to escort him from the building, canceled his access to electronic systems, and so on.

The system behaved correctly as specified, but this resulted in negative consequences. Furthermore, the specification itself was correct—the failure lies in user error in that the terminated manager did not file the paperwork. However, in non-digital physical systems, it is unlikely that this would have happened. A quick phone call to HR once the firing process was underway would have remedied and reversed the problem. IoT SAND research thus must also grapple with this new problem of designing systems to be self-aware enough to recognize that frantic attempts by system administrators and users to reverse computer activity indicate that the system is doing something wrong.

Stream Semantics. Many, if not most, IoT devices either produce or consume data streams. Even simple IoT devices, such as IoT switches, are stream-based. When the switch is "on," it generates a steady

stream of electricity -- it generates a stream of 1s when it is on, and when it is off, it generates a steady stream of 0s. A pushbutton switch is identified by the fact that it sends a stream of 0s, a stream of 1s, and then a stream of 0s. Most electronic sensors are stream-based. For example, an IoT thermometer will generate a continuous stream of temperature readings.

Almost all systems analysis and design efforts are based on the concept of states and transitions between them. For example, in UML, execution behavior is captured in the UML activity diagram and state machine diagram. Streams, however, are not single states but recurrent patterns of states. Further, we are often not interested in the individual values of the data stream itself but rather the trends and transitions in the data stream. Thus, to represent and capture a more complex set of data requires a recognition of the difference and different methods of representation in the analysis and design of IOT systems.

Trends. Individual values in a stream are meaningless. It is not particularly useful to know that a thermometer is currently measuring 37.5 degrees Celsius because, in the very next microsecond, it may be measuring 37.4 degrees Celsius. Instead, what is valuable is to know the trend, i.e., is the trend rising, falling, or remaining stable? Stream trends are similar to traditional aggregate descriptive statistics (e.g., sum, minimum, maximum, average, standard deviation). However, most of them have "blocking" properties (Raman et al., 1999). That is, the calculation of these statistics requires knowledge of all data points, but these cannot be obtained in the stream (e.g., Chen & Chen, 2020).

Transitions. In addition to stream trends, in the IoT systems, we are often interested in stream transitions. Consider pushing a button to activate a lightbulb. The designer is not interested in the stream of 0s indicating 'off' or a stream of 1s indicating 'on' but in the transition from 0 to 1 and then back to 0 -- that is when the button is pushed and released. Stream transitions are not necessarily cleanly identifiable. For example, a specific proximity sensor could detect a human presence when the pattern changes from a stable capacitance of 100-300 to a stable capacitance pattern above 1000. A second proximity sensor has other low- and high-range values. When there is a human presence, it must be calibrated.

Existing state-based SAND techniques do not have a vocabulary to express concepts of streams such as trends and transitions. This makes it challenging to specify the behavior of IoT devices using traditional information systems models. SAND methodologies in the Post Methodology era might need to be adapted to address SAND issues for these advanced technologies.

5 Conclusions

This paper has identified the progression of systems analysis and design, emphasizing the major developments in the different eras that have defined this field. Starting from general algorithms and the need to provide structure to systems implementation, SAND has progressed to modeling complex, advanced, and emerging technologies, such as the Internet of Things. Not only do SAND methodologies need to evolve to keep pace with advancing IT technologies, but they also need to address changing business needs. Such revolutions and evolutions of SAND methodologies will continue.

In analyzing the progression, we identified a pattern. SAND methodologies are moving towards (1) higher-level abstraction and (2) business- or ecosystem-abstraction. Both are due to the need to manage the complexity of wanting to achieve more with a faster speed. We illustrate using three examples at different stages of this movement. The Work System Method is a higher-level business abstraction. The DevOps approach has identified its scope (development and operations), and by understanding the pattern of movement to higher-level abstraction, we can identify the foundations in relevant fields (e.g., customer life-cycle in marketing, management issues in maintenance, and international standards for supporting help desk) to facilitate the development of SAND methodologies for DevOps and DevSecOps. The final example is illustrated through an emergent technology, the Internet of Things (IoT), where its potential is still expanding. Because the complexity of using IoT in an ecosystem (e.g., smart city) remains unclear, SAND's role will be to identify the necessary higher-level business/ecosystem abstraction to facilitate the exploration of IoT in an ecosystem. Nevertheless, we need to start by identifying the lower-level technical abstraction before moving to a higher-level one.

In addition to discussing SAND challenges for new technologies such as IoT and identifying future research directions of SAND, this paper extends the classification proposed by Avison and Fitzgerald (2006). First, the time eras are expanded where the Methodology era was divided into the Methodology Proliferation era and Methodology Standardization era. This extension is important because of the many SAND methodologies and activities that happened during the Methodology era. Further, the Methodology

era can clearly be distinguished into the Methodology Proliferation era and the Methodology Standardization era. Second, the classification proposed by Avison and Fitzgerald (2006) is one-dimensional, which is based on different time periods. In our classification, another dimension (i.e., a Contextual/Structural dimension) is added where circumstances and situations that prompted or resulted from the methodology revolutions and evolutions are depicted. This Contextual/Situational dimension allows us to discuss the factors initiating, impacting and resulting from the changing eras and the characteristics of technologies and business concerns in each era.

In summary, SAND methodologies have evolved and advanced to deal with evolving levels of abstraction and the complexity of the business problems being modeled. The field of SAND has succeeded in providing methodologies for systems analysis, design, and development while not ignoring the practical and theoretical implications. The future for SAND will involve the exploration of many research ideas and challenges that are highlighted in this paper.

References

Alter, S. (2003a). Sidestepping the IT artifact, scrapping the IS silo, and laying claim to "systems in organizations." *Communications of the Association for Information Systems*, *12*, 494-526.

Alter, S. (2003b). 18 reasons why IT-reliant work systems should replace "the IT artifact" as the core subject matter of the IS field. *Communications of the Association for Information Systems*, *12*, 366-395.

Alter, S. (2006). *The work system method: Connecting people, processes, and IT for business results*. Work System Press.

Alter, S. (2013). Work system theory: Overview of core concepts, extensions, and challenges for the future*. Journal of the Association for Information Systems*, *14*(2), Article 1.

Ambler, S. W. (2009). *The Agile Scaling Model (ASM): Adapting agile methods for complex environments. IBM Rational Technical Report.* Retrieved from https://www.researchgate.net/profile/Scott_Ambler/publication/268424579_Adapting_Agile_Methods_for_Complex_The_Agile_Scaling_Model_ASM_Adapting_Agile_Methods_for_Complex_Environments.pdf ; Accessed: October 10, 2020.

Armstrong, D. & Hardgrave, B. (2007). Understanding mindshift learning: The transition to object-oriented development," *MIS Quarterly*, *31*(3).

Astrahan, M. M. & Jacobs, J. F. (1983). History of the design of the SAGE computer—The AN/FSQ-7. *Annals of the History of Computing, 5*(4), 340-349.

Avison, D., & Fitzgerald, G. (2006). Methodologies for developing information systems: A historical perspective. In D. Avison, S. Elliot, J. Krogstie, & J. Pries-Heje (Eds.), *The past and future of information systems: 1976-2006 and beyond* (vol. 214, pp. 27-38). Springer.

Bajaj, A., Batra, D., Hevner, A., Parsons, J. & Siau, K. (2005). Systems analysis and design: Should we be researching what we teach? *Communications of the Association for Information Systems*, *15*, 478-493.

Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A software architect's perspective*. Addison-Wesley Professional.

Beck, A., Grenning, J., Martin, R. C., Beedle, M., Highsmith, J., Mellor, S. van Bennekum, A., Hunt, A. Schwaber, K. Cockburn, A. Jeffries, R., Sutherland, J., Cunningham, W. Kern, J., Thomas, D., Fowler, M. Marick, B. (2001). *Manifesto for agile software development*. Agile Alliance. Retrieved from http://agilemanifesto.org/.

Bell, T. E. & Thayer, T. A. (1976). Software requirements: Are they really the problem? In *Proceedings of the 2nd International Conference on Software Engineering*.

Benington, H. D. (1983). Production of large computer programs. *Annals of the History of Computing*, 5(4), 350-361.

Berard, E. V. (1993). *Essays on object-oriented software engineering* (Vol. 1). Prentice-Hall.

Bhat, M., & Herschmann, J. (2020). *Agile and DevOps primer for 2020*. Gartner.com. Retrieved from https://www.gartner.com/en/documents/3979994/agile-and-devops-primer-for-2020.

Boehm, R. & Turner, R. (2004). Balancing agility and discipline: A guide for the perplexed. Addison-Wesley.

Bolton, R. N., McColl-Kennedy, J. R., Cheung, L., Gallan, A., Orsingher, C., Witell, L., & Zaki, M. (2018). Customer experience challenges: Bringing together digital, physical and social realms. *Journal of Service Management, 29*(5), 776-808.

Booch, G., (1993). *Object-oriented analysis and design with applications* (2nd Edition). Benjamin Cummings.

Burrell, G. & Morgan, G. (1979). *Sociological paradigms and organizational analysis*. Heineman.

Burton-Jones, A., Recker, J., Indulska, M., Green, P., & Weber, R. (2017). Assessing representation theory with a framework for pursuing success and failure. *MIS Quarterly*, *41*(4), 1307-1333.

Campbell-Kelly, M. & Aspray, W. (1996). Computer: A history of the information machine. Basic Books.

Ceruzzi, P. E. (2003). A history of modern computing, second edition. The MIT Press.

Chen, D., & Chen, L. (2020). Sliding-window probabilistic threshold aggregate queries on uncertain data streams. *Information Sciences, 520*, 353-372.

Chen, P. P. S. (1976). The entity-relationship model—Toward a unified view of data. *ACM Transactions on Database Systems, 1*(1), 9-36.

Coad, P., Yourdon, E., & Coad, P. (1991). *Object-oriented analysis* (Vol. 2). Yourdon Press.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM 13*(6), 377-387.

Codd, E. F. (1971). Further normalization of the data base relational model. In *Courant Computer Science Symposia Series 6- Data Base Systems*.

Codd, E. F. (1974). Recent investigations in relational data base systems. *Information Processing, 74*, 1017-1021.

Colbert, E. (1989). The object-oriented software development method: A practical approach to object-oriented development. In *Proceedings of the Conference on Tri-Ada '89: Ada Technology in Context: Application, Development, and Deployment* (pp. 400-415).

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., & Jeremaes, P. (1994). *Object-oriented development: The fusion method*. Prentice-Hall.

Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research, 20*(3), 329-354.

De Champeaux, D., Lea, D., & Faure, P. (1993). Object-oriented system development. Addison-Wesley Longman Publishing.

DeGrandis, D. (2011). DevOps: So you say you want a revolution? *Cutter IT Journal, 24*(8), 34-39.

DeMarco, T. (1979). Structure analysis and system specification. Yourdon Press.

Dennis, A. R., Samuel, B. M., & McNamara, K. (2014). Design for maintenance: How KMS document linking decisions affect maintenance effort and use. *Journal of Information Technology, 29*(4), 312-326.

El-Saddik, A. (2018). Digital twins: The convergence of multimedia technologies. *IEEE Multimedia, 25*(2), 87-92.

Embley, D. W., Kurtz, B. D., & Woodfield, S. N. (1991). Object-oriented systems analysis: A model-driven approach. Yourdon Press.

Erich, F., Amrit, C., & Daneva, M. (2017). A qualitative study of DevOps usage in practice. *Journal of Software: Evolution and Process, 29*(6), e1885.

Erickson, J. & Siau, K. (2008). Web services, service oriented computing, and service oriented architecture: Separating hype from reality. *Journal of Database Management, 19*(3), 42-54.

Erickson, J., Lyytinen, K., & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: The state of research. *Journal of Database Management, 16*(4), 88-100.

European Space Agency. (2006). HOOD. Retrieved from https://www.esa.int/TEC/Software_engineering_and_standardisation/TECKLAUXBQE_0.html.

Fagin, R. (1977). Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems, 2*(3), 262-278.

Fagin, R. (1981). A normal form for relational databases that is based on domains and keys. *ACM Transactions on Database Systems, 6*(3), 387-415.

Gane, C. & Sarson, T. (1977). *Structured systems analysis: Tools and techniques. Improved Systems Technologies*. McDonnell Douglas Information.

Hardgrave, B. C., Davis, F. D., & Riemenschneider, C. K. (2003). Investigating determinants of software developers' intentions to follow methodologies. *Journal of Management Information Systems, 20*(1), 123-151.

Hemon, A., Lyonnet, B., Rowe, F., & Fitzgerald, B. (2019). From Agile to DevOps: Smart skills and collaborations. *Information Systems Frontiers, 22*, 927-945.

Hirschheim, R. & Klein, K. (1989). Four paradigms of information systems development. *Communications of the ACM, 32*(10), 1199-1216.

Hirschheim, R., Klein, K., & Lyytinen, K. (1995). *Information systems development and data modeling: Conceptual and philosophical foundations*. Cambridge University Press.

Hughes, T. P. (1998). *Rescuing prometheus*. Vintage.

Iivari, J., Hirschheim, R., & Klein, H. (2001). A dynamic framework for classifying information systems development methodologies and approaches. *Journal of Management Information Systems, 17*(3), 179-218.

Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is DevOps? A systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016* (pp. 1-11).

Jacobson, I. (1993). *Object-oriented software engineering: A use case driven approach*. Pearson Education India.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process*. Addison Wesley Professional.

Jurik, J. A. & Schemenaur, R. S. (1992). Experiences in object-oriented development. In *Proceedings of the Conference on TRI-Ada '92* (pp. 189-197).

Larman, C. & Basili, V. R. (2003). Iterative and incremental development: A brief history. *IEEE Computer, 36*(3), 47-56.

Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2019). A survey of DevOps concepts and challenges. *ACM Computing Surveys, 52*(6), Article 127.

Lientz, B. P., & Swanson, E. B. (1980). *Software maintenance management*. Addison-Wesley.

Lukyanenko, R., Pastor, O., & Storey, V.C. (2021). Foundations of information technology based on Bunge's systemist philosophy of reality. *Software and Systems Modeling*, *20*, 921-938.

Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., Kuvaja, P., Mikkonen, T., Oivo, M., & Lassenius, C. (2019). DevOps in practice: A multiple case study of five companies. *Information and Software Technology, 114*, 217-230.

Mao, R., Zhang, H., Huang, H., Rong, G., Shen, H., Chen, L., & Lu, K. (2020). Preliminary findings about DevSecOps from grey literature. In *Proceedings of the 20th IEEE International Conference on Software Quality, Reliability, and Security (QRS),* (pp. 450-457).

Martin, J., & Odell, J. J. (1994). *Object-oriented methods*. Prentice-Hall PTR.

Meyer, B. (1997). *Object-oriented software construction* (vol. 2, pp. 331-410). Prentice-Hall.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics, 38*(8), 114-117.

Myrbakken H., Colomo-Palacios R. (2017). DevSecOps: A multivocal literature review. In A. Mas, A. Mesquida, R. O'Connor, T. Rout, A. Dorling (Eds.), *Software process improvement and capability determination. SPICE 2017. Communications in computer and information science* (vol. 770, pp. 17-29). Springer.

Nerson, J. M. (1992). Applying object-oriented analysis and design. *Communications of the ACM, 35*(9), 63-74.

Object Management Group. (2020, September 13). Business process model and notation. Retrieved from http://www.bpmn.org.

Odtadmin, (2019). *The Standish Group report 83.9% of IT projects partially or completely fail*. Opendoortechnology. Retrieved from https://www.opendoorerp.com/the-standish-group-report-83-9-of-it-projects-partially-or-completely-fail/.

Page-Jones, M. (1988). *The practical guide to structured systems design*. Yourdon Press.

Poppendieck, M. & Poppendieck, T. (2003). Lean software development: An agile toolkit. Addison-Wesley Professional.

Raman, V., Raman, B., & Hellerstein, J. M. (1999). Online dynamic reordering for interactive data processing. In *Proceedings of the 25th Vldb Conference*.

Reenskaug, T. (1996). *Working with objects OOram framework design principles*. Retrieved from https://www.semanticscholar.org/paper/Working-With-Objects-OOram-Framework-Design-Reenskaug/d826bac400be957ca5296f672e678cfe9aae8ad0

Roche, J. (2013). Adopting DevOps practices in quality assurance. *Communications of the ACM, 56*(11), 38-43.

Rowley, J.E. (1993). Information systems methodologies: A review and assessment of their applicability to the selection, design and implementation of library and information systems. *Journal of Information Science, 19*(4), 291-301.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. E. (1994). *Object-oriented modeling and design*. Prentice-Hall.

Sánchez-Gordón, M. & Colomo-Palacios, R. (2020). Security as culture: A systematic literature review of DevSecOps. In *Proceedings of the 1st International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS 2020) (*pp. 266-269).

Sardjono, W. & Retnowardhani, A. (2019). Analysis of failure factors in information systems project for software implementation at the organization. In *2019 International Conference on Information Management and Technology (ICIMTech)* (vol. 1, pp. 141-145).

Schmidt, R., Lyytinen, K., Keil, M., & Cule, P. (2001). Identifying software project risks: An international Delphi study. *Journal of management information systems, 17*(4), 5-36.

Schwaber, K. & Sutherland, J. (2017). The Scrum guide: The definitive guide to Scrum: The rules of the game. Retrieved from https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf.

Selic, B., Gullekson, G., & Ward, P. T. (1994). *Real-time object-oriented modeling*. John Wiley & Sons.

Shen, Z., Tan, S., & Siau, K. (2018). Challenges in learning UML: From the perspective of diagrammatic representation and reasoning. *Communications of the Association for Information Systems*, *43*, Article 30, 545-565.

Shlaer, S. & Mellor, S. J. (1992). *Object life-cycles: Modeling the world in states*. Yourdon Press.

Siau, K., Long, Y. & Ling, M. (2010). Toward a unified model of information systems success. *Journal of Database Management, 21*(1), 80-101.

Siau, K. & Rossi, M. (2011). Evaluation techniques for systems analysis and design modelling methods: A review and comparative analysis. *Information Systems Journal, 21*(3), 249-268.

Siau, K. & Tan, X. (2005a). Improving the quality of conceptual modeling using cognitive mapping techniques. *Data And Knowledge Engineering, 55*(3), 343-365.

Siau, K. & Tan, X. (2005b). Technical communication in information systems development: The use of cognitive mapping. *IEEE Transactions on Professional Communications, 48*(3), 2005b, 269-284.

Siau, K. & Tan, X. (2005c). Evaluation criteria for information systems development methodologies. *Communications of the Association for Information Systems, 16*, 856-872.

Siau, K., Wand, Y., & Benbasat, I. (1997). The relative importance of structural constraints and surface semantics in information modeling. *Information Systems, 22*(2-3), 155-170.

Smith, H. J., Keil, M., & Depledge, G. (2001). Keeping mum as the project goes under: Toward an explanatory model. *Journal of Management Information Systems, 18*(2), 189-227.

Spafford G. & Herschmann, J. (2019). *Hype cycle for DevOps, 2019*. Gartner.com. Retrieved from https://www.gartner.com/en/documents/3947533.

Stroustrup, B. (1988). What is object-oriented programming? *IEEE Software, 5*(3), 10-20.

Svensson, R. B., Claps, G. G., & Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology, 57*, 21-31.

Swanson, E. B. (1994). Information systems innovation among organizations. *Management Science, 40* (9), 1069-1092.

Tomas, N., Li J., & Huang, H. (2019). An empirical study on culture, automation, measurement, and sharing of DevSecOps. In *International Conference on Cyber Security and Protection of Digital Services (Cyber Security)* (pp. 1-8.).

Wakefield, J. (2018). *The man who was fired by a machine*. BBC News. Retrieved from https://www.bbc.com/news/technology-44561838. Retrieved August 21, 2020.

Wand, Y. & Weber, R. (1993). On the ontological expressiveness of information systems analysis and design grammars. *Journal of Information Systems, 3*(4), 217-237.

Wand, Y. & Weber, R. (2017). Thirty years later: Some reflections on ontological analysis in conceptual modeling. *Journal of Database Management, 28*(1), 1-17.

Wasserman, A. I., Pircher, P. A., & Muller, R. J. (1990). The object-oriented structured design notation for software design representation. *Computer, 23*(3), 50-63.

Wei, C., Chiang, R., & Wu, C. (2006). Accommodating individual preferences in the categorization of documents: A personalized clustering approach. *Journal of Management Information Systems, 23*(2), 2006, 173-201.

Weinberg, G. M. (1982). *Rethinking systems analysis and design*. Little, Brown Boston.

Wiedemann, A., Forsgren, N., Wiesche, M., Gewald, H., & Krcmar, H. (2019). Research for practice: The DevOps phenomenon. *Communications of the ACM, 62*(8), 44-49.

Wirfs-Brock, R., Wilkerson, B., & Wiener, L. (1990). *Designing object-oriented software*. Pearson.

Yourdon, E. (1979). Structured design: Fundamentals of a discipline of computer program and systems design. Yourdon Press.

Yourdon, E. (1988). Modern structured analysis. Yourdon Press.

Zhao, L. & Siau, K. (2002). Component-based development using UML. *Communications of the Association for Information Systems, 9*, 207-222.

About the Authors

Keng Siau is the Department Head and Chair Professor of Information Systems at the City University of Hong Kong. Prior to joining the City University of Hong Kong, he was the Head of Business Programs at the Missouri University of Science and Technology. Before joining the Missouri University of Science and Technology in June 2012, Professor Siau was Edwin J. Faulkner Chair Professor and Full Professor of Management at the University of Nebraska-Lincoln (UNL). He was also the Director of the UNL-IBM Global Innovation Hub. Professor Siau is Editor-in-Chief of the Journal of Database Management (SCI, ABDC's A journal). He has served as the North America Regional Editor of the Requirements Engineering journal (2010-2016). He has also served as the Vice President of Education for the Association for Information Systems (AIS) and the AIS representative on the Board of Partnership for Advancing Computing Education from July 2011 to June 2014. Professor Siau has more than 250 academic publications. According to Google Scholar, he has a citation count of more than 16,000. His h-index and i10-index, according to Google Scholar, are 65 and 158, respectively. Professor Siau is consistently ranked as one of the top information systems researchers in the world based on his h-index and productivity rate. In 2006, he was ranked as one of the top ten e-commerce researchers in the world (Arithmetic Rank of 7, Geometric Rank of 3). In 2006, the citation count for his paper "Building Customer Trust in Mobile Commerce" was ranked in the top 1% in the field as reported by Essential Science Indicators. He is also on the 2020 and 2021 Stanford University's lists of top 2% most-cited scientists in the world and ranked as one of the top computer scientists in the U.S. and the world.

Carson C. Woo is Stanley Kwok Professor of Business at the Sauder School of Business, University of British Columbia. His research interests include systems analysis and design, requirements engineering, conceptual modelling, and design science research. In particular, he is interested in using conceptual models to acquire relevant contextual information (e.g., business goals) and utilizing it to design new information systems, or aligning it to existing information systems design, so that future changes can be accommodated to business needs without social-technical challenges. Dr. Woo is editor-in-chief of the Data and Knowledge Engineering journal by Elsevier, editor of Information Technology and Systems Abstracts Journal at the Social Science Research Network (SSRN), and member of the steering committee of the International Conference of Conceptual Modeling, where he served as chair of the committee during 2019-2020.

Veda C. Storey is the Tull Professor of Computer Information Systems and professor of computer science at the J. Mack Robinson College of Business, Georgia State University. Her research interests are in data management, conceptual modelling, and design science research. She is particularly interested in the assessment of the impact of new technologies on business and society from a data management perspective. Dr. Storey is a member of AIS College of Senior Scholars and the steering committee of the International Conference of Conceptual Modeling.

Roger Chiang is a Professor of Information Systems at Carl H. Lindner College of Business, University of Cincinnati. He received his Ph.D. in Computers and Information Systems from the University of Rochester. His research interests are in business intelligence and analytics, data and knowledge management, and intelligent systems. He has published over sixty refereed articles in journals and conferences, including ACM Transactions on Database Systems, ACM Transactions on Management Information Systems, Communications of the ACM, The DATA BASE for Advances in Information Systems, Data & Knowledge Engineering, Decision Support Systems, Journal of American Society for Information Science and Technology, Journal of Database Administration, Journal of Management Information Systems, Marketing Science, MIS Quarterly, and the Very Large Data Base Journal. He has also served as a Senior Editor and Associate Editor at some of these leading journals.

Cecil Eng Huang Chua is an associate professor at the Missouri University of Science & Technology. He received a Ph.D. in Information Systems from Georgia State University, a Masters of Business by Research from Nanyang Technological University and both a Bachelor of Business Administration in Computer Information Systems and Economics and a Masters Certificate in Telecommunications Management from the University of Miami. Cecil has several publications in such journals as *Information Systems Research, Journal of the AIS, MIS Quarterly* and the *VLDB Journal.* He is a senior editor for both *AIS Transactions on Human-Computer Interaction* and the *Pacific Asia Journal of the Association for Information Systems,* desk editor for the *Project Management Journal* and an associate editor for both *Information & Management* and the *Information Systems Journal.* Cecil has consulted for a range of

organizations including Daimler SEAsia, General Motors Singapore, the Singapore Ministry of Defense, and Fonterra- the New Zealand milk cooperative that produces 1/3rd of the world's globally traded milk.

Jon W. Beard is an Associate Teaching Professor in the Information Systems and Business Analytics Department in the Ivy College of Business at Iowa State University. He has worked in industry as a systems engineer and a management and IT consultant. His research and teaching are on Systems Thinking, Design Thinking, Systems Analysis and Design, Business Process Management, Project Management, and Strategic Information Systems. Jon has published research in the *IEEE Transactions on Engineering Management*, the *Journal of Strategic Information Systems*, *Communications of the Association for Information Systems*, the *Journal of Business Ethics*, and the *Journal of Information Systems Education*. He is the editor of a book on Information Technology Applications for Crisis Response and Management and two books on impression management and information technology.