frontiers in Computer Science

SPECIALTY GRAND CHALLENGE
published: 17 March 2022
doi: 10.3389/fcomp.2021.766053

On the Frontiers of Software Science and Software Engineering

Yingxu Wang *†

Department of Electrical and Software Engineering, International Institute of Cognitive Informatics and Cognitive Computing (I2CICC), Schulich School of Engineering, Hotchkiss Brain Institute, University of Calgary, Calgary, AB, Canada

Advances in software engineering, software science, computational intelligence, and intelligent mathematics have led to the establishment of *Frontiers in Computer Science—Software* (FCSS). FCSS aims to promote transdisciplinary research on software science and engineering (SSE), autonomous systems, and computational intelligence. FCSS covers not only classical empirical software engineering and industrial processes, but also contemporary topics of software science, intelligent programming languages, autonomous code generation, mathematical foundations of software, and programming knowledge bases. FCSS reports empirical studies and emerging topics in software engineering including tools, development platforms, industrial processes, management infrastructures, quality assurance schemes, big data systems, and software migrations across languages and platforms.

Keywords: software science, software engineering, theoretical foundations, intelligent mathematics, empirical studies, architectures, best practices, autonomous software generation

OPEN ACCESS

Edited and reviewed by:
Kaleem Siddiqi,
McGill University, Canada

***Correspondence:**
Yingxu Wang
yingxu@ucalgary.ca

†FIEEE, FBCS, FI2CICC, FAAIA, and FWIF, Specialty Chief Editor of Frontiers of Software Science and Software Engineering

Specialty section:
This article was submitted to Software, a section of the journal Frontiers in Computer Science

Received: *28 August 2021*
Accepted: *09 November 2021*
Published: *17 March 2022*

Citation:
Wang Y (2022) On the Frontiers of Software Science and Software Engineering. Front. Comput. Sci. 3:766053. doi: 10.3389/fcomp.2021.766053

INTRODUCTION

The latest advances in computer science, software theories, intelligence science, knowledge science, intelligent mathematics, and autonomous systems have triggered the emergence of software science as a transdisciplinary field overarching software engineering, programming theories, formal methods, and computational intelligence (Turing, 1950; von Neumann, 1946; Dijkstra, 1976; Hoare, 1978; Hoare et al., 1987; Wang, 2014; Wang, 2007a; Wang et al., 2008; Wang and Gafurov, 2010; Wang, 2016).

Software science studies the formal properties and mathematical models of software, general methodologies for rigorous and efficient software development, and coherent theories underpinning software behaviors and software engineering practices (Wang, 2014). The discipline of software science encompasses theories and methodologies, intelligent mathematics, system software, fundamental algorithms, organizational theories, cognitive complexity of software, and intelligent behavior generation theories. Recent developments in software science have paved the way to enable novel technologies for *AI programming* (AIP) (Wang and Xu, 2019).

Software engineering is a discipline underpinned by software science that adopts engineering approaches to develop large-scale software towards high productivity, low cost, trustworthy quality, and controllable development schedule (Wang, 2007a). Software engineering is one of the most complicated branches of engineering fields because its objects are highly abstract and intangible. It encompasses system modeling, architecting, development methodologies, programming technologies, and supporting platforms. It also covers heuristic principles, tools/environments, best practices, case studies, experiments, trials, and performance benchmarking.

This editorial addresses the objectives, missions, challenges, and latest development of *Frontiers in Computer Science—Software* (FCSS). In the framework of FCSS, software science foci on foundations and theoretical software engineering, whilst software engineering covers heuristic principles, tools, development environments, best practices, and programming technologies. A set of key challenges and opportunities of software science and engineering has been recognized in *Challenges and Opportunities in Software Science and Engineering*. The emerging fields of software science and novel technologies of software engineering are elaborated in *The Frontier of Software Science and The Frontier of Software Engineering*, respectively.

CHALLENGES AND OPPORTUNITIES IN SOFTWARE SCIENCE AND ENGINEERING

Despite the fast growth of software engineering over the past 60 years, software development has still required intensive manual interactions in the software industry (Dijkstra, 1976; Hoare, 1978; Wang, 2014). This phenomenon indicates a fundamental challenge to both theories and technologies of *Software Science and Engineering* (SSE), which are substantially constrained by the following reasons and traditional practices.

1) *The fundamental theories towards software science are immature*: It is indicated by a common perception shared by many researchers and practitioners that software engineering does not obey any known physical laws. However, there is a lack of basic research seeking what really are the laws and their theoretical foundation underpinning SSE.
2) *The traditional perceptions on the essences of software are questionable*: Software or program is used be treated as a list of code. However, latest discoveries reveal software is a cohesive system of intelligent behaviors generated by dynamic interactions between the sets of *operational events* (E), *process models* (PM), and *structural models* (SM) across the three dimensions (Wang, 2004; Wang, 2008a).
3) *Essential Intelligent Mathematics (IM) for modeling and manipulating human and software behaviors are missing*: IMs (Wang, 2020) for SSE are contemporary mathematical means beyond propositional logics and discrete mathematics, which encompass *system algebra, concept algebra*, and *Real-Time Process Algebra* (RTPA) (Wang, 2008a), etc. It is discovered that suitable IMs for software system specifications and algorithm descriptions had been left behind to the demands in SSE. As a result, none of the essential processes from user requirement elicitation, system specification, to code generation may be rigorously manipulated towards the maturity of software science.
4) *The current programming languages are anti-productive because they are not designed for machine enabled software generation*: Our typical programming languages are still machine-oriented constrained by deterministic condition-driven compilers. The limited expressive power of them, particularly in the most important real-time environments, has substantially constrained their adaptation to intelligent software specifications and AIP towards autonomous intelligence generation (Wang and Xu, 2019). Instead, non-programmable neural networks have become the optional solutions for SSE in most challenging real-time contexts (Wang, 2004; Wang, 2008b).

A set of *Grand Challenges* (GCs) to SSE have been identified in basic research (Turing, 1950; von Neumann, 1946; Dijkstra, 1976; Hoare, 1978; Hoare et al., 1987; Wang, 2014; Wang, 2007a; Wang et al., 2008; Wang et al., 2021; Wang, 2002; Wang, 2007b; Wang et al., 2009a; Wang et al., 2004; Wang, 2020; Wang, 2008c; Wang, 2012c; Wang, 2004; Wang and Xu, 2019; Wang et al., 2009b) of this field, which may be highlighted by the following fundamental queries:

1) What are the necessary and sufficient conditions for enabling run-time software intelligence in SSE?
2) Why have AI and autonomous systems been developed by data-driven neural networks rather than program-driven software engineering?
3) Does that of GC2 indicate a theoretical or technical challenge to SSE?
4) How mature are our computing platforms and programming languages for enabling autonomous software generation, mostly at run-time?
5) Is *Stored-Program-Controlled* (SPC) computers (Turing, 1950; von Neumann, 1946), i.e., von Neumann machines (VNM) (von Neumann, 1946), adequate enough for intelligent software design? If not, what kind of computers will be needed for the next generation of intelligent software engineering?
6) Are our programming languages sufficiently expressive for designing intelligent software? What would happen to a software system if the deterministic if-then-else structures were indeterminable at designed-time or exhausted at run-time?
7) Are our mathematical means ready for rigorously expressing software system requirements? Is our inference power adequate for expressing real-time inexhaustive, indeterministic, and uncertain behaviors in SSE?
8) How may an intelligent software system be trusted and verified when its state space is infinitive in *de facto*, such as those of self-driving vehicles and mission-critical robots?
9) How may a nondeterministic intelligent programming language be created in order to enable run-time intelligent behavior generation for handling indeterministic events and autonomous decision-making requirements?
10) How will SSE enable the next generation of intelligent system software and autonomous behavior generation systems?

These GCs provide a set of theoretical and technological challenges to and opportunities for SSE as well as the software industry. Any breakthrough on the set of GCs will lead to a wide range of technical innovations and applications in the software and computational intelligence industries.

THE FRONTIER OF SOFTWARE SCIENCE

It is recognized that a rigorous theoretical framework of software science is yet to be sought based on a plenty of repositories of empirical knowledge about the nature of software and its development. As Edsger Dijkstra stated that "Software engineering is programming when you can't (achieve) (Dijkstra, 1976)." Therefore, software science is what one cannot solve by empirical software engineering (Wang, 2014). The pinnacle of software science is the theories of formal methods (Turing, 1950; Hoare, 1978; Hoare et al., 1987; Wang et al., 2008) and platforms of system software including operating systems, compilers, database systems, and Internet-based distributed systems (von Neumann, 1946; Dijkstra, 1976; Wang, 2014). C.A.R. Hoare has revealed that software is a *mathematical entity* (Hoare, 1978) that may be formally denoted by a behavioral process known as the *unified process theory* in formal methods towards software engineering (Hoare et al., 1987). A set of 30+ laws of programming has been discovered by Hoare and his colleagues (Hoare et al., 1987). While being a visiting professor in Prof. Hoare's laboratory at the University of Oxford, the author of this article has developed a Real-Time Process Algebra (RTPA) (Wang, 2008a). RTPA extends Hoare's sequential process theory to a system of intelligent mathematics for formally denotating system and human cognitive, inference, and behavioral processes as a general theory towards software science (Wang, 2014). Based on RTPA, a comprehensive set of 95 mathematical laws and associated theorems on software structures, behaviors, and processes has been formally established (Wang et al., 2008).

Software science is a discipline that studies the formal properties and mathematical models of software, general methodologies for rigorous and efficient software development, and coherent theories and laws underpinning software behaviors as well as software engineering practices. The architecture of software science encompasses theories and methodologies, intelligent mathematics, intelligent system software, and software engineering processes (Wang, 2007a). It is noteworthy that the focuses of software science are on the fundamental platforms of *Intelligent System Software* (ISS) including those of intelligent operating systems, intelligent requirement specifiers, intelligent compilers, and intelligent programming knowledge bases. The development of ISS will be based on an indispensable foundation known as intelligent mathematics that enables software engineering to be matured towards software science based on rigorous denotational mathematics beyond inexpressive programming languages.

It is noteworthy that IM (Wang, 2012a; Wang, 2012d; Wang, 2020) is centric in software science as a category of denotational mathematical structures. IM deals with complex mathematical entities beyond pure numbers such as abstract objects, complex relations, behavioral interactions, denotational concepts, knowledge, processes, programmable intelligence, and general systems. IM is an indispensable foundation for dealing with the abstract and complex entities of software structures and behaviors. A set of IMs have been created in the last decade embodied by *system algebra* (Wang, 2008b), *concept algebra* (Wang, 2006; Wang, 2010), RTPA (Wang, 2008a), *semantic algebra* (Wang, 2010; Wang, 2013), and *inference algebra* (Wang, 2012b) towards AIP. IM provides a coherent set of contemporary mathematical means and explicit expressive power for manipulating both complex mathematical objects and long-chains of serial or deep-layers of embedded mathematical operations in applied domains. In software science, a General Mathematical Model of Software (GMMS) (Wang, 2014) is discovered based on RTPA that reveals any software system is a derived instance of GMMS for rigorously manipulating arbitrary software systems as behavioral interactions in the *universe of discourse of software* (Ω) determined by the Cartesian product Ω = E x PM x SM (Wang, 2014). Therefore, the maturity of IM may indicate the maturity of software science for rigorously processing system architectures and behaviors with abstract concepts, complex relations, and dynamic processes (Wang, 2020).

THE FRONTIER OF SOFTWARE ENGINEERING

The latest advances of theories and methodologies in software science, computational intelligence, and IMs have paved a way to enable machines to generate programs in software engineering (Wang, 2007a). The challenges to intelligent program generation have been constrained by the lack of autonomous programming theories and the difficulties stemmed from the complexity of software. The missing of machine-understandable semantical rules of software behaviors has prevented software engineering from advancing to intelligent program generation. Pilot studies indicate that a fusion of the analytic (mathematical rule-based) and gray-box (machine-learning-based) methodologies will be necessarily adopted towards autonomous program generation (Wang, 2004; Wang and Xu, 2019).

An ultimate strategy for the next generation of software engineering is AIP (Wang, 2004; Wang et al., 2009b; Wang and Xu, 2019), which has been enabled by a comprehensive set of software laws for software architectures, structures, and behaviors. In the AIP approach to software engineering, IMs service as rigorous mathematical means for conveying human programming knowledge and skills to machines (Wang, 2014). IMs for AIP is underpinned by: 1) System algebra for software architectural design; 2) Concept algebra and semantic algebra for rigorous requirement modeling; and 3) RTPA for formally manipulating software structures and behaviors. Not only the architectures of computational intelligent systems, but also their dynamic behaviors can be rigorously and systematically manipulated by IMs. Several large-scale projects designed based IMs have demonstrated that they are a set of powerful mathematical means for dealing with concepts, knowledge, behavioral processes, and human/machine intelligence in real-world software engineering.

The relationship between software science and software engineering is explained by analogizing those of pure and applied physics. Without theoretical physics there would be no maturity of applied physics; so is software science for software engineering. It is recognized that the phenomena where almost all the fundamental problems that could not be solved in the past 60 years in software engineering has been stemmed from the lack of rigorous theories in

the discipline of software science toward autonomous software generation.

CONCLUSION

The emerging field of Software Science and Engineering (SSE) investigates into the theoretical foundations and intelligent mathematics for autonomous software generation. This editorial has explored the latest basic research in software science. Applications of software science and intelligent mathematics in software engineering and autonomous software generation have been presented. A set of key challenges and opportunities for SSE has been formally reviewed and analyzed towards intelligent software engineering and AIP. The advanced of SSE will enable the horse being put in the front of the cart in order to enhance traditional human-centered programming by intelligent machines in the central processes of software engineering.

Fronters in Computer Science (FCS)—Software (FCSS) is a new section in Journal of FCS. FCSS aims to promote transdisciplinary research and engineering applications in software science, software engineering, software architecture, and industrial processes. The coverage of FCSS encompasses not only classical empirical software engineering, but also contemporary topics of software science, AI programming, and their mathematical and cognitive foundations. Original papers, revised conference articles, and industrial technical reports are welcome in all fields and topics of software, programming, algorithms, software engineering, and software science.

AUTHOR CONTRIBUTIONS

The author confirms being the sole contributor of this work and has approved it for publication.

ACKNOWLEDGMENTS

Many people have contributed to the establishment of FCSS. The Specialty Chief Editor (SCE) of FCSS would like to thank the EIC of FCS, KS, as well as associate editors and the board of reviewers. The SCE of FCSS would like to acknowledge the professional work of Dr. Thomas Croft, Dr. Enrique Morillas, Bennett Colgan, and Rossana Isola at the editorial office of Frontiers.

REFERENCES

Dijkstra, E. W. (1976). *A Discipline of Programming*. Englewood Cliffs, NJ: Prentice-Hall.

Hoare, C. A. R. (1978). Communicating Sequential Processes. *Commun. ACM* 21 (8), 666–677. doi:10.1145/359576.359585

Hoare, C. A. R., Hayes, I. J., Jifeng, H., Morgan, C. C., Roscoe, A. W., Sanders, J. W., et al. (1987). etcLaws of Programming. *Commun. ACM* 30 (8), 672–686. doi:10.1145/27651.27653

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind* LIX (59), 433–460. doi:10.1093/mind/lix.236.433

von Neumann, J. (1946). The Principles of Large-Scale Computing Machines. *Ann. Hist. Comput.* 3 (3), 263–273.

Wang, Y. (2002). "Cognitive Models of the Brain," in Proceedings of the First IEEE International Conference on Cognitive Informatics (ICCI'02), Calgary, AB, Canada, August 2002 (IEEE CS Press), 259–269.

Wang, Y. (2009). "Formal Description of the Cognitive Process of Memorization," in *Transactions on Computational Science*. Editors M. L. Gavrilova and C. J. K. Tan (Berlin, Germany: Springer), 5, 81–98. doi:10.1007/978-3-642-02097-1_5

Wang, Y., and Gafurov, D. (2010). The Cognitive Process of Comprehension. *Int'l J. Cogn. Inform. Nat. Intelligence* 4 (3), 44–58. doi:10.4018/jcini.2010070104

Wang, Y. (2012). In Search of Denotational Mathematics: Novel Mathematical Means for Contemporary Intelligence, Brain, and Knowledge Sciences. *J. Adv. Math. Appl.* 1 (1), 4–26. doi:10.1166/jama.2012.1002

Wang, Y. (2012). Inference Algebra (IA). *Int'l J. Cogn. Inform. Nat. Intelligence* 6 (1), 21–47. doi:10.4018/jcini.2012010102

Wang, Y., Karray, F., Kaynak, O., Kwong, S., Leung, H., Hou, M., et al. (2021). Perspectives on the Philosophical, Cognitive and Mathematical Foundations of Symbiotic Autonomous Systems. *Philosophical Trans. R. Soc. (A) Oxford, UK* 379 (2207), 1–20. doi:10.1098/rsta.2020.0362

Wang, Y. (2020). "Keynote: Intelligent Mathematics: A Basic Research on Foundations of Autonomous Systems, General AI, Machine Learning, and Intelligence Science," in Proceedings of the IEEE 19th Int'l Conf. on Cognitive Informatics and Cognitive Computing (ICCI*CC'20), Beijing, China, September 2020 (NY: IEEE Press), 5.

Wang, Y., Latombe, J.-C., Zhang, D., and Kinsner, W. (2009). Advances in Cognitive Informatics and Cognitive Computing: Report on IEEE ICCI'08 at Stanford University. *Int'l J. Cogn. Inform. Nat. Intelligence* 3 (4), 91–95. doi:10.4018/jcini.2009062306

Wang, Y., Liu, D., and Ruhe, G. (2004). "Formal Description of the Cognitive Process of Decision Making," in Proceedings of the 3rd IEEE International Conference on Cognitive Informatics, Victoria, BC, Canada, August 2004 (IEEE CS, Press), 124–130. doi:10.1109/coginf.2004.1327467

Wang, Y. (2016). On Cognitive Foundations and Mathematical Theories of Knowledge Science. *Int'l J. Cogn. Inform. Nat. Intelligence* 10 (2), 1–25. doi:10.4018/ijcini.2016040101

Wang, Y. (2004). "On Cognitive Informatics Foundations of Software Engineering," in Proceedings of the 3rd IEEE Int'l Conference on Cognitive Informatics (ICCI'04), Canada, August 2004 (IEEE CS Press), 22–31.

Wang, Y. (2006). "On Concept Algebra and Knowledge Representation," in Proceedings of the IEEE 5th Int'l Conference on Cognitive Informatics (ICCI'06), Beijing, China, July 2006, 320–331. doi:10.1109/coginf.2006.365514

Wang, Y. (2010). On Concept Algebra for Computing with Words (CWW). *Int'l J. Semantic Comput.* 4 (3), 331–356. doi:10.1142/s1793351x10001061

Wang, Y. (2012). On Denotational Mathematics Foundations for the Next Generation of Computers: Cognitive Computers for Knowledge Processing. *J. Adv. Math. Appl.* 1 (1), 118–129. doi:10.1166/jama.2012.1009

Wang, Y. (2008). "On Mathematical Laws of Software," in *Transactions of Computational Science*. Editors M. L. Gavrilova, C. J. K. Tan, Y. Wang, Y. Yao, and G. Wang (Berlin, Germany: Springer), 2, 46–83. doi:10.1007/978-3-540-87563-5_4

Wang, Y. (2013). On Semantic Algebra: A Denotational Mathematics for Natural Language Comprehension and Cognitive Computing. *J. Adv. Math. Appl.* 2 (2), 145–161. doi:10.1166/jama.2013.1039

Wang, Y. (2008). On System Algebra. *Int'l J. Cogn. Inform. Nat. Intelligence* 2 (2), 20–43. doi:10.4018/jcini.2008040102

Wang, Y. (2008). On the Big-R Notation for Describing Interative and Recursive Behaviors. *Int. J. Cogn. Inform. Nat. Intelligence* 2 (1), 17–28. doi:10.4018/jcini.2008010102

Wang, Y. (2012). On Visual Semantic Algebra (VSA): A Denotational Mathematical Structure for Modeling and Manipulating Visual Objects and Patterns. *Softw. Intell. Sci. New Transdisciplinary Findings*. 68–81. doi:10.4018/978-1-4666-0261-8

Wang, Y. (2008). Rtpa. *Int'l J. Cogn. Inform. Nat. Intelligence* 2 (2), 44–62. doi:10.4018/jcini.2008040103

Wang, Y. (2007). *Software Engineering Foundations: A Software Science Perspective*, 1. NY, USA: CRC/Auerbach Publications, 580 pp.

Wang, Y. (2014). Software Science: On the General Mathematical Models and Formal Properties of Software. *J. Adv. Math. Appl.* 3 (2), 130–147. doi:10.1166/jama.2014.1060

Wang, Y. (2007). The Cognitive Processes of Formal Inferences. *Int'l J. Cogn. Inform. Nat. Intelligence* 1 (4), 75–86. doi:10.4018/jcini.2007100106

Wang, Y., and Xu, J. (2019). "RTPA-based Software Generation by AI Programming," in Proceedings of the 18th IEEE Int'l Conference on Cognitive Informatics and Cognitive Computing (ICCI*CC'19), Polytechnic Milan, Italy, July 2019 (IEEE CS Press), 41–46. doi:10.1109/iccicc46617.2019.9146036

Conflict of Interest: The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.