SCIENCE CHINA
Information Sciences

Path planning for mobile robot using self-adaptive learning particle swarm optimization

Guangsheng LI[1,*] and Wusheng CHOU[1,2]

Citation: SCIENCE CHINA Information Sciences **61**, 052204 (2018); doi: 10.1007/s11432-016-9115-2

View online: https://engine.scichina.com/doi/10.1007/s11432-016-9115-2

View Table of Contents: https://engine.scichina.com/publisher/scp/journal/SCIS/61/5

Published by the Science China Press

SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

May 2018, Vol. 61 050101:1–050101:14
https://doi.org/10.1007/s11432-017-9372-7

Special Focus on Internetware: Research Progress and Current Trend

Situation analytics — at the dawn of a new software engineering paradigm†

Carl K. CHANG

Department of Computer Science, Iowa State University, Ames Iowa 50011*, USA*

Received 21 December 2017/Revised 16 February 2018/Accepted 26 February 2018/Published online 20 April 2018

Abstract In this paper, I first review the seminal work by Thomas Kuhn — *The Structure of Scientific Revolutions* — and elaborate my view on paradigm shifts in software engineering research and practice as it turns 50 years old in 2018. I then examine major undertakings of the computing profession since early days of modern computing, especially those done by the software engineering community as a whole. I also enumerate anomalies and crises that occurred at various stages, and the attempts to provide solutions by the software engineering professionals in the past five decades. After providing such a background, I direct readers' attention toward emerging anomalies in software engineering, at a severity level that is causing another software engineering crisis, and suggest a set of criteria for feasible solutions. The main theme of this paper is to advocate that situation analytics, equipped with necessary definitions of essential concepts including situation and intention as parts of a new computational framework, can serve as the foundation for a new software engineering paradigm named the Situation-Centric Paradigm. In this framework, situation is considered a new abstraction for computing and is clearly differentiated from the widely accepted existing abstractions, namely function and object. I argue that the software engineering professionals will inevitably move into this new paradigm, willingly or unwillingly, to empower Human-Embedded Computing (HEC) and End-User Embedded Computing (EUEC), much more than what they have done with traditional human-centered or user-centric computing altogether. In the end, I speculate that an ultimate agile method may be on the rise, and challenge readers to contemplate "what if" hundreds of thousands "end-user developers" emerge into the scene where the boundaries between end users and developers become much more blurred.

Keywords ultimate agile method, context, intention, Internet of things, paradigm, situation, software engineering, human-embedded computing

Citation Chang C K. Situation analytics — at the dawn of a new software engineering paradigm. Sci China Inf Sci, 2018, 61(5): 050101, https://doi.org/10.1007/s11432-017-9372-7

1 Introduction

The purpose of this paper is to present the background and rationale for an optimistic proposition that the human-centric computational abstraction "situation" may be used to help usher in Human-Embedded Computing (HEC) or End-User Embedded Computing (EUEC), a drastically different way of developing and evolving software systems and providing computer based services. There are early signs to support my belief that radical change towards a new software engineering paradigm is on the horizon.

First and foremost, let me make note that 2018 marks the 50th anniversary of software engineering (SE) since the term was coined at the 1968 NATO conference [1], and it is generally regarded as the

Email: chang@iastate.edu
† Invited paper

birth of software engineering as a computing field. In those 50 years, we have come a long way. It is time to celebrate and appreciate the outstanding work done by many SE pioneers and fine researchers. Celebration aside, we must also evaluate the evolution path of software engineering to understand how the field has progressed in the past half century, and how it should move forward to achieve even more. I will first refresh readers' mind and analyze what elements of a discipline can be used to qualify a paradigm, and why I think a paradigm shift in software engineering is imminent.

As we have entered an "at-scale" Internet of things (IoT) era [2], it is prudent for SE researchers to rethink the typical approaches to conduct SE research and develop software systems. There is a flood of hot topics and buzzwords in use by SE researchers racing to secure the SE community's attention and obtain approval for their work. Such fashionable keywords include dynamic, cooperative, evolvable, autonomous, emergent, and situational [3]. In addition, recently used keywords in SE literature such as cognitive [4] and humanistic [5] also have begun to label certain concerns in SE research. We, the SE researchers, often classify our work dealing with human factors as being user-centric and, more recently, human-centered. Do current user-centric SE research projects truly cover all humanistic concerns, especially those of the end-users? Do we have a powerful method to quickly evolve software services as needs arise in the field after the system has been deployed, due to user's cognitive or physical feedback sensed in the deployed system in real-time? The answers to all such questions are "no".

The remainder of this paper is organized as follows. In Section 2, I briefly review the notion of "scientific paradigm" pertaining to Kuhn's original work [6] on this subject, and highlight anomalies as the prerequisite leading to a "crisis" — thus an imminent "paradigm shift". Through reviewing the history of modern computing in the past five decades, I also give my reasoning about a possible paradigm shift in software engineering based on my observations on emerging "software engineering anomalies". In Section 3, I first declare that we have entered the aware computing era. I then differentiate context-awareness from situation-awareness, and explain why the latter must be bundled with the human subject's mental state that is mostly hidden. In Section 4, I introduce my work on situation analytics, and substantiate my claim that situation analytics can be the foundation of a new SE paradigm. My theory to justify most noticeable SE paradigm shifts occurring in the past five decades is founded on the realization of widely adopted computational abstractions as pivots of our profession, namely the functional paradigm, the object-oriented paradigm and the situation-centric paradigm that has been showing its strength lately but neglected by most SE professionals. As I keep reminding readers about the human dominance in situation-aware computing, in Section 5 I further define two new notions for readers to consider: human embedded computing and end-user embedded computing, where the latter is a special case of the former. Finally, I conclude this paper in Section 6, and offer my speculations into the future of computing.

2 Software engineering paradigm

SE researchers often refer to a coherent bundle of emerging and sometimes unfamiliar concepts and practices in a rather loose fashion as a new "paradigm", to the extent that more rigorously minded scholars may frown at times. In this section I shall examine what a paradigm is, how we have recognized and embrace paradigms in SE, and how we acknowledge a fundamental shift and discover new paradigm requirements to meet future challenges.

2.1 A word on paradigm

I suggest that we resort to the classical and widely cited source [6] to establish a common base when we elaborate on the subject of paradigm. Thomas Kuhn, an American physicist and philosopher, used the word "paradigm" to mean "exemplar". It came from the Greek origin paradeigma (or "exemplum" from the Latin) that were used in, for example, Aristotle's study on epistemology where exemplar meant a very best or most instructive example. The Merriam-Webster dictionary defines paradigm as "a scientific school or discipline within which theories, laws and generalizations and the experiments performed in

support of them are formulated". In the history of scientific research, we consider it a clear "paradigm shift" from Newton's "classical mechanics" to the modern-day "relativistic mechanics" and "quantum mechanics" [6]. Such paradigm shifts are motivated by the human quest for understanding the surrounding world in a process termed puzzle solving by Kuhn. Kuhn explained in his book [6] that paradigm shift means following the path from observed anomalies at the scene to the appearance of a crisis, then eventually to the realization of a revolution once the underlying science (laws, principles, predictability, etc.) is believed to have been developed. We shall discuss the changing landscape of SE research in terms of its paradigm shift.

2.2 Software engineering paradigms

Keeping in mind Kuhn's defined conception of "paradigm", we will now focus on software engineering as discipline and attempt to discuss its true paradigm shifts in the past five decades. In this regard, Rajlich suggested a new SE paradigm based on software evolution [7]. From Kuhn's original observations, I can describe a set of indicative properties that may help characterize the SE profession and serve as hints for recognizing a new paradigm in software engineering [6, 7] as follows:

(1) It is sufficiently unprecedented to attract a group of adherents away from competing models.

(2) It is sufficiently open-ended to leave all sorts of problems for the new group to resolve.

(3) It is disputable due to resistance to the paradigm shift, especially among those with the greatest knowledge of the old paradigm.

As such, a working definition of "software engineering paradigms" can be hereby suggested as "a coherent tradition of software engineering laws, theories, techniques, practices, applications and instrumentation".

In the past, most frequently observed SE anomalies include undue process overhead, increasingly volatile requirements, critical and sometimes life-threatening failures in meeting the required quality of services (QoS), etc. Similar to other scientific and engineering disciplines, observed SE anomalies provoke new solutions, while failed solutions that quickly accumulate in both number and intensity can lead to the notion of crisis. There are varying ways to align actual and perceived SE anomalies depending on perspectives – e.g., chosen software abstraction or adopted process models. Perhaps most SE professionals are familiar with the popular discussions on SE process models, in terms of how we produce software: planned SE paradigm vs. agile SE paradigm. One serious question to ponder is whether there is indeed a science for the so-called new SE paradigm "Agile" [8].

2.3 Modern computing changed the life of mankind

Let us expand our discussion a bit broader and examine how modern computing changed the way we work and live. One perspective for this alignment is to observe the dependency on computers as a daily tool in ordinary human activities from two viewpoints – complexity of problems and abundance of solutions. Only for the purpose of convenience in presenting the following discussion, we roughly divide SE history into decade-long time periods, starting from 1968, with a possible error range for up to five years (as we are more concerned with "popularity to the public" than "novelty by the inventor") if the readers really care to be precise on the timeline. For example, Java by James Gosling initiated in 1991 [9] but we place it into the 1998–2007 time frame.

Roughly speaking, modern computing before 1968 was largely a privilege for the "fortunate" scientific circles or the "affluent" business undertaking. My story only begins in 1968 when modern computing had forcefully driven mankind to embark on a revolution of information processing. In a sense, this was a paradigm shift in the human quest for information processing at the millennium proceeding from the first spoken word, to the first written language, and to the invention of printing by the Chinese. Table 1 provides a gross view on our progress in modern computing since 1968, with an eye towards SE evolution.

Table 1 Progress in modern computing from 1968

Human dependency	State of computing	Anomalies/crises and complexity of problems	Abundance of SE solutions
From never to seldom (1968–)	Early commercial machines, time-sharing, PL/I, PASCAL	SE was coined due to programming scalability problems	Software factory, high-level language, modularity, waterfall metaphor
From seldom to not much (1978–)	Workstations, networking, Unix/C	Volatile computing environments & requirements	Software development environments, prototyping, early requirements engineering
From not much to occasionally (1988–)	PC, Internet, gigabyte storage, servers	SE in the very-large setting, collaborative SE on the rise	Planned process, SE environments, productivity tools, object-oriented analysis/design/programming, risk management, formal methods
From occasionally to often (1998–)	PDA, data-enriched telecommunications/2G, Java/JVM	Undue process overhead, frequent software failures, WWW syndrome, poor usability, Y2K syndrome	Agile process, components, open-source, human-computer interaction, usability testing
From often to always (2008–)	Tablets, smart phones, B2B/B2C services, mobile Internet/smart phone/3G	Ever-increasing needs for mobility, severe concerns on security, higher expectations on service granularity and timeliness	Model driven, services computing, model checking, micro-services/mobile apps
From always to immersive (2018–)	Cloud/edge/fog computing, IoT, emergence of 4G/LTE	Extreme ubiquity, complexity on system of systems, micro tasks, nationwide initiatives such as NextGen of UK and many derivatives in other countries	Continuous integration/delivery, DevOPs, crowd sourcing

2.4 Paradigm-like shifts in software engineering

With the above cursory recount of SE history, we will now proceed to examining SE activities since 1968 in the form of a series of paradigm-like shifts arranged chronologically. The reason that I do not directly use the phrase "paradigm shift" is because of my choice of a very rigorous definition of the term "paradigm", following Kuhn's teachings.

First, let us ask whether the NATO conference in 1968 started a new paradigm in SE. Since the outset of modern computing after the first electronic digital computer was invented[1)], programming became an important enabler of both scientific and business endeavor. However, during the early days programming was a very exclusive profession and only a very small percentage of professionals could master programming using low-level representations of computing procedures. Early programming work was highly labor intensive, and a programmer's productivity was extremely low. Computing pioneers viewed such scarcity of programmers and low productivity of programming as the earliest sign of software crisis, and it was quite real. One can draw an analogy to the fear that at the turn of the 20th century a large population in the US would need to become telephone switchboard operators during the early days of telephony. As such, I would concur to the awareness of a crisis in the software profession

1) www.atanasoff.org.

voiced at the 1968 NATO conference. SE professionals reached for the first time a consensus that we can recognize as a paradigm shift. We simply could not solely rely on training or retraining electrical engineers, physicists and mathematicians to produce sufficient programming output to fulfill sharply rising demands. Consequently, career programmers were recruited and trained, and higher level representations of computer programs together with systematic production mode, such as that based on the waterfall metaphor, emerged as a way to counter the crisis. What we witnessed in the following decade includes the introduction of some high-level programming languages, proposals of different software process models, and recommendations of SE concepts and principles such as abstract data types, modularity, information hiding, structured analysis, structured design and structured programming.

As software engineers continued on their endeavors, and gradually moved from programming in the small to the programming in the large for about two decades, more experiences and sharper wisdom were gained. A good collection of best practices, such as risk management, was then summarized in a widely popular SE process named the Spiral Model [10]. Many companies began to "spiral it" and considered it a new process if not a new "paradigm". My opinion is that the Spiral Model did not really lead to a new SE paradigm, rather it indeed served well to inspire and motivate the SE profession to adopt best practices available between the 1970s and 1980s. However, the SE profession as a whole also matured from 1968 with a better underlying science, such as the somewhat robust SE economic models, especially the often-cited COCOMO 81 approach to software sizing [11] (and its later edition named COCOMO II). As such, one might consider the period from 1968–1988 and the next ten years as maturing into a distinct SE paradigm. I shall call it the "functional paradigm". During the course of this paradigm, the fundamental computational abstraction was "functional module".

Then, many SE researchers regarded the Agile Manifesto [12] as the marker that ushered in a new SE paradigm. Quite rightfully, many failed software projects should prompt the community to recognize a new crisis. In general, I will not argue that the Agile Manifesto is indeed an important community-wide recognition for the need to depart from the rigid, planned process model. However, we seem to be still in the early stage of developing it into a qualified new SE paradigm. Toward this end, we will need to ask and get clear answers to three questions following the teachings by Kuhn: What are the anomalies observed during the two decades after 1968? Next, as a consequence of these anomalies inflicting SE professionals, was there a sense of crisis widely perceived by the SE community? And, finally, was there a science supporting the agile methods? That is, recognizing a crisis mode is not enough for entering a new paradigm. A paradigm must present a science.

If we treat the concepts of paradigm and paradigm shift as rigorously as taught by Kuhn, it turns out that making a claim that the agile methods brought in a new SE paradigm is an overstatement — it is waiting to become one, but not quite. We do not yet have a reliable science supporting the agile process, even though the corresponding practices, successful to some extent, have made it a promising candidate. For example, at present we simply do not have a robust prediction model in Scrum [13] to give reliable estimates of the required product quality if we extend or shrink the sprint duration for a given set of user stories in the product backlog. Even so, it would still be useful by examining the field to identify all anomalies from the old (i.e., functional) paradigm dominated by the planned/waterfall process model, and declare a new SE crisis like those pioneers did at the 1968 NATO conference. We shall continue working hard to develop a rigorous scientific underpinning, and wish for the birth of a new SE paradigm based on the optimism endowed upon the agile methods.

2.5 Emerging anomalies in SE

Based on the above discussion, to officially recognize a new SE paradigm we will need to establish the legality of declaring that the SE profession is facing a new crisis. The Agile proponents have done a good job to recognize a new SE crisis judging from the scale of failures in the SE profession. I will not repeat that. My analysis will be to weigh the anomalies accumulated to date since we embarked on the road to aware computing [14] two decades ago. Will there be a sizable harm to our profession if we do not declare a crisis mode upon entering the IoT era? I observe three emerging anomalies occurring in today's

SE profession and plaguing prevalent SE practices, as explained below. My purpose is to mention only those pertinent to the succeeding discussion, without any attempt to enumerate a long list of anomalies that may exist today in the SE profession.

Cognitive anomalies. It is not an overstatement that software has become indispensable to modern-day living via convenient and universal access ramps such as smart phones and mobile apps. Consequently, the vast base of day-to-day users of software applications enjoys an ever-expanding opportunity to inject their feedback and demand more and better services. Companies such as Xiaomi knew how to play this type of game well. Software developers are now challenged by their cognitive inability to comprehend the complexity and ever-evolving minds and expectations of their customers, even with their best and earnest effort. Today's software engineers need to better understand computational psychology and cognitive sciences. The emerging notion of "individualize service" (not "personalized service" through limited parameterization) is also well beyond the available processes and tools available to developers to cope with customer demands. Conversely, there may be the end user's cognitive inability to provide proactive, instant, cognitive feedback to the developers in order to receive rapidly improving and better services.

Data anomalies. As stated in the beginning of this paper, we have now entered the IoT era. We can now generate lots of heterogeneous and high-dimensional data as well as noise. Making sense out of such huge volumes of data will prove to be challenging. Based on the Data-Information-Knowledge-Wisdom knowledge structure [15, 16] software developers face steep challenges to extract useful information out of fast accumulating data, identify knowledge patterns, and synthesize wisdom in a specific application domain. The potential dirtiness of data, conceivably bound to happen in the sensor-laden IoT era in a broad spectrum of physical environments for service deployment, makes things even worse. There are practical difficulties to clean up data, and to synchronize and aggregate data obtained in separately established time series, aka time-series data fusion, in order to create useful services.

Connectivity anomalies. Considering the extreme scale of IoT-enabled devices and services networks, 100% connectivity is impossible. In the sense of humans as sensors [17] we will meet an extra dimension of uncertainty because of the sometimes-unpredictable nature of humans. Software engineers are lacking in both their experience and proven techniques in dealing with such kinds of uncertainty due to highly unstable and unpredictable connectivity.

Facing these and other anomalies, software developers need to be fully equipped with powerful and improved techniques and tools in order to support a wide spectrum of IoT applications development. Unfortunately, we are far away from devising feasible solutions to adequately address such anomalies. Recent attempts to provide partial solutions such as the advances in search base software engineering (SBSE) [18], intelligent software engineering [19] and genetic improvement (GI) [20] are all regarded as extending helping hands to SE researchers. As the above anomalies are severe because they significantly hamper our ability to develop software systems amid the fast changing faces of IoT-driven computing, I believe that we have entered another period of software crisis in the 21st century.

2.6 Criteria for feasible solutions

Although there may not be a shortage of good ideas to provide feasible solutions to adequately address the above severe anomalies, I suggest that feasible solutions will need to satisfy the following criteria.

Convenience. In the IoT era, users are increasingly becoming impatient. Any feasible solution to provide services to modern-day users must be convenient as perceived in order to be accepted.

Non-invasiveness. As always, personal concerns such as privacy and respect should be not sacrificed for a solution to be feasible.

Affordability. The pervasiveness of the Internet and abundance of smart devices means that the worldwide community would expect any feasible solutions to be affordable.

Communicability. Any feasible solution will need to enable exchange of useful information between the end users and the developers with a much more rapid (perhaps even in real-time) service-feedback-fix cycle.

Energy-efficiency. Energy consumption has been a burning issue in mobile computing and a more acute concern for powering IoT-centric devices.

Individualization. As a software engineer we should move beyond the so-called "personalized service" via parameterization (mostly through enumeration of alternatives or a predefined range of parameter values, which is hard to be exhaustive), and face the new challenge to support genuine individualization of services for each individual user. This new criterion can draw an analogy from what is now required for other technological domains such as "precision medicine" [21].

At present, there does not exist a comprehensive and wholesome solution to address all three anomalies and satisfy the above criteria in an integrated manner. It is therefore indicative of a fast approaching SE crisis if we cannot meet the unprecedented demands soon as the society is now bursting into the IoT era. I hereby explore deeper into situation-aware computing in search for a highly promising solution, hopefully with some convincing arguments and some optimism.

3 Situation-aware computing

In this section, I plan to highlight the essence of situation-aware computing which is distinct from a more familiar subject that is generally understood as context-aware computing.

3.1 Aware computing

Collectively, our society is now situated in the world of aware computing. In this research domain, many researchers used to base their work on the notion of location awareness. In reality, there is more to context than location [22]. In the IoT era, we will witness the deployment of more and more different types of sensors giving rise to different genres of contexts, for example, location, temperature, traffic flow, air quality, etc.

What is missing is an effective mechanism to organize and fuse diverse contextual data collected from various sensors, including humans [17], for enhanced computational intelligence in software development, maintenance and evolution. In my opinion, that missing link is situation.

Situation awareness is a millennium old topic, and humans are dealing with situations on a daily basis. For example, situation can be regarded as an ancient concept going back to the 5th BC in Sun Tzu's *The Art of War*. Modern day situation-awareness represents a contemporary study originated in the military space — e.g., sensing from touch control for navigation [23], mental models and decision making of pilots [24, 25], etc.

In the academic community, prior studies exist on situation and situation-awareness by John McCarthy [26, 27], Michael Bratman [28], Jon Barwise [29, 30], and recently by Haddawy et al. [31]. In reviewing the academic literature we often encounter the terms related to belief, desire, intention, motivation, etc., and these are terms intimately tied to humans. Thus, situation-awareness is very much a "humanistic" concern. Naturally, a computational abstraction (hereby dubbed "situation") closely mimics modern-day human situations, such as facing air pollution when going out or fighting through traffic jam when driving in Beijing during morning rush hours.

3.2 Situation-awareness vs. context-awareness

SE researchers are often confused between context-awareness and situation-awareness. I suggest that we make a differentiation between these two different, albeit related, concepts based on the following observations.

Context-awareness. When we work with context-awareness, it is customary to fit humans into contextual data. That is, in most context-aware studies, data is the first-class citizen. For example, if a usability study indicates that a service is considered "usable" because the collected usability data suggests

so, human users are considered to be satisfied, which may actually deviate from reality. Such a data-first, quantitative approach often leaves humans with no choice but to accept the fate of "factual data".

Situation-awareness. On the contrary, situation-awareness requires fitting contextual data to humans. In this way, the human becomes the first-class citizen. For example, after a service has been deployed, an unsatisfied human user, while using the service, will need to be checked regardless of what an earlier usability study had suggested. The new data collected from human users in real-time truly matters, and the challenge is how we should and what we can do about it in real-time. This is also where the current adaptive, dynamic systems solutions fail to cope, because the prevalent reconfiguration and failover techniques based on a predefined set of parameters can never exhaustively accommodate all possible real-life situations. It is particularly difficult for dealing with human-centric factors such as the emotional states and on-demand service requirements surfaced at the last minute. In general, software engineers are not accustomed to such a notion.

We need to understand that humans primarily deal with perceived situations, not merely contexts. That is, our lives are primarily driven by situations, not contexts. Situations are outcomes of epistemological reasoning based on the available contextual information. Oftentimes situations can be different even though all contexts are the same when two individuals possess a different mindset (or, mental state) at a particular time or in a specific place. Different chief executives sitting in a "situation room" may very likely perceive different situations and make different decisions when the presented contextual information appears to be the same. Situations are, unfortunately, very complex because humans are highly complex beings.

3.3 A word on complex humans

Perhaps there is no need to convince the readers that humans are mentally, psychologically, physically and physiologically complex. First, humans are never perfect, and such a comment applies equally to software developers and end users. It is equally possible that an end-user may not be able to give needed requirements nor can a software engineer accurately capture user's requirements. Then, we know that humans evolve as "situations" arise. More precisely, human sensory adaptability to contextual cues from the environment is translated into mind adaptability to perceived situations. To understand a human, we need to approach from mental, emotional, motivational, and intentional dimensions of human mental states. Oftentimes the situation can become more complex and unpredictable as humans tend to behave differently when moving from an isolated environment into a crowd. Humans as a research subject have been extensively studied by psychologists, cognitive scientists, physiologists, neuroscientists, sociologists, logicians, linguists, anthropologists, etc., and computer scientists. Note that, for computer scientists, everything must be computable.

3.4 From situation to intention

For logicians, the world consists of objects, properties of objects and relations among objects. And, there are parts of the world, clearly or vaguely recognized in common sense and human language. These parts of the world are called situations by logicians. Events and episodes are situations in time, and scenes are visually perceived situations, etc. [29]. The challenge for computer scientists is whether such concepts can be cast into a computational model.

Another logician's topic intimately related to situation is the notion of intention. When a human user has a desire, he intends to act upon the desire and often persists until his desire is met. Sometimes, we may also confuse desire with intention. According to Malle [32], intention is different from desire in that:

(1) Intention dictates action.

(2) Intention results from some form of reasoning.

(3) Intention requires commitment.

Bratman [28] further restricts intention to the scope of practical reasoning. It also emphasizes future-directed intention so that concepts such as prior intention and derivative intention can help establish an intention hierarchy. Understanding the above terms and concepts are necessary for the software engineers and requirements engineers to effectively perform when dealing with software development tasks such as user vs. system goals, use cases/scenarios, and other related SE entities in our profession.

4 Situation analytics as the foundation for a new SE paradigm

4.1 Situ: support fast-evolving and individualized services

I have previously written about situations to report my work on "Situ" [33], and published a more recent article on "situation analytics" [34]. In both publications, I defined situations to be a triplet [M, B, E], where M stands for mental (hidden) context, such as desire and emotion, B stands for behavioral context, such as clicking a button on the keyboard, and E stands for environmental context, such as a list of recommended products showing on the computer screen. I further define intention to be a temporal sequence of situations (each situation is time-stamped), that is, $\text{Int} = \{\text{Sit}_1, \text{Sit}_2, \ldots, \text{Sit}_n\}_t$ where in Sit_n a human user's desire has been met in order to end an intentional path. In the existing literature of context-aware and situation-aware studies, most authors only consider environmental contexts and some authors considered behavioral contexts. However, it is my belief that we cannot completely define a computational situation model without considering the human mental state that is largely hidden. And, when human mental states change or evolve, computing and software services will need to be modified as well in order to satisfy the user's changing or emerging needs. In modern-day computing, such modifications will need to be completed very rapidly, if not in real-time, so that users can be better or timely served. It is also because of the addition of the mental context, which is individualized in nature, there can be some interesting discussion on genuinely individualized services. Interested readers, please refer to those two earlier publications. As to how we can actually infer what humans want please refer to some pilot studies reported in [35, 36], and the work done by the MIT Affective Computing Lab and the MIT Brain & Cognitive Sciences Department, just to name a few.

4.2 Situation as a new abstraction for computing

As humans are becoming more and intimately embedded in computing, software engineers will need to work with a new, human-centric abstraction that can take into account human desires, human situations and human intentions. As humans are framed by daily situations, situations are a useful abstraction in developing human-embedded computing systems to support our daily living. Moreover, end-users will become ever more influential as they may demand individualized services via situation analytics in real-time.

4.3 Situation analytics for a new SE paradigm

From the perspective of computational abstraction there is another way to analyze the evolutionary paths of the SE profession and define the corresponding paradigm shifts. Towards this end, we will not argue whether the agile process models represent a paradigm shift from the planned process models. Instead, through a different perspective, a computing paradigm can be characterized by the abstraction concept it adopts, not the applicable process model or models. That is, we could view process models as being created to support software development and evolution suitable for the underlying computational abstraction. We shall elaborate this perspective as follows with justifications.

Since 1968, the SE profession gradually and eventually shifted into a functional paradigm. In the functional paradigm software engineers worked with the abstraction called functions (or functional modules) where a lot of studies were founded on the concepts of functional decomposition, structured analysis and design, concepts of cohesion and coupling, cyclomatic complexity metric, software process models, be

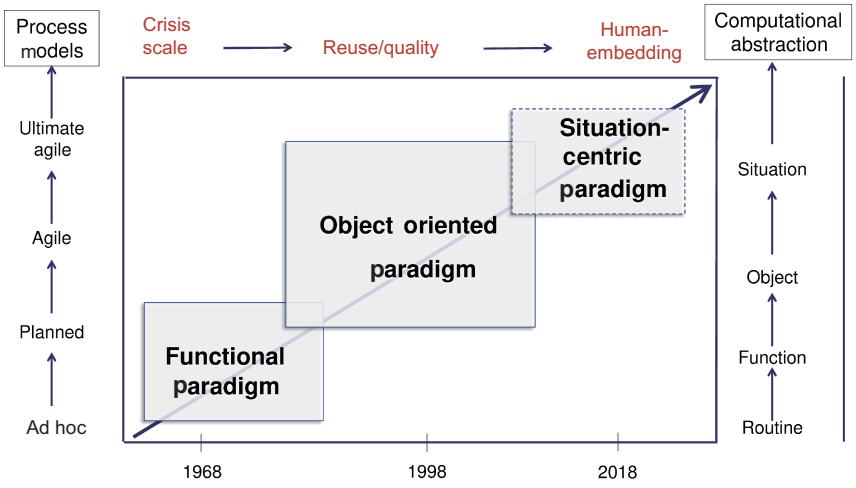

Figure 1 (Color online) Software engineering paradigm shifts.

it waterfall or spiral, and the associated scientific underpinnings contributed by various SE researchers (e.g., Boehm's software economics).

Roughly at the end of the 1980's and the beginning of the 1990's, the SE profession began to popularize "object orientation" (not a new concept though), and started diligently practicing object-oriented analysis (OOA), design (OOD) and programming (OOP). In such as practice, the SE profession essentially shifted into the object-oriented paradigm. In the object-oriented paradigm software engineers work with the abstraction called objects (or classes), and researchers worked hard to create a science to support it, where object-oriented programming languages and object-oriented processes all took a stand and took shape. The SE world began to faithfully embrace the object-oriented paradigm.

What is more revealing is that inside the functional paradigm software developers actually worked largely with the environmental context, which can be very much characterized as inputs and outputs, and both can be conveniently attributed to the environments. The governing computational abstraction during this period was "functional modules" for improved modularity. When the object-oriented paradigm emerged, software engineers began paying more attention to the behavioral context that can be readily modeled by agents, object states, object behaviors, etc., and how they act or react (e.g., stimulus and response). The governing computational abstraction in this paradigm became "objects" and "classes". Among many other plausible reasons, the functional paradigm emerged because of the need to attack the crisis of increasing complexity as well as scalability, and the object-oriented paradigm emerged to attack the crisis for reusability as well as quality.

I argue that neither paradigm paid sufficient attention to the increasingly important yet largely neglected mental context, despite that there had been a couple of loosely coordinated (if at all) works such as user-centric requirements engineering or usability testing, etc. We need direct concern with human users, not just during requirements elicitation, not just during usability testing, but always, even after services deployment. Situation as defined above happens to be effective in capturing the mental context because the M in the [M, B, E] triplet is precisely what had been inadequately treated, if not totally lacking, in the previous two paradigms. I repeat the statement that I made earlier: the missing link is "situation".

At this point during my course of arguments, it may become apparent that my intention is to declare that situation analytics may be considered as a candidate to help pave the road to the next SE paradigm (recall that earlier I declared an emerging SE crisis) where humans must become the first-class citizen in the next generation of computing systems. As such, humans are an indispensable and persistent concern in software development and services evolution. Yes, many researchers have claimed that users are important but there has always been lacking a suitable abstraction to directly work with. Situation, defined as [M, B, E], is a possible candidate to fill the gap. Figure 1 depicts this perspective as paradigm

shifts from the functional paradigm to the object-oriented paradigm, and now to the situation-centric paradigm.

5 Future human-centered software engineering research

Prevalent human-centric SE studies are predominantly consumed by developers' concerns. Although useful, issues tackled by many researchers to improve development environments, streamline teamwork, and devise and engage other productivity and quality measures do not directly address end-users concerns. I argue that developers are only part of the means and end-users should be regarded as the ultimate ends and must be served with high satisfaction. In the past, we simply have not worked at the right level of abstraction to provide direct and timely support for software and services evolution to meet the end-users' ever-changing and constant evolving demands. Software engineers to date have not been adequately armed with techniques and tools to deal with real-time and on-demand human user feedback after the system or service has been deployed. For example, when the system detects that an aging older adult becomes unable to navigate a GUI on a mobile screen, or move a mouse accurately on a laptop, perhaps due to deteriorating small muscles, the system should automatically invoke other form of input (or if options do not exist, automatically search for services on the web and extend its functionality to enable a different usage pattern). To enable such a smart software service, we need to understand both physical and mental states of a human user — conceivably that no exhaustive enumeration can capture all variations. We need to incorporate a sensible human-centric abstraction into our computational model or framework during software development. Such efforts should continue on after deployment of the service. Our work on detecting new human intention to predict changing requirements [36] only marks the start of this new research direction.

Finally, now that we have at our disposal a different computational abstraction that is more conducive in dealing with humanistic concerns, i.e., situation, we need to explore a little deeper into the corresponding computing frameworks to fully take advantage of this new, human-centric and largely unexplored abstraction. Since we have now introduced several key human-centric concepts (human situation, human intention, human desire, human affective state such as emotion, etc.) that are largely neglected from the traditional "user-centric" research studies, I propose that we embrace two new computing frameworks, HEC and EUEC, to support the emerging situation-centric paradigm.

HEC works beyond Human-Centric Computing (HCC) [37] in that the human is now an integral part of a computing system at run time, and the provided services co-evolve with human users. The co-evolving humans in such a computing setting include not only the end users but also other concerned stakeholders, as needed, such as the developers, management staff in the developers' organization who may need to monitor on return on investment (ROI), and system support staff and operators in charge of system operations who may need to ensure ROI through uninterrupted service rendering. EUEC is a special case of HEC where end-users are the dominant concerns. HEC and EUEC based computing frameworks provide many new opportunities.

In the current technological landscape, it may be plausible to enumerate the properties of HEC as follows (shortened as U^3).

Ubiquitous (spatial). In an HEC environment, human presence should always be sensed so that at any point of the human-system interface, when needed, human intelligence and cognitive feedback can be brought in on demand to benefit the overall system operation.

Uninterrupted (temporal). Execution of an HEC system should allow 24×7 sensing of its human stakeholders whenever they have feedback to provide in order to benefit the overall system operation.

Uncertain (natural). With the realization that the computing environment may be volatile and human users may be evolving from time to time both physically and cognitively, a robust HEC system should always be prepared to deal with a degree of uncertainty, large or small.

In order to cope with U^3, future human-centric SE research will become ever more interdisciplinary,

and more collaboration with HCI researchers as well as cognitive scientists can be expected.

6 Conclusion and some speculations

This paper propose that we should work with the correct abstraction in order to deal with the mental context that has been largely neglected from our past practice in developing computer-based solutions. Specifically, I suggest that we can use a new, wholesome computational abstraction called situation for future work in human-centric SE research and professional practice. I gave a definition to situation to make it a computational module by itself. As such, this human-centric abstraction can conveniently and seamlessly co-evolve with the system, environment and human users, and the evolution cycle can become shorter once human real-time feedback can be faithfully captured and correctly interpreted. The ultimate aim is to accomplish on-demand service evolution. To be able to do that, we will need a mechanism to design and program situations, we need a system architecture to ensure the always-on situation-awareness, we need a classification scheme to discern and organize situations patterns, we need effective situation-aware testing methods before deploying newly formed on-demand services, among many other required components to fully establish a situation-analytic software development and evolution environment. There is still a long, long way to go, and much more effort will need to be invested to realize the dream for real-time service evolution. At this juncture, we can only work humbly and expect to make small, incremental improvements.

I also suggest that we forgo the habit of calling different process models as "paradigms". In the past 50 years, we have been practicing different incremental and iterative development approaches [38], including the later agile methods, towards the same overarching goal: deliver software on time and within budget. Since Brooks published his seminal paper [39] to illustrate the essential vs. accidental difficulties in 1987, I suggest that we now add another intrinsic difficulty – effectiveness of human stakeholders to be embedded in the computer-based systems as we move forward. My reasoning about SE paradigm shifts is based on the notion of computational abstraction that has become or may become dominant in the SE evolution path.

Calling situation a new computational abstraction or not, going forward we are destined to include human mental states in our computing endeavors. Research aside, as educators we should consider a new interdisciplinary curriculum to integrate essential elements from computer science, human computer interaction and cognitive science to prepare a new breed of software engineers [34].

I also noticed the interesting timing of my research work on situation analytics when the SE profession began to actively invest into DevOps [40] so that the developers' unit can become an integral part of the enterprise operations. The concepts introduced in this paper timely address the challenges for the HEC system to be inclusive of all concerned human stakeholders, not just software developers or end-users, in business and system operations.

For readers who may be curious about what kind of process models will be needed in order to support such a situation-centric human-embedded SE paradigm, perhaps it should work in conjunction with a future generation of DevOps. I hereby offer a conservative working definition of the ultimate agile method (UAM) that works opportunistically with various stakeholders to benefit the entire enterprise and the users community.

An UAM for software development and evolution leaps from the prevalent agile practices to provide much finer-grained supports to define the ground truth of human situations, and to capture real-time feedback of human users in order to deliver both scheduled and on-demand services and micro-services in real time, in parallel with perceived and actual changing and emerging situational concerns of such human users.

To conclude, I speculate that down the road, tens or hundreds of thousands of "end-user developers" would arrive at the scene owing to the national and worldwide initiatives such as the "Computer Science

for All Initiative"[2)] by the Obama Administration, the YouthSpark[3)] Initiative of Microsoft, etc. When that day comes, will there be a new meaning to the Human-Embedded Computing or End-User Embedded Computing?

Acknowledgements This paper was partially supported by 111 Intelligence Base of High Confidence Software Technologies.

References

1 Bauer F, Bolliet L, Helms H. Report of a conference sponsored by the nato science committee. In: NATO Software Engineering Conference, 1968. 8
2 Xu Y, Helal A. Scalable cloud-sensor architecture for the internet of things. IEEE Internet Things J, 2016, 3: 285–298
3 Mei H, Huang G, Xie T. Internetware: a software paradigm for internet computing. Computer, 2012, 45: 26–31
4 Chentouf Z. Cognitive software engineering: a research framework and roadmap. J Softw Eng, 2014, 7: 530–539
5 Kennedy M R, Umphress D A. People solutions to software problems. CrossTalk, 2011. 16–20
6 Kuhn T S. The Structure of Scientific Revolutions. Chicago: University of Chicago Press, 2012
7 Rajlich V. Changing the paradigm of software engineering. Commun ACM, 2006, 49: 67–70
8 White A S. An agile project system dynamics simulation model. Int J Inf Technologies Syst Approach, 2014, 7: 55–79
9 Gosling J, Joy B, Steele G L, et al. The Java language specification. Pearson Education, 2014. https://www.pearson.com/us/higher-education/program/Gosling-Java-Language-Specification-Java-SE-8-Edition-The/PGM137443.html
10 Boehm B W. A spiral model of software development and enhancement. Computer, 1988, 21: 61–72
11 Boehm B W. Software Engineering Economicsvolume. Upper Saddle River: Prentice Hall PTR, 1981. 197
12 Beck K, Beedle M, van Bennekum A, et al. Manifesto for Agile Software Development. Twelve Principles of Agile Software, 2001. http://agilemanifesto.org/
13 Schwaber K, Beedle M. Agile Software Development With Scrum. Upper Saddle River: Prentice Hall, 2002, 1
14 Chang C, Schilit B. Aware computing. Computer, 2014, 47: 20–21
15 Ackoff R L. From data to wisdom. J Appl Syst Analy, 1989, 16: 3–9
16 Rowley J. The wisdom hierarchy: representations of the DIKW hierarchy. J Inf Sci, 2007, 33: 163–180
17 Wang D, Amin M T, Li S, et al. Using humans as sensors: an estimation-theoretic perspective. In: Proceedings of the 13th International Symposium on Information Processing in Sensor Networks, Berlin, 2014. 35–46
18 Harman M. Software engineering meets evolutionary computation. Computer, 2011, 44: 31–39
19 Witte R, Sateli B, Khamis N, et al. Intelligent software development environments: integrating natural language processing with the eclipse platform. In: Proceedings of Conference on Artificial Intelligence. Berlin: Springer, 2011. 408–419
20 Petke J, Haraldsson S, Harman M, et al. Genetic improvement of software: a comprehensive survey. IEEE Trans Evolution Comput, 2017
21 Neergard L. Obama proposes 'precision medicine' to end one-size-fits-all. Drug Discovery and Development, 2015
22 Schmidt A, Beigl M, Gellersen H W. There is more to context than location. Comput Graphics, 1999, 23: 893–901
23 Biferno M A, Stanley D L. The Touch-Sensitive Control/Display Unit: a Promising Computer Interface. Technical Report, SAE Technical Paper. 1983
24 Endsley M R. Measurement of situation awareness in dynamic systems. Hum Factors, 1995, 37: 65–84
25 Endsley M R. Toward a theory of situation awareness in dynamic systems. Hum Factors, 1995, 37: 32–64
26 McCarthy J. Situation calculus with concurrent events and narrative. 1995. http://www-formal.stanford.edu/jmc/narrative.html
27 Mccarthy J, Hayes P J. Some philosophical problems from the standpoint of artificial intelligence. In: Machine Intelligence. Edinburgh: Edinburgh University Press, 1969. 463–502
28 Cohen P R, Morgan J L, Pollack M E. Intentions in Communication. Cambridge: The MIT Press, 1990
29 Barwise J, Perry J. The Situation Underground. Palo Alto: Stanford University Press, 1980
30 Barwise J, Perry J. Situations and attitudes. Philpapers, 1991, 25: 743–770

2) http://www.centerdigitaled.com/k-12/White-House-Launches-Computer-Science-for-All-Initiative-with-41-Billion-Request.html.

3) Microsoft YouthSpark Initiative, http://www.makecode.com.

31 Haddawy P, Frommberger L, Kauppinen T, et al. Situation awareness in crowdsensing for disease surveillance in crisis situations. In: Proceedings of the 7th International Conference on Information and Communication Technologies and Development, Singapore, 2015. 38

32 Malle B F, Knobe J. The distinction between desire and intention: a folk-conceptual analysis. In: Intentions and Intentionality: Foundations of Social Cognition. Cambridge: The MIT Press, 2001. 45–67

33 Chang C K, Jiang H, Ming H, et al. Situ: a situation-theoretic approach to context-aware service evolution. IEEE Trans Serv Comput, 2009, 2: 261–275

34 Chang C K. Situation analytics: a foundation for a new software engineering paradigm. Computer, 2016, 49: 24–33

35 Dong J, Yang H I, Chang C K. Identifying factors for human desire inference in smart home environments. In: Proceedings of International Conference on Smart Homes and Health Telematics. Berlin: Springer, 2013. 230–237

36 Xie H, Yang J, Chang C K, et al. A statistical analysis approach to predict user's changing requirements for software service evolution. J Syst Softw, 2017, 132: 147–164

37 Jaimes A, Sebe N, Gatica-Perez D. Human-centered computing: a multimedia perspective. In: Proceedings of the 14th ACM International Conference on Multimedia, Santa Barbara, 2006. 855–864

38 Larman C, Basili V R. Iterative and incremental developments: a brief history. Computer, 2003, 36: 47–56

39 Bullet N S. Essence and accidents of software engineering, fp brooks. IEEE Comput, 1987, 20: 10–19

40 Bass L, Weber I, Zhu L. DevOps: A Software Architect's Perspective. Upper Saddle River: Addison-Wesley Professional, 2015