CRITICAL NATIONAL NEED IDEA

Jerry Zhu, Ph.D.
UCSoft
2727 Duke Street, Suite 602
(Telephone) 703 461 3632
Jerry.zhu@ucsoft.biz

Title

Save Billions in Software Industry Each Year with Disruptive Innovation

Keywords

Software engineering, methodology, paradigm, age, transformation, infrastructure, productivity

Save Billions in Software Industry Each Year with Disruptive Innovation

Jerry Zhu, Ph.D.
Jerry.zhu@ucsoft.biz

There has never been shortage in the new stream of software processes, from chaotic process to unified process to agile methods. Each process claims to have solved the problems of its predecessors but inevitably has also introduced new problems often revealed to the users and not the authors of the new process. This paper describes why a novel discipline and associated technology can be developed to replace today's myriad methodologies and dissolve all the problems manifested in today's software industry. It proposes new concepts concerning software and its development and presents unprecedented opportunities and challenges to all enterprises.

All life is problem solving. Our mind-set, the way we see problems, depends not on what is out there but on what we have been "trained" to "see." We may discard—indeed we can be blind to—anomalies that do not fit. The pattern, which shapes our thinking, grants a particular perceptual blindness and rigidity to our perceptions of the world. This pattern of seeing is called perspective. The world perspective is a function of two factors. On the one hand, it depends on the material of experience, which is its foundation; on the other hand, it depends on the conceptual apparatus and the meaning rules that are bound up with it. A change in conceptual apparatus is reflected in a change in problems that one solves on the basis of the same data of experience.

Related to perspective is paradigm. A paradigm is a set of beliefs or basic assumptions about reality, normally beneath the level of awareness and therefore mostly never questioned. The paradigm is the lens through which we look at the world; therefore, it determines our perspectives. Perspectives revealed through one lens are normally invisible through a different lens. To change our strategy is to change our perspective. A change of perspective without a change of its paradigm is the "orthodox novelty," more of the same thing. A change in perspective based on the change of its underlying paradigm is an "emergent novelty," a change in "species."

A company's production process, the process that produces products or services, may be defined along three dimensions:

- Skills: Processes take place between individuals or organizational units as individual or collective skills.
- Objects*:* Processes result in manipulation of objects. These objects could be Physical or Informational.
- Activities*:* Processes could involve two types of activities: Managerial (e.g., develop a budget) and Operational (e.g., fill a customer order).

Any organized human activity gives rise to two fundamental and opposing requirements: the division of labor into various tasks to be performed and the coordination of these tasks to accomplish the activity. The design of software methodology and the structure of project organization, can be defined simply as the way the labor is divided into distinct tasks and the coordination among these tasks. Coordination based on skills, objects, activities correspond to three categories of methodology: lightweight, rightweight, and heavyweight. Heavyweight methodology defines tasks and their order of execution, bears responsibilities and specifies rules. Lightweight methodology places the responsibility back on people who make decisions momentarily on how the process should proceed. A change of coordination based on different dimension is a change in paradigm. A change in task definition within the same coordination is a change in perspective, hence a change of conceptual apparatus.

The most important change taking place today is in the way we try to understand the world and in our conception of its nature. If our views of the world were out of date, our theories and the behaviors they drive would be out of date. Accordingly, we'd continue to miss opportunities and risk serious mistakes as reality and an old way of thinking used to interpret it continue to diverge. The software industry has been experiencing failure methodologies, as demonstrated by the high failure rate of software projects. The so-called agile methodologies are good tries, but they are still not the right solutions, because they cause new problems. The mistake is that software development has been based on wrong paradigms. This paper shows why current paradigms fail and a new paradigm needs to emerge. Organizations and nations that are capable of discerning their passing paradigms and constructing new theories based on the emerging paradigm will become the economic, cultural and political superpowers of the 21st century.

The History and Present of Software Engineering

The term software engineering (SE) was coined at the 1968 NATO conference to introduce software manufacture to the established branches of engineering design. It was a deliberately provocative term, implying the need for software manufacture to be based upon theoretical foundations and practical disciplines that were traditionally used in established branches of engineering. It was believed during the conference that software designers were in a position similar to architects and civil engineers. Naturally, we should turn to these ideas to discover how to attack the design problem. The generic design process is to determine the design's objectives in terms of specific requirements, which will be called functional requirements. To meet the functional requirements, a physical embodiment characterized in terms of design parameters must be created. Engineering design is defined as mapping from the functional space to the physical solution space. Design activities of both fields may be approximately mapped: product planning to requirements, conceptual design to analysis, embodiment design to design and detail design to implementation. Both engineering design and SE clearly differentiate between problem and solution spaces and offer techniques and representations for exploring, bounding and structuring those spaces. Both fields rely on two categories of requirements: behavioral objectives and quality constraints. Both also depend heavily on technological and economic constraints as success criteria.

There are also significant differences between the two. In traditional engineering there is a clear consensus as to how things should be built, which standards should be followed and which risks must be taken care of. If an engineer does not follow these practices and something fails, he or she gets sued. In SE, there is no such consensus and everyone follows his or her own methods. A huge diversity of design methodologies exists in the market today. Project fitness and effective use of methodologies become critical to the success or failure of software projects. There have been numerous examples of projects failing due to an improper methodology being used. For some organizations, the problem is the inadequacy of current methodologies, prompting them to keep looking for different and better ones. However, it has also been found that software engineers' adherence to any methodology, far from facilitating development, only makes the design more problematic. Developers ignore certain aspects of methodologies not from a position of ignorance, but on the more pragmatic basis that certain elements are not relevant to the developments they face.

Four decades after SE was first introduced as a model for the field of software development in 1968, issues surrounding software production remain unresolved. NATO conference attendees did not assert that software development is actually engineering, but rather, they presupposed that it would be fruitful to consider software development to be engineering for whatever benefits that might bring. The outcomes of the field of SE do not resemble those of any other branches of engineering in terms of success rate, error-laden deliverables, intellectual rework and subjective uncertainty.

There is no correlation between project success and the use of the tidy "engineering" development process. Studies have shown that some messy-looking projects succeeded quite nicely, while many process-oriented projects fail quite badly. Developers find no practical advice from the SE model to solve problems on live projects but place their trust in unproven in-house methods. Adherence to methodology has been far from facilitating the development process and has only made the design process more problematic. Methodology adoption is a "fetish of technique" rather than a solution to the design problem at hand.

According to *The CHAOS Report*, the success rate for software projects was a mere 34 percent in 2003. The remaining 66 percent either failed or were severely challenged. "Software failures are unprejudiced: they happen in every country to large companies and small; in commercial, nonprofit and governmental organizations; and without regard to status or reputation. Of the IT projects that are initiated, from 5 to 15 percent will be abandoned before or shortly after delivery as hopelessly inadequate. Many others will arrive late and over budget or require massive reworking. Few IT projects, in other words, truly succeed. The cost of litigation from irate customers suing suppliers for poorly implemented systems must be considered. The yearly tab for all these costs conservatively runs somewhere from $60 billion to $70 billion in the U.S. alone." (*IEEE Spectrum*, *September* 2005) "Moreover, software errors cost the U.S. economy $59.5 billion annually and 80 percent of development costs go to identifying and correcting defects. In fact, few products (of any kind) other than software are shipped with such a high rate of errors." (NIST 2003) SE has failed to contain the inherent complexity arising from frequent analysis paralysis, which leads to confusion as to how the software design process may effectively and efficiently proceed. Both the theory and practice of what constitutes a desirable SE design process remains poorly understood. It appears that as fast as SE makes progress, the demand made on it continues to increase beyond its capabilities. Given the widespread problems evidenced in the field of SE, was it valid for software designers to attempt to emulate their engineering design counterpart? To answer this question, we need to look deeper into the terms "engineering" and "machine" and compare them with software.

The Fundamental Problem of SE

The industrial revolution is concerned with the mechanization of work. There are two concepts: work and machine. Work is real and reduces to atoms. An atom has two properties: mass and energy. Work is defined as applying energy to matter to change the property of the matter. Machine is defined as any object used to apply energy to matter. To design any work is to analyze it: to reduce the work into work elements, mechanize those work elements by assigning machines to them and assign to people those that cannot be mechanized. What we then have is a network of work elements performed by men and machines. We call this network manufacturing.

The input and output of manual work are material objects subject to the constraints of natural law, the law of physics. It is the natural law that dictates how work should be coordinated. That is, the movement of manual labor should be congruent with the movement of objects, governed by the natural constraints, into the final product. Frederick Taylor, the founder of scientific management, destroyed the romance of work. Instead of a "noble skill." it becomes a series of simple motions. Hence the coordination mechanism of manual work is based on the activity as the scientific way of organizing work. The motion of objects is translated into a prescribed network of activities and their interactions. Taylor changed the paradigm from coordination based on skills to coordination based on activities and accordingly revolutionized the world of manufacturing.

All engineering design methodologies belong to one class of machine design system. Any design methodology of this class resides between two ends: art and science. The control of the process of production begins with craftspeople. It is implicit that a craftsperson accumulates knowledge and by doing so gains control over the interaction between the tool and the material being transformed. If one wanted to automate a production process, then the accumulated knowledge would have to be embodied—hardwired—into the manufacturing system. A halfway stage would require that the knowledge be partially embodied in the machinery and partially embodied in the system operators. The knowledge they would have of operating would be gained from assimilating operating manuals and by doing. Control of the process would be based on the knowledge of the nature and dynamics of the interrelationships between the system and its environment and between the components of the system and the input. As we learn more about the process, about the old variables, new variables emerge from the mists of ignorance. Process knowledge progresses from pure art (no variables are identified) to pure science (all variables are identified and understood) as we move from a low- to a high-knowledge stage. To gain perfect control, we need to have perfect knowledge.

There is a natural relationship between degree of procedure and knowledge stage. If we use a high degree of procedure in a low-knowledge-stage production process, unanticipated problems will crop up frequently. If we use a low degree of procedure in a high-knowledge-stage production process, it is inefficient to use lots of expertise to carry it out.

SE is rooted in the machine paradigm. Making software is like making machines. The first step of software design is to propose a collection of product features—what the system should do—and then map them into the solution space. Software, like machines, is functionally decomposed into features that are then allocated to the resulting components. The functional decomposition also becomes anchored in contracts, subcontracts and work-breakdown structures. These product features are equivalent to process variables in manufacturing. The Waterfall approach is at the end of science and the Agile approach is at the end of art. It is well known that product features can't be completely known up front. The Waterfall model uses a high degree of procedure (from requirement to analysis to design to implementation to test), assuming all product features can be known up front. The Agile approach uses zero degree of procedure, assuming no product features can be known up front. Therefore, unlike traditional engineering, SE mostly mismatches the degree of procedure and the knowledge stage, resulting in either higher rework or lower efficiency. Both cause huge waste of resources.

Can SE within the current paradigm evolve to increase a knowledge stage that matches the degree of procedure like traditional engineering does? The answer is no. The success of traditional engineering, the ability to evolve from art to science lies in its underpinnings. Specifically, the fundamental underpinnings of branches of traditional engineering are embedded in physical principles. The knowledge of underpinnings is scientific knowledge, and the knowledge of design is engineering knowledge. The task of engineers is to apply both types of knowledge to the solution of technical problems. They then optimize these solutions within the requirements and constraints of the project. For example, electronic engineers apply their scientific knowledge of electronic properties of silicon and their engineering knowledge of circuit design to build circuits. Mechanical engineers apply their scientific knowledge of mechanical properties of engineering materials and their engineering knowledge of mechanical design to build machines. For an engineer in any engineering branch to be successful in his or her career, he or she must master the corresponding scientific knowledge through education and accumulate engineering knowledge through practicing the profession. Without a solid knowledge of the material mechanics—its

underpinnings—civil engineers would build bridges that they could not guarantee.

Engineering activity is how engineers decipher problems within the set of constraints imposed by the medium in which they are working. The design process creates design elements that are explained based on the corresponding scientific knowledge. For the machine to be workable, its underpinning—the law of nature—must be stable and does not change. Because the design parameters of a machine are based on the explanations of the law of nature, a change of natural law would mean a change of design condition. An airplane would not be workable if gravity or air density would continuously change. Stable underpinning implies objective scientific knowledge. Design is scientific when the explanation of the design elements based on the scientific knowledge is objective.

With SE, the fundamental underpinning is essentially personal opinion embodied in user requirements. It is well known that users do not know what the requirements are. User requirements are speculated and fed into the development cycle and tested in the form of deliverables. As a result, software components, unlike particulars found in nature, are not constrained by natural laws. The lack of natural constraints and physical dimensions in software implies that solutions that make tangible material meet our expectations do not apply. Software engineers, trained in the knowledge of design (e.g., information engineering), do not have the scientific knowledge on which design relies. Due to the lack of complexity-limiting natural constraints, software, if left otherwise unchecked, will tend to expand arbitrarily, toward the only constraint left—the capacity of our brains. This lack of objectivity raises people's expectations beyond all reason of what can and should be achieved within a project's time and resource limitations. Therefore SE, within its current paradigm, will not progress to same maturity as that of traditional engineering and will remain guesswork rather than disciplined inquiry. The only hope is for a change of paradigm, a change from coordination based on skills or activities to one based on informational objects.

The Future of SE

The most important contribution of management in the 20th century, according to Peter Drucker, was to increase manual worker productivity fiftyfold. Taylor worked as a manual worker and studied manual work. Since that time, manual worker productivity began its unprecedented rise—3½ percent per annum compounded—which means fiftyfold since Taylor. On this achievement rest all the economic and social gains of the 20th century. The productivity of the manual worker has created what we call "developed" economies. Before Taylor there was no such thing—all economies were equally "underdeveloped." After Taylor, productivity increases resulting from differences in skills have not existed. There have been none in respect to productivity other than between hard workers and lazy ones or between physically strong ones and weak ones. Productivity increases have been the result of new tools, of new methods, of new technology. Taylor's solution, or what scientific management achieved, was a shift of production coordination based on skills to one based on activity where to great extent muscle was replaced by machines.

The most important contribution of management in the 21st century will be to increase knowledge worker productivity. In Peter Drucker's words, in terms of actual work on knowledge worker productivity we are, in the year 2000, roughly where we were in the year 1900, a century ago, in terms of the productivity of the manual worker. It means that in terms of knowledge work, production processes are currently coordinated based on personal opinion. Software projects fail because of lack of good software professionals. A breakthrough in knowledge worker productivity will depend on a shift of coordination away from personal opinion to knowledge content itself. The throughput of knowledge work is informational objects that are ideas, mental images or representations of what may exist in the world about us. These informational objects are not subject to the law of physics but an arrangement that can be found to be inherent in the objects themselves. This should offer advantages far in excess of those provided by patterns imposed by any personal opinions. The development of a universal ordering of ideas must be based on a study of the notions we believe correspond to the contents of the real world about us. This inquiry must supply rules whereby the notions may be placed in positions that must hold. Personal opinion must, so far as possible, be ruled out as a reason for placement: the only help must come from a careful examination of the structure of the ideas themselves.

This structure will bear directly on evolution and linguistics. Evolution goes beyond what can be described in well-defined language and instead enforces a language that is itself evolving. Certain forms of change and variability in an evolutionary process can be well described. And when described, the variability is represented by constancy, namely by time-independent describing sentences. It is when we interpret the sentences that we add the reality, the described variability. Evolution is a concept that cannot be described in a single formal language. This means that there is no formal language permitting description of its own interpretation process. The pair (description, interpretation) is productive in the sense that the interpretation process may be described in a higher language. This suggests the idea of an evolving language trying to catch up with its own evolution. The language must first evolve and can then

describe the previous level of its evolution. This productive process can continue as an interpretation process grow mature to be descriptive in need of interpretation at the next level of evolution. This productive evolutionary process was described in these terms by Bertrand Russell:

> *That every language has a structure concerning which in the language, nothing can be said, but that there may be another language dealing with the structure of the first language, and having itself a new structure, and that to this hierarchy of languages there may be no limit.*

In developing software, knowledge is all important, and it grows all the time through discoveries and decisions we make. Requirement change is due to this unruled knowledge growth and largely internally, not externally, induced. Carefully planned knowledge growth may greatly reduce the requirement change needed. One such knowledge-growth strategy that minimizes requirement volatility is to identify the purpose or the essential invariant need, called abstraction, that is more objective than subjective. Once such need is identified, exhausted and modeled, it is time to search for next-level essential invariant need. This second-level need realizes the first-level need through transformation of the first-level need, or subclass first-level need (e.g., biology is a special kind, or subclass, of physics), by adding more information. This process can go on until all needed requirement information is encapsulated in this hierarchy of requirement model that is parsimonious, invariant, precise and inclusive. The result is that we develop a stable requirement model that is emergent and open ended. Related theories include hierarchy theory and the methodology of deductive science. It reaches the goal that none of today's methodologies can achieve: complete business and software alignment based on the law of deduction.

The Emerging Discipline and Technology

An engineering discipline is a consensus among its practitioners about systematic instructions of how things should be built. Guided by the discipline, engineers apply engineering knowledge in the context of scientific knowledge to solve technical problems. Civil engineering has a consensus. Consensus in SE has yet to emerge. Consensus lies in the theory and its coupling with practice. Consensus in traditional engineering is possible because it contains well-established scientific knowledge and the application of it in design is objective. In other words, the explanations of engineering design and its resulting components must be objectively determined in terms of scientific principles outside of personal opinion. The principles serve as stable underpinnings in which design is embedded. In SE, the stable underpinnings will be the business context and the linguistic hierarchy on which design is based. To develop software is to build knowledge level by level from the most abstract to the most concrete as an emergent process. The most abstract level is the business process model, and the most concrete level is the code.

The resulting technology, application life-cycle management (ALM), would potentially substitute all ALM technologies current in the market. ALM, according to Forrester, is described as the coordination of development activities to produce software applications. Nearly one-third of enterprises already use ALM, and almost half are aware of it. Forrester defines AML as:

> *The coordination of development life-cycle activities, including requirements, modeling, development, build, and testing, through: 1) enforcement of processes that span these activities; 2) management of relationships between development artifacts used or produced by these activities; and 3) reporting on progress of the development effort as a whole.*

The scope of ALM is described as follows:

- ALM is a discipline, and a product category as well, and can be accomplished without supporting tools.
- ALM is not merely a collection of life-cycle activities, but rather synthesizes them into a coherent system with an emphasis on interaction of these activities.
- An ALM solution is the integration, not merely a collection, of life-cycle tools. Effective tool support for ALM connects the practitioner tools within a development project, such as an IDE, a build-management tool and a test-management tool. It's the connections, rather than the tools themselves, that make up an ALM solution.

ALM, as a newly accepted concept, has gone through accretion from AML 1.0 to the purposeful design of AML 2.0. AML 1.0 is point-to-point tools integration. These tools may contain redundant and inconsistent ALM features. The microprocesses that regulate practitioner efforts are embedded in each practitioner tool, and the macroprocesses that regulate interactions between these practitioners live in the integrations between these tools. This means that process assets aren't versionable assets, can't share common components and can't be managed as a portfolio. For this reason, most shops focus their process governance efforts on paper-based process assets, hoping that they correspond to the processes instantiated in their tool sets.

Forrester defines the architectural ingredients of ALM 2.0 as presented below:

- Practitioner tools assembled out of plug-ins
- Common services available across practitioner tools
- Repository neutrality
- Use of open integration standards
- Microprocesses and macroprocesses governed by externalized work flow

There is no solution in the market that possesses all characteristics of ALM 2.0 as defined by Forrester. A key issue is the choice between a single-vendor platform to get the best ALM capabilities and a free pick of practitioner tools on its own merit but sacrificing on ALM. It has been accepted that organizations need ALM to have any hope of tackling the software crisis of poor delivery. An appropriate ALM strategy is also a sign of maturity within the organization. But the maturity of the organization is hard to reach if well-established discipline does not exist in the industry. The huge diversity in design approaches used by practitioners currently indicates the immaturity of the software industry and makes it impossible to create an industry-wide ALM solution.

If the discipline is at fault, technology appropriation alone won't achieve the intended goals of ALM. Can we have an ALM strategy with ensured successful delivery without using any ALM tools? That is a test of ALM discipline. If the answer is no, we have a defective discipline that in turn leads to a defective AML product. Given the immaturity of the software industry today, any AML solutions offered in the market currently are built on defective discipline and hence are defective solutions. The emerging discipline based on language hierarchy serves the role of an industry-wide discipline so that an industry-wide ALM solution becomes possible. The single discipline of ALM logically determines the selection of the practitioner's tools, integration and the implementation of the ALM 2.0 vision that addresses concerns in all situations.

Build the Innovation Infrastructure

Emergent industries, based on disruptive technologies and their associated discontinuous-innovation base, are critical to the growth of economies. Consequently, ways to encourage and assist the development and market penetration of these innovations are of interest to both policy makers and corporate strategists. Disruptive technologies and discontinuous innovations could create entirely new industries or replace the requirements for success in existing industries (e.g., the electronic watch replacing the mechanical watch). Theoretical transformation of SE not only alters the way software is developed but also transforms business models between software buyers and producers. Therefore, the resulting technology is disruptive innovation. It is important to differentiate between technological and organizational innovation and between continuous and discontinuous innovation.

Continuous Technological Innovation	Technological innovation along a particular trajectory of technology competence development
Discontinuous Technological Innovation	Technology competence development is taken away from the existing trajectory, and it assumes some form of abrupt change in the business environment. Such changes often are a combination of technological, social, political and economic factors. The firm has a feeling of being "out of breath" or "beyond its comfort zone" in terms of technology competence.
Continuous Organizational Innovation	Organizational change in processes and structures without changing the identity of the firm for better efficiency
Discontinuous Organizational Innovation	Organizational change in identity, customer value or boundary for sudden transition in organizational capacity

In technology-intensive industries such as the IT industry, competitive advantage is built and renewed through discontinuous innovation that creates a new family of products and business and results in a new "product-technology-market" paradigm that greatly improves the value offered to customers. Discontinuous innovation offers greater competitive advantage but might not improve market penetration and, as a result, requires greater attention from academics and government. However, due to their relative novelty, discontinuous innovations lack the required infrastructure. Infrastructure is necessary for the development of radical products that are very different and new. For example, when electrical lighting was first introduced, it lacked a supporting infrastructure.

There are upstream and downstream infrastructural components. Upstream infrastructure is related to technology development. The growth in technological knowledge and competence results in a four-stage progression: 1) basic research, 2) state of industrial manufacturing, 3) bottlenecks to technological development and 4) stable new technology. In stage 1, the scientific base or principles that the innovation is based on exist, but products and supplies do not. As knowledge regarding the science and the ability to apply it grow, it is possible to manufacture prototypes. In stage 2, competing standards and industrial processes exist. Firms are forced to design and build their own production equipment. Bottlenecks or constraints that hinder use or production are encountered and overcome in stage 3. Once these bottlenecks to production and/or use are

addressed, the innovation becomes a stable new technology (stage 4).

Downstream infrastructure relates to the demand side, or "market pull" for the products that develop as a result of discontinuous innovation. In an emergent market based on discontinuous innovation, the market passes through four stages: (1) nonexistent market channels, (2) initial market acceptance, (3) market augmentation and (4) new markets. Initially, these markets are faced with nonexistent market channels (stage 1): not only are there no distribution channels for new products, but potential customers are not even aware of the technology's existence. Firms that enter the market at this time must realize that they must make an effort to develop infrastructure, since potential customers need to be made aware of the technologies, and time and effort will be required before customers are prepared to accept the new products. If advocates of the emergent industry do not focus on raising the awareness and acceptance of potential customer groups, the eventual acceptance of products by customers will be delayed, perhaps indefinitely.

For the IT industry, discontinuous technological innovation requires discontinuous organizational innovation for the producers as well as market infrastructure transformation. A change of software development process means a change of organizational structure and processes of the producer. It also changes the behaviors of producers and consumers in the market because the innovation redefines products and services and their exchange patterns in the marketplace. These can be the two biggest barriers for adoption of the innovation, because current organization's structure and market infrastructure lock them in place against change. Being the largest IT consumer, the federal government is in a perfect position to lead the innovation adoption by first utilizing the new discipline. Hence, it is in the government's best interests to invest the necessary research and development and then use the technology it supports.

The United States has been at the center of science and technology. It has become more challenging to maintain this leadership. Staying in the forefront of science and technology meets both long- and short-term national needs. As the national debt hits a historic record, this innovation will save tens of billions of dollars in waste in IT spending and operating costs by the government. "It's time we once again put science at the top of our agenda and work to restore America's place as the world leader in science and technology," President-elect Barack Obama said in a radio address when he selected four top scientific advisers. "Whether it's the science to slow global warming, the technology to protect our troops and confront bioterror and weapons of mass destruction, the research to find lifesaving cures, or the innovations to remake our industries and create 21st-century jobs—today more than ever, science holds the key to our survival as a planet and our security and prosperity as a nation."

Professor Russell Ackoff said that we are in the early stage of transformation between two ages: the industrial age and the systems age. An age is a period of history in which people are held together by, among other things, use of a common method of inquiry and a view of the nature of the world. To say we are experiencing a change of age is to assert that both our methods of trying to understand the world and our actual understanding of it are undergoing fundamental and profound transformation. The research proposed in this paper serves as catalyst to the transformation in the software industry. Because we live in a world where change has always been accelerating, the competitive edge of modern organizations lies in their ability to absorb rather than resist change. Because modern organizations critically depend on their information systems for daily operations, information systems are required not only to support corporate processes but also to be adaptive in response to evolving business requirements. A transformation in the software industry will increase the capacity of information systems by a different magnitude. Because of the industry-wide innovation and its impacts on other industries critically dependent on software, success requires a necessary infrastructure that is beyond the reach of any single business, let alone a small business.

Conclusion

SE historically emulated the traditional engineering design approach in the belief that software developers were in the same position as civil engineers to attack problems. Civil engineering has a clear consensus on how things should be built and what standards should be followed; however, SE is different—it has no such consensus: everyone promotes his or her own methods. Furthermore, unlike bridges that are normally on spec and on budget and do not fall down, software is seldom on spec or on budget and almost always falls down. This is due to the difference in underpinnings, physical principles in traditional engineering but arbitrary human imagination in SE. Establishing the missing stable underpinnings in software development holds the potential to transform the theoretical foundation of SE into one comparable to the foundation of any branch of conventional engineering. This disruptive innovation replaces requirements for success in software industry and is critical for the growth of economies and creating new businesses and products. However, industry-wide innovation lacks the desired infrastructure. Accordingly government support would be crucial in bringing the unproven technology to market.