10.6977/IJoSI.201903_5(3).0006
Usharani Hareesh Govindarajan, D. Daniel Sheu, Darrell Mann /
Int. J. Systematic Innovation, 5(3), 72-90 (2019)

Review of Systematic Software Innovation Using TRIZ

Usharani Hareesh Govindarajan[1*], D. Daniel Sheu[1], Darrell Mann[2]
[1]National Tsing Hua University, [2]IFR consulting
[*]Corresponding author, e-mail: hareesh.pillai@ie.nthu.edu.tw
(Received 1 March 2016; final version received 20 March 2019)

Abstract

This paper attempts to review the use of TRIZ, Theory of Inventive Problem Solving, in the field of software innovation. TRIZ finds widespread applications in many fields of engineering such as mechanical, electrical, electronics, chemical, materials, industrial engineering, etc. Even, TRIZ has its applications in management and strategies. However, the applications of TRIZ in the field of software engineering to solve problems that arise during phases such as software design, development, coding, testing, and maintenance seems to be in its very initial phase. The primary objectives of this paper are to review and consolidate the current state of the art in the area of TRIZ for software related problems by a literature review. The current review will help academicians and industry experts to understand the current state and to visualize a possible future direction.

Keywords: Software TRIZ, Systematic Software Innovation, TRIZification of software, Software TRIZ review.

1. Introduction

Innovation can be viewed as an invention that has been successfully translated into commercial success. An invention is an event that helps in finding a better way of doing things. Inventive thinking or, more generally, 'creativity', has traditionally been viewed as a random occurrence that occurred anywhere from office brainstorming sessions to coffee breaks to morning showers -- an 'anywhere anytime phenomenon'. It was also assumed that the occurrence of such 'thinking outbursts' was untraceable and almost impossible to replicate within a given timeframe. If 'invention' is about the generation of 'ideas', innovation is about the conversion of those ideas into commercialization. It is well-known that at the present time, 98% of all innovation attempts are ended in failure (Mann, 2012). Within the world of Information Technology, the failure rate is currently slightly worse, running at a failure rate of 98.5%.

1.1 Systematic Innovation Background

Systematic Innovation (SI) is a field which concerns about developing or using systematic methods/processes to generate innovative ideas for Technical, Strategic, or Business aspects of Opportunity Identification and/or Problem Solving. (Sheu, 2015). Figure 1 shows a proposed classification of Innovation Methods (IM) in which SI is a major part of it. (Sheu, 2015). (Sheu and Lee 2010). TRIZ is the Russian acronym for "Theory of Inventive Problem Solving" and is a branch of systematic innovation with ample support levels available in the form of community, training, publications and enthusiasts. TRIZ has circulated around the world fairly successfully in more than 50 countries as indicated by (Bradford, 2016.) The TRIZ philosophy and applications have been expanded into various usage fields such as the ones shown in Figure 2. (Sheu , 2015). (Sheu and Lee, 2010). Table 1 shows the typical extended application areas of Modern TRIZ (SI). This paper focuses on the review of systematic software innovation using TRIZ. This is a very new area having relative much less developments compared to other application areas however with great potential for further developments.

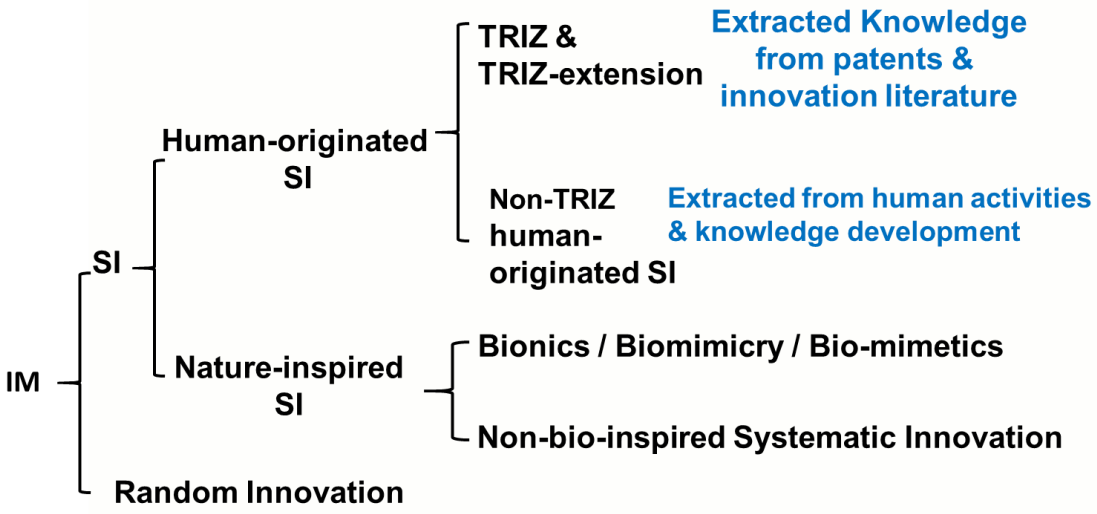


Fig. 1 A proposed classification of Innovation Methods

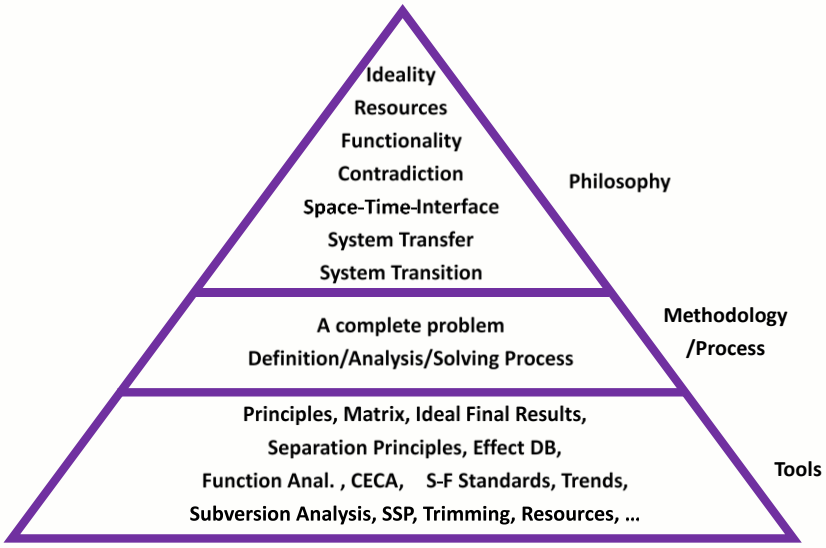


Fig. 2 Hierarchical View of TRIZ (Mann, D. L. 2009, Sheu, D. D. 2015)

Table 1 Extended Application Areas of Modern TRIZ (SI)

- Identifying Innovative Products & Services
- Solving Engineering problems
  - New and existing product developments/improvements
  - New and existing process/equipment developments/improvements
  - Patent circumvention/regeneration/enhancements
  - **Software innovation**
- Management/Service Applications
  - Establish Innovation Strategies/Business Model innovations
  - Service innovation
  - Identifying Organizational conflicts & solving them
- Combine with other tools to solve problems:
  - VE; QFD; FMEA; 6-Sigma tools, Lean, Kepner-Trego; …

2. TRIZ Philosophy

TRIZ is a philosophy, a set of systematic thinking methods, and a set of tools with software. Figure 2 shows a hierarchical view of TRIZ structure. At the base level, there are many tools for problem-solving. At the middle level, there is the methodology or process, which is a complete problem definition, analysis and solving process. In the process, it employs the various tools from the base level at appropriate stages of the process to define, analyze, and solve problems. Regardless at the tools level or methodology level, they are all based on some powerful philosophies known as Pillars of TRIZ. The traditional TRIZ has 4 philosophies which are Ideality, Resources, Functionality, and Contradiction. (Mann, 2007.) Identified Space/ Time/ Interface as the fifth pillar of TRIZ. (Sheu, 2015.) Identified System Transfer and System Transition as the sixth and seventh pillars of TRIZ. These are fundamental philosophies that make TRIZ powerful.

Figure 3 shows a conventional problem-solving approach in which experiences and trial and error are used to take a specific problem into specific solution(s). A typical TRIZ Problem-solving Process is shown in Figure 4. Traditional problems solved by TRIZ are problems in technology and engineering context. Such problems require new, out of the box solutions unknown before (Souchkov, 2007-2014). TRIZ philosophy believes that in the center of most inventive problems lies a contradiction. A contradiction consists of a logical incompatibility between two or more propositions. TRIZ solves two types of contradictions. The technical contradictions which exist in the system prevent it from reaching a specific goal or achieving the desired solution and the physical contradictions occur when a parameter of the problematic system has incompatible needs to satisfy negative requirements, likely opposite requirements. The TRIZ method aims to eliminate contradictions in order to solve problems. Technical contradictions can be solved through 40 inventive principles, while physical contradictions can be solved through separation principles which include at least separation in space, time, system level, relationship, etc.

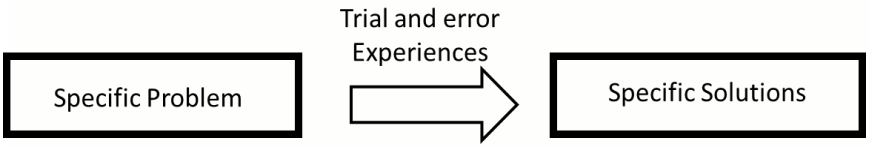


Fig. 3 Conventional Problem Solving Approach

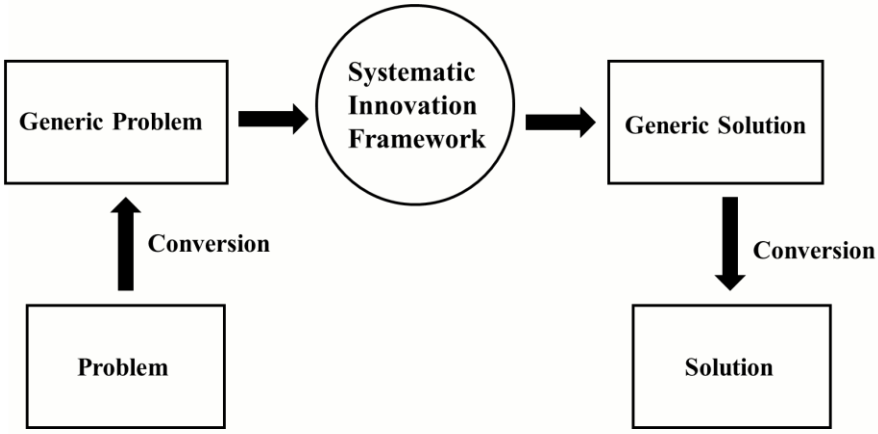


Fig. 4 TRIZ Problem-solving Process

2.1 Some TRIZ tools relevant to software engineering

Innovation involves the deliberate application of information, imagination, and initiative in deriving greater or different values from resources, and includes all processes by which new ideas are generated and converted into useful products. Systematic innovation is the process of methodically analyzing and solving problems with a primary focus on identifying the correct problem to be solved and then generating innovative solution concepts Khomenko N. states that in order to be universal, tools and techniques should be as general as possible. However, general tools tend to bring general solutions. The ideas generated are sometimes so general that it might not be of any practical use. To summarize we need to customize generic tools to produce highly optimized results and to customize we need the theoretical background (Khomenko, 2010). This paper aims to build this theoretical background. The Classical (Russian) TRIZ methodology contains a host of tools. "A review of TRIZ and its benefits and challenges in practice" published in technovation 2013 summarizes widely used tools (Ilevbare et al. 2013). In this section, an introduction to some tools that can be applied in software engineering is provided below (Toivonen, 2014).

1. 40 inventive principles - Inventive principles are generic problem solutions (contradiction elimination). They are compiled from mining patent databases and other sources of problems and their associated solutions. So far according to TRIZ terminology, there are 40 identified Inventive principles.
2. Contradiction matrix - A contradiction in the broadest sense is a problem to be solved. Contradictions are always between one or more parameters that need improvement against one or more parameters that are a hindrance and prevent the improvement. The contradiction matrix helps to reduce or eliminate such contradictions by point-

ing users to solutions which are known as inventive. Inventive principles are built on the analysis of technical systems patents. Moreover, the matrix is a statistical analysis of the use of these inventive principles in technical domains. Applying such statistical analysis in another domain helps to get a different perspective to cross-disciplinary problem solving. The general core concept is that while a problem may be unique to a given domain the abstract essence of the problem might have already been solved in another domain. Statistical analysis helps to understand this perspective thereby helping to solve problems.

3. Trends of evolution - TRIZ problem-solving visualizes evolution as a process that has a finite point (a point beyond which the need to evolve is not needed or not possible) Systems evolve with time through time and trends of evolution tools help collectively summarize the evolution patterns in various areas, suggest the evolution trend for a problem. By mapping system's current state regarding these trends it is possible to discover areas where there is a lot of potential for improvement.
4. Function and Attribute Analysis (FAA) -FAA is a technique to form an understanding of the current state of a system by mapping its elements and their interactions. FAA also helps to map both the positive and negative intangibles of a system.
5. Perception Mapping - Perception mapping is a method for approaching complex problems by mapping the network that the individual perceptions form and identifying which perceptions hold key positions in that network and focus improvement efforts on those areas.
6. Nine Windows Method (AKA system operator Method) - helps to look at the problem from different viewpoints regarding time (the past, present, future) and abstraction level (system, microsystem, macro system) It is flexible and can be used to understand a problem, discover resources and generate solutions.
7. Ideal Final Result -This tool allows the mapping of what perfect looks for different stakeholder groups regarding different attributes of the system (like speed, cost, etc.). The results are documented in the matrix where on dimension is formed by stakeholders and the other by system attributes. The matrix is useful for identifying contradictions. Ideality is given the below formula.

$$\text{Ideality} = \Sigma \text{ Benefits} / \Sigma \text{ Cost} + \Sigma \text{ Harm}$$

8. Resource Tools -By mapping the available resources in a system it is possible to generate solution ideas that rely on free and/or underutilized resources. Resources can also act as a trigger for solutions. Recourses can also be intangible like human cognitive biases.

2.2 Available TRIZ Software

There have been several attempts over the last 20 years to encapsulate TRIZ heuristics, tools, and protocols into software tools. This section is a review on generic TRIZ software's that have been pre-customized to solve software engineering problems. The first of these, 'TECHOPTIMIZER' from Invention Machine and 'Innovation 'WORKBENCH' from Ideation, were very much focused on the codification of TRIZ ideas from the world of engineering, and particularly the world of mechanical engineering. Other tools have subsequently been derived by a multitude of other players, such as GOLDFIRE by Invention Machine Corporation (subsequently sold to HIS Markit), PRO-INNOVATOR by IWIN company, IDEATION BENCHMARK by Ideation are examples of commercial software's available in this domain. etc. are commercial software's available in this domain. Other derivative software from TRIZ include 'PATENTINSPIRATION', which has sought to obscure much of the complexity of TRIZ behind smart solution search algorithm design. None of these providers have created any software specifically for the IT world. There are also a number of individual researchers or teams have developed some proprietary

software for various TRIZ tools. However, they are not dedicated for software innovation. So far, the only place where specific 'IT-TRIZ' software tools will be found are those offered by Systematic Innovation Ltd, in the form of the MATRIX+ and EVPOT+ (Trends) tools, both of which contain specifically focused IT-industry problem types and solution databases

3. Review of Systematic Innovation in Software Engineering

Information technology (IT) refers to all jobs that have to do with computing for all aspects of managing and processing information. IT involves ever-expanding areas of computing such as the internet, telecom equipment, engineering, healthcare, e-commerce, computer hardware, software, electronics, semiconductors, and computer services solving problems. IT problems are problems arising anywhere in the given above list. Troubleshooting is an example of IT problem. Troubleshooting is often applied to repair failed products or processes on a machine or a system. It is a logical, systematic search for the source of a problem in order to solve it and make the product or process operational again. Troubleshooting is needed to identify symptoms, determining the causes and solving it. Software reliability estimation is another are in computer science where TRIZ can be applied to increase flexibility, extensibility, and customizability. This section is a review of systematic publications in line with prior TRIZ application to solve software engineering problems (Domb, 2003). There have been several attempts to encapsulate TRIZ heuristics, tools, and protocols into software engineering for a few years now. (Kluender, 2011). (Ng, 2013). This section is a summarization of such attempts.

Figure 5 shows the events relevant to systematic software innovation. Systematic innovation saw its first publicly visible application in the field of software engineering in the year 1999 when Kevin C. Rea applied the technique to solve a concurrency problem. His observations were published in the TRIZ journal (Rea, 1999), (Rea, 2000), (Rea, 2002), (Rea, 2005d). Around then, many academicians, enthusiasts, and researchers have applied various TRIZ tools broadly in the field of Computer Science. This section is a review of many such prominent works. Even though many case studies of TRIZ applications to solve software engineering problems are not available for the public due to host company's non-disclosure policies, for clarity sake a timelines graph below list prominent published works (available in open forums and published in English language) in time order from year 1999 to year 2015 followed by a short summarization of the publications. Because of the language barrier, some Korean and Chinese publications are not included in the chart below.

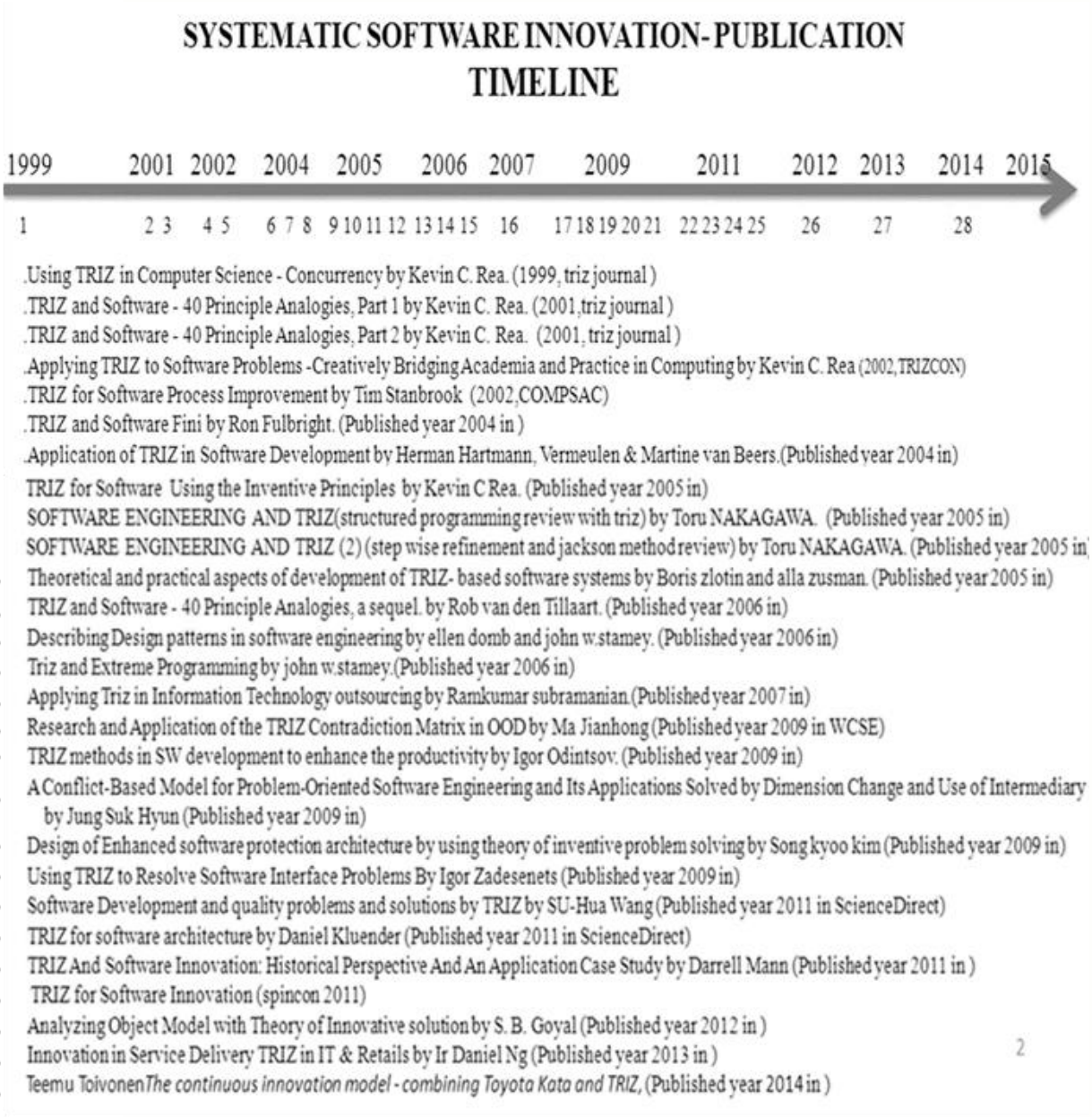


Fig. 5 Timeline of Papers Published

In the year 1999 Kevin C. Rea, a research scholar and consultant, attempted to break psychological inertia towards usage of TRIZ in the field of software engineering by demonstrating a solution to a software concurrency problem. He used the Su-field (substance field) analysis and the principles of contradiction in his demonstration which was published in the TRIZ Journal (Rea, 1999). The next year Rea published papers in 2 parts which were a conversion of the 40 engineering inventive principles in Information Technology or software context (Rea, 2000). In 2002 Rea published a paper titled "Applying TRIZ to Software Problems" which gave an overview of various techniques that could be used in inventive software engineering. The paper also had given an example of implementing a multisport communications buffer using Su-field model. Thereby starting off a new area of applying TRIZ in software engineering, some experts also consider Rea's work as the beginning of software TRIZification.

In the year 2004, Fulbright published a paper titled "TRIZ and Software Fini" which was an exten-

sion of Rea's work of 2001. The paper demonstrated software context of a few inventive principles whose equivalence was not given by Rea in his earlier work (Fulbright, 2004). The work was followed by Herman Hartmann, Vermeulen and Martine Van Beers. In their paper titled "Application of TRIZ in Software Development" supported the discussion on the subject how software engineering can also use TRIZ philosophy to solve problems. The publication focused on area's centric to software engineering such as Inventive Principles, Fast Algorithms, Moore's law, software size, architecture development and trends of technological evolution (Hartmann et al., 2015.)

Darrell Mann in the year 2004 through his article in TRIZ journal gave an introduction to the field of science with a comparative example of software versus a mechanical engine system. He also customized TRIZ pillars and contradiction matrix according to software requirements. The subject context of Darrell Mann was expanded in his book "Systematic Software Innovation" published in the year 2008 (Mann, 2008).

Kevin C. Rea in the year 2005 published the paper "TRIZ for Software Using the Inventive Principles" the objective of writing up was to showcase an example thereby breaking some amount of psychological inertia towards problem-solving using TRIZ. The contradictions that the example dealt with are "waste of time" against "accuracy of manufacturing" and the solution was stated via inventive principles numbered 24 mediator and 26 copying (Rea, 2005). Toru Nakagawa, a Japanese innovation scientist, in the year 2005 wrote a two-part paper (Nakagawa, 2005a,) (Nakagawa, 2005b). The first part titled "Software Engineering and TRIZ (structured programming review with TRIZ)" explains the concept of structured programming with center around a workaround for go-to statements used in programming constructs. "Go-to-less programming from the TRIZ prospective". TRIZ principles 1 (Segmentation), 6 (Universality), 7 (Nesting) were used for making the program easy to understand and advocated 'Structured Programming'. The second part titled "Software Engineering and TRIZ (2) (stepwise refinement and Jackson method review)" is a refinement of Jackson's method of structured programming in correlation with TRIZ along with some discussion on 'Prior-reading technique'. TRIZ principles like Segmentation, Local Quality, Intermediary, Prior Action, and Homogeneity have been used to make the comparison.

Boris Zlotin and Alla Zusman in the year 2005 published a paper, "Theoretical and practical aspects of the development of TRIZ- based software systems," which in detail describes the need for TRIZ software and the people who needed to develop such systems with the requirement's and Consideration's need to make it keep in mind while building such systems (Zlotin and Zusman, 2005). TRIZ and Software - 40 Principle Analogies, a sequel published by Tillaart in the 2006 is an analogy of 40 inventive principles explained in a software context (Tillaart, 2006). The work is an updated analogy of Rea's work with some extra consolidations and value in the form of examples. A similarity study between Altshuller's 40 inventive principles and software design patterns by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides also known as "The gang of four" (Domb and Stamey, 2006).. The paper discusses time-space trade-off followed by a similarity study of design patterns with TRIZ such as adapter pattern with principle of mediator, bridge pattern with extraction principle, composite and iterator pattern with principle of universality, decorator pattern with the principle of nesting, flyweight pattern with the principle of transition to a new dimension and proxy pattern with the principle of parameter change

John W. Stamey published TRIZ and Extreme Programming (XP) which is an introduction to Waterfall model of software development with a comparative study of XP model to TRIZ Inventive Principles (Stamey, 2006). An Information Technology outsourcing analogy to 40 inventive principles under the paper titled "Applying TRIZ in Information Technology Outsourcing" by Ramkumar

Subramanian in the year 2007 has discussions on various laws in reference to inventive problem solving and its outsourcing equivalence (Subramanian, 2007).

"Research and Application of the TRIZ Contradiction Matrix in OOD" by Jianhong Ma published in the year for the field of object-oriented software design is proposed, paper further deals with the abstraction of parameters in object-oriented software design, construction of contradiction matrix, the application of the matrix and the establishment of design patterns. "TRIZ methods in software development to enhance the productivity" by Igor Odintsov published in the year 2009 shows TRIZ tool application in various Software Development Life Cycle stages (Odintsov, 2009)."A Conflict-based model for problem-oriented software Engineering and its applications solved by dimension change and use of intermediary " published by Jung Suk Hyun in the year 2009 deals with problem-oriented software engineering via an author specified problem-solving model named butterfly model (Hyun, 2009). The paper also solves a shopping cart problem using the proposed model.

"Design of enhanced software protection architecture by using the theory of inventive problem solving" published by song- kyoo kim in the year 2009 is on the stochastic software protection using closed queues with unreliable backup(Song, 2009). The paper performs stochastic multilayer software protection analysis and random backup module protection based on TRIZ contradiction principles 1, 10 and 11."Using TRIZ to resolve software interface problems" published by Igor Zadesenets in year 2009 is a description to the problem-solving process using TRIZ (Zadesenets, 2009). The TRIZ models in the discussion here are the object-relationship model and the cause-effect model and how software problems can be solved using TRIZ methods. "Software Development and quality problems and solutions by TRIZ" published by Su-Hua Wang in the year 2011 is a description of quality problems in the field of software engineering and its solution using TRIZ (Wang, 2011). The paper discusses TRIZ fundamentals and tools followed by problems in software development followed by the applicability of TRIZ in software problems in broad scale.

"TRIZ for software architecture" (Mann, 2011) describes inventive principles and the contradiction matrix in a software context. The paper re-architectured a flight simulator using TRIZ tools with similarity analysis of software quality attributes with technical parameters of a contradiction matrix and future scopes of these tools were proposed.

"TRIZ and Software Innovation" by Darrell Mann in the year 2011 gives a historical timeline style review of innovation in the field of computer science. The discussion is on 26 newly uncovered patterns for discontinuous software evolution which are placed under 3 groups namely physical, temporal and interfacial. The paper concludes with a case study of unmanned ariel vehicle control systems to enhance operational capability by using TRIZ contradiction matrix.

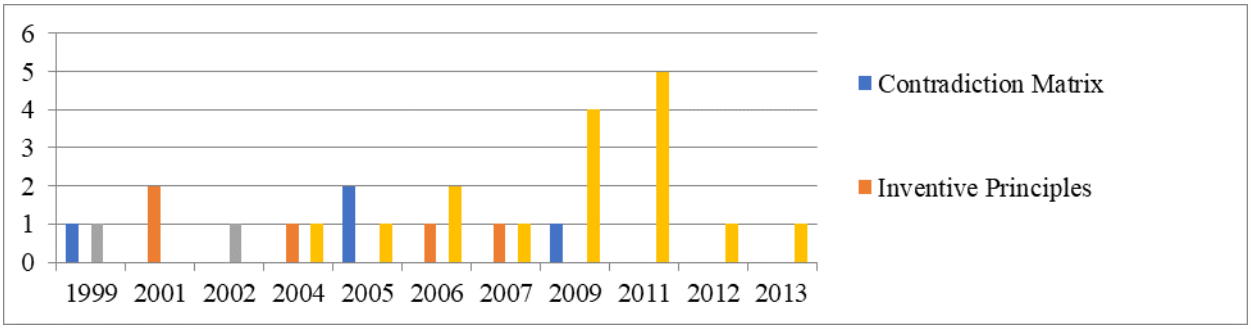
CRAFITTI consulting an innovation think tank distributed a comprehensive online presentation in the year 2011 titled "TRIZ for software innovation" which discusses various aspects of software innovation like patent analysis, elements of TRIZ contradiction, ideal final result development philosophy, and various trends laws of evolution and some advices on how to embed TRIZ into an enterprise. "Analyzing object models with theory of innovative solution" by S. B. Goyal published in the year 2012 gives a co-relation to Object Oriented Modeling Paradigm and TRIZ applicability in Object-Oriented Environment (Goyal, 2012). The paper gives an introduction to Object Orientation and Modeling technique UML (Unified Markup Language) and TRIZ. The paper concludes with a process of applying TRIZ to problem-solving in object-oriented modeling

A comprehensive presentation titled "Innovation in service delivery TRIZ in IT and retails" by Ir Daniel Ng available online from November 2013.The presentation starts with an introduction to TRIZ basic contradiction and the inventive principle is covered followed by few case studies. The presentation also contains various publication details in TRIZ and concludes with case sharing about internet mining and retail industry.

3.1 Review Consolidation

The review which takes into account publications since 1999 shows the most explored areas in TRIZ for software suggests contradiction matrices and inventive principles as the most popular areas of exploration as shown in Table 2. Detailed expansions of these attempts is in the earlier section..

Table 2 Areas of TRIZ exploration in software context



3.2 Book Review

The time order of some relevant books regarding to systematic software innovation is shown in Figure 6.

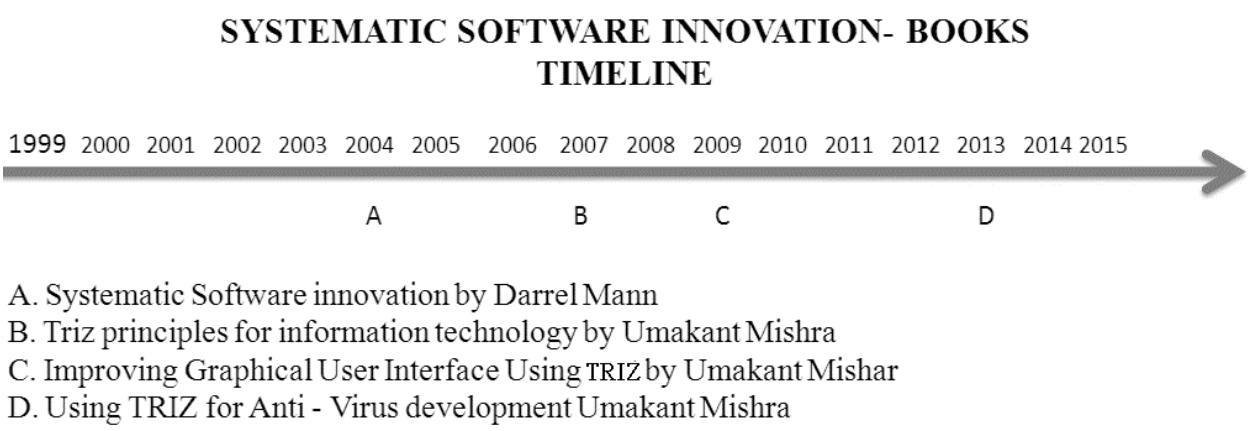


Fig. 6 Time Order of Books (Published in English Language)

Some contexts of the books are briefed below.

A. Systematic Software Innovation by Darrell L. Mann

Darrell Mann has integrated various TRIZ techniques and philosophy in this book which was re-written several times the final draft was published in 2004. The book is targeted towards the software engineering area and is a guide for professionals wanting to apply TRIZ in software engineering domain.

B. TRIZ Principles for Information Technology by Uma Kant Mishra

The books started as a manuscript presented in TRIZCON-2007. The response to the manuscript was overwhelming from around the world. The book summarizes how inventive principles can be used in IT domain by demonstrating patent analysis, case studies and pictorial examples against each principle of the invention. The book was also acclaimed highly by Toru Nakagawa of japan and was translated in the Japanese language later.

C. Improving Graphical User Interface using TRIZ by Uma Kant Mishra (published in the year 2009)

The book is for GUI designers and TRIZ researchers. Graphical user interfaces have become critical to the interaction element in almost all products even though there is a great improvement in GUIs used a generation earlier there still are limitations. TRIZ principles like “Ideality”, “Functionality”, “Trends”, "Contradictions", “Inventive Principles” etc. could be used to solve such problems. The book cites more than 100 inventions from US Patent Database and explains how the contradictions in the prior art methods have been overcome by applying very simple but innovative concepts.

D. Using TRIZ for Anti-Virus Development - Building Better Software through Continuous Innovation by Uma Kant Mishra.

"Using TRIZ for Anti-Virus Development" is a book by Uma Kant Mishra, on the application of TRIZ Techniques for improving the Anti-Virus technology. The book demonstrates how various techniques of TRIZ, including Contradictions, Inventive Principles, Inventive Standards, Ideality, Su-Fields, Resources, and Trends of Evolution etc. are useful for taking the Anti-Virus technology forward to the next generation.

4. Current State

The preceding descriptions of activities and milestones concerning the convergence of TRIZ and ‘software’ suggests that the level of effort has been considerable. Even a cursory examination of the world of IT professionals, however, would rapidly reveal that the impact of this effort has been minimal. The large majority of IT professionals, in other words, will still have never heard of TRIZ. Refer to Figure 7 In terms of the Gartner Hype Cycle (Fenn et al. 2008), neither TRIZ nor its ‘Systematic Innovation’ successor would be perceived to have entered even the ‘technology trigger’ start point of the curve. This fact should provide some clues as to the likely future scope for TRIZ/SI activities in the software world. Before we enter that discussion, however, it is worthwhile to exploring some of the possible reasons why TRIZ/SI has not yet been viewed as a ‘Technology Trigger’ within the world of IT.

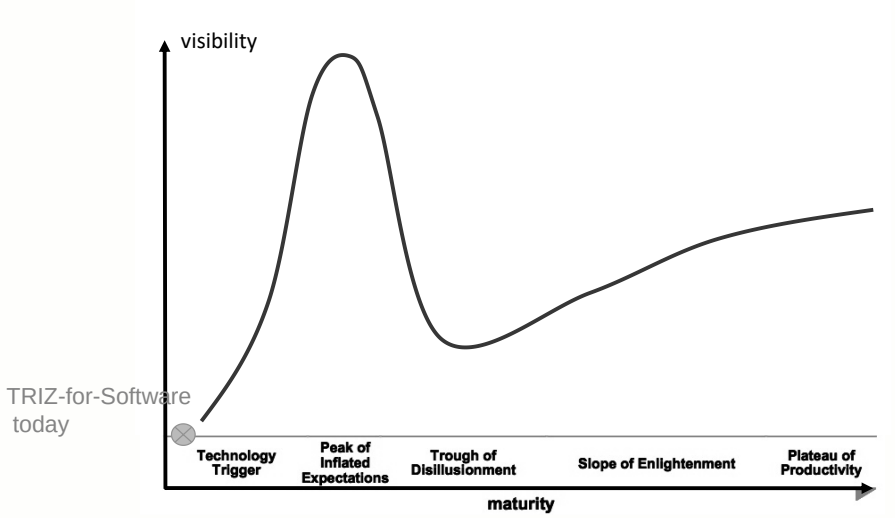


Fig. 7 Hype cycle and ‘TRIZ/SI for Software’ position

A review of the previously discussed TRIZ and software literature from Section 3 of this paper reveals two distinctly different approaches to the challenge of applying TRIZ to problems and challenges

within the IT world. The first of these approaches is to be found in nearly all of the texts discussed. It is an approach based on re-application of already established TRIZ tools, protocols, and procedures to IT problems. In theory there is nothing wrong with this strategy since a large part of the basic premise of TRIZ is 'someone, somewhere already solved your problem' and so an analogous problem in the world of, say, mechanical engineering, should according to the theory provide solution clues to a person working in the IT sector. In the case of truly universal findings like the 40 Inventive Principles this 'analogous worlds' assumption has proved to be valid. An extensive investigation by multiple authors has failed to reveal a '41st Principle' that is found in the world of software that is not found in any other sector (Tillaart, 2006).

Beyond this finding, however, the relevance of the analogical approach has been found to be extremely limited. Attempts to apply the classic Altshuller's Contradiction Matrix – a tool created in 1973 by the software industry even existed – is virtually meaningless since the 39 parameters that make up the sides of the Matrix bear little, if any, resemblance to parameters that a software engineer would consider to be relevant. Similar disconnects can be observed with attempts to deploy the TRIZ S-Fields and Inventive Standards tools: the level of abstraction required for software engineers to meaningfully use the tools is significant. Considerable enough at least that were a software engineer new to TRIZ to accidently read one of the papers or articles on the subject their likely reaction would be either, a) this has absolutely nothing to do with me, or, probably more likely, b) the solution being proposed in this case study is a really bad solution to the problem and so the method through which the solution was derived must therefore also be bad. Which is a way of saying that there are few, if any, published papers that contain anything that a software engineer would think to be a 'good solution'? Not to mention the fact that in the large majority of published cases, the mediocre result was not derived by actually using TRIZ in the first place.

When Mann and the Systematic Innovation Company entered the world of software through the eventual publication of the Systematic (Software) Innovation book, it was the result of an extensive research, commenced in 1999, to go back to the original TRIZ philosophy and to actually analyze hundreds of thousands of breakthrough software solutions. Three big things emerged from this decade-long and still going research:

a) The large majority of the classical TRIZ tools were meaningless in the context of software problems. Making an analogous connection between a parameter in the 39x39 Altshuller Contradiction Matrix and a software problem might generate some Inventive Principle solution suggestions, but these suggestions would be largely irrelevant to the specific problem at hand. (Mann, D. L. 2008) reports an average relevance of less than 20%. If the TRIZ tools were to ever become relevant to software engineers, new research and new tools would need to be created.
b) Working with actual software engineers and examining the sorts of problems they encounter during their work it very quickly became clear that their biggest problem was *not knowing what the problem was*. The roots of this problem come from the prevailing software industry challenge of the gap between the software architects and coders and their system 'customers'. The customers tending to not know what's possible, and the coders not knowing what their output is actually going to be used for. A big part of this gap may be seen to involve the 'unspoken' – lack of tacit knowledge and lack of understanding of the emotional drivers that affect peoples' behavior.
c) Also through the experience of working with software engineers, whenever they do encounter a problem it is very rarely what might be classified as a 'software problem'. Far more likely was that the problem was a management problem or a problem with the supporting technical systems which the software was expected to control. Once a solution could be configured, it could almost

always be coded. The need for solving 'coding' problems was and still is very much the exception rather than the rule.

As a consequence of these findings, the architecture of the Systematic Software Innovation book changed considerably compared to other TRIZ tomes (section 3.2, Book A). Firstly it compiled together all of our research findings to build software-bespoke new tools. Second, and more importantly, it introduced new tools and approaches from outside TRIZ that would better assist software engineers in understanding their real customer needs rather than the ones contained in the specifications they published.

Despite all of the time and effort that went into the production of the Systematic Software Innovation text and associated software tools, it has made a very little impact beyond a small number of IT service organizations. Perhaps not surprisingly this disappointing outcome has provoked a significant additional program of research to reveal the underlying reasons for this lack of recognition by the software community on TRIZ and the new suite of Systematic Innovation tools.

One thing for sure is that there is no shortage of innovation attempts taking place in the IT world. Figure 8 shows another version of the Hype Cycle, this time showing the relative positions of some of those attempts along the cycle. To the best of the authors' knowledge, none of these attempts has made any use of TRIZ/SI. They are all innovation attempts borne of a perceived customer need followed by trial-and-error solution finding. Given the choice of deploying a repeatable innovation process (e.g. TRIZ) or using trial-and-error, most industries would tend to opt for the more efficient approach. So, paradoxically, the IT world – which is one of the most innovative on the planet right now – is the one showing the least inclination to using more efficient processes. Why might this be?

One very logical answer to the question might be that trial and error works in the virtual world because it is possible to make very rapid solution iterations at negligible cost when compared to what needs to occur to make a solution iteration in the physical world.

Another one is that ideas spread much faster in the virtual world. No sooner has one coder found an interesting solution to a customer need, every other coder in the vicinity is able to see what has been done and is able to easily reproduce it. Helped in no small part by the fact that in most parts of the world it is very difficult to protect the IP that might be associated with a new piece of software.

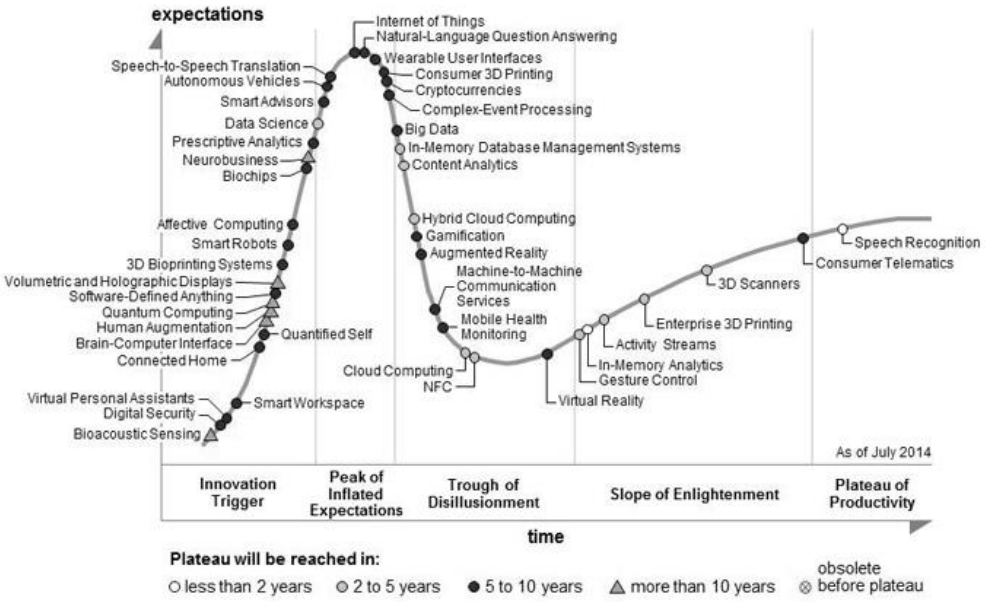


Fig. 8 Assorted IT Industry Innovation Attempts on the Hype Cycle

Taken together, these two factors perhaps indicate that the world of IT innovates 'well enough' already without the need for any kind of systematic process. We will return to that thought in the next section of the paper. Before that, however, we will make a small diversion to investigate what TRIZ and Systematic Innovation might have to tell us about the likely future direction and evolutionary potential of the software.

4.1 The 'Ideal' Software?

One of the pillars of TRIZ/SI is that all systems evolve in a direction of increasing ideality towards an 'Ideal Final Result' destination defined as the point when the system delivers all of the desired benefits ('functions') with zero negatives (typically defined as 'costs' and 'harms'). Because fundamentally, as a system becomes more ideal, the number of effective solution possibilities becomes progressively smaller. This is counter-intuitive for most players and nearly all industries. Refer to Figure 9, what it in effect means, if we plot an evolution story that connects current players with the evolutionary end point, it quickly becomes possible to identify the likely winners and losers. The Figure shown here for the IT industry as a generic whole makes no attempt to be comprehensive in terms of mapping a compendium of current players on the left-hand-side of the image, but it does contain the current biggest ones – the primary one being the IT Services industry and the millions of coders that work within it – and also the ones that will inevitably eventually supersede them. If the 'ideal' software, on the right-hand point of the cone, does everything it needs to do 'by itself' (is 'autopoietic' in the vernacular), then fundamentally it does not require programmers to create it anymore. Software Developers that aren't associating themselves with the emerging worlds of affective computing, or Big Data Analytics or expert systems and genetic algorithms beware, evolutionary convergence clearly says your days are numbered.

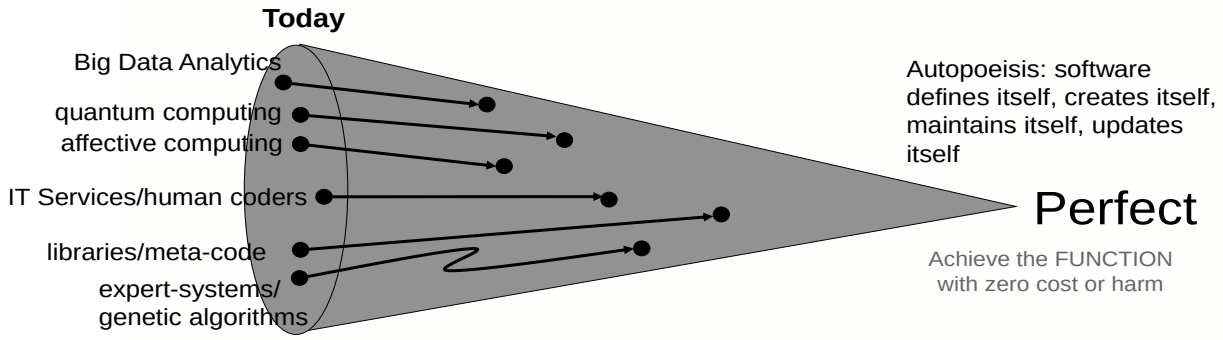


Fig. 9 Convergent Evolution of the IT industry towards its 'Ideal Final Result'

So much for the evolutionary destination of 'software' and the software creation industry, we now shift the focus of attention to the Trends part of the TRIZ story in order to examine some of the key evolutionary jumps that the industry will likely make during the journey towards the autopoietic 'ideal final result' destination.

4.2 Evolution Potential

The original TRIZ research into the evolution of systems found within the physical world uncovered a number of patterns of evolution that have subsequently come to be described as the 'Voice of the System', or 'signposts' that direct innovators towards ideal solutions. The Systematic Software Innovation research program sought to identify whether there were equivalent signposts to be found in the IT industry EvPot+ software analog contains 26 such evolution patterns. A parallel piece of research to do the same job in the world of business and management uncovered 32 (so far) patterns in that describe the evolution directions of an enterprise (Mann, 2009). Figure 10 illustrates a composite of Trend patterns from the IT and business worlds that are relevant to the IT industry.

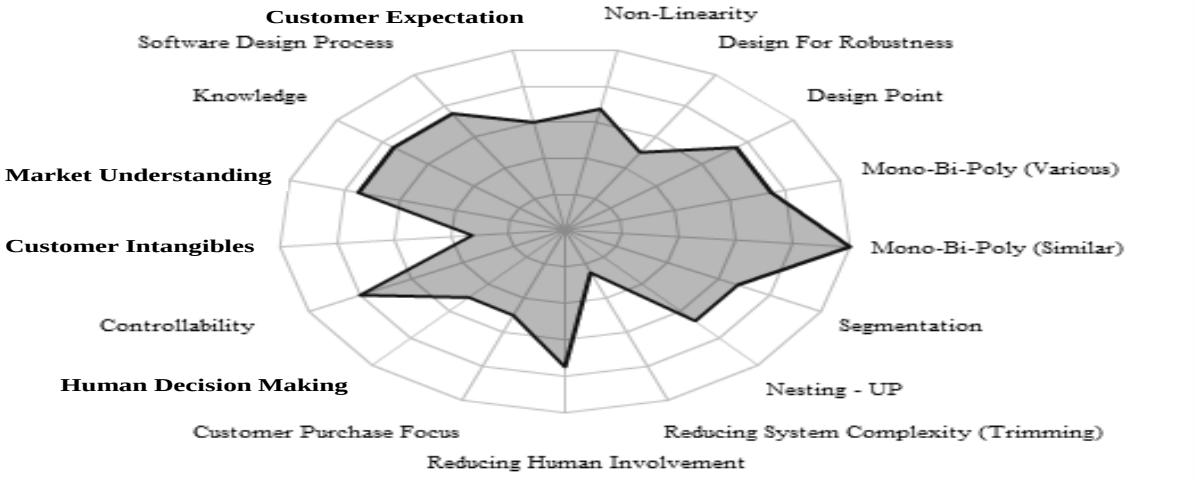


Fig. 10 Composite Evolution Potential Radar Plot of IT Industry

As is the usual convention with the resulting 'Evolution Potential' plot, each Trend is represented by a spoke on the radar plot, and the plot details how far along a particular trend the industry has a whole

has thus far evolved. At this point in the evolutionary history, some 65% of the Evolution Potential has been utilized. Which in turn means that 35% of the possible evolution jumps the industry could make have thus far not been exploited. What might some of this untapped potential be able to tell innovators about the future likely solution directions of the industry as a whole? Again, this is a question that goes beyond the scope of the purpose of this paper, but by way of helping us to answer the earlier stated question about the future of innovation methods within the IT world, here are a few clues provided by the Trends:

1. *Controllability Trend* – software takes on predictive ('feed-forward') capabilities in order to anticipate its own future needs, and eventually becomes autopoietic.
2. *Reducing Human Involvement Trend – human is progressively removed from the system at both the coding, but also specifier and customer ends of the value chain*
3. *Customer Intangibles Trend – software is increasingly capable of tapping into the emotional and 'unspoken' real needs of customers and users*
4. *Nesting (Up) Trend – software is increasingly integrated into higher level systems; source code becomes absorbed into higher level 'meta-languages' (Mathematica, et al, where the user is able to design algorithms without ever having to learn how to code)*
5. *Design For Robustness Trend – the software evolves to become more and more error-proof, to eventually become 'anti-fragile' – attempts to break the system end up making the system stronger*
6. *Trimming Trend – all of the superfluous software (the IT Services industry right now might be thought of as millions of smart people re-inventing the same basic wheels) will be 'trimmed' from systems such that what is left delivers all of the intended capability without unneeded excess.*
7. *Customer Expectation Trend – the software industry will shift from 'service' to 'experience' (taking care of the intangibles) to, eventually, 'transformation', at which point it will take over the responsibility for delivering the intended outcomes from the customer.*
8. *Design Point Trend – the software algorithms will learn how to adapt and reformulate themselves according to different operating regimes…*
9. *Knowledge Trend - …until eventually it will be able to sense and adapt to the prevailing and emergent contexts of a given user situation.*

And so with these clues firmly at the fore of our thinking, back to our final question…

5. Future Scope

Will TRIZ/SI ever find a role in the IT world? The answer to this question has to be yes. The answer is clear since Systematic Innovation fundamentally encapsulates a host of 'universal truths' – everything evolves towards an ideal end state, and will do so through a series of contradiction-solving, discontinuous (s-curve) jumps that follow a set of Evolutionary 'Laws'. In this sense, the IT world is no different from the physical world (Mann, 2011).

Beyond that high-level similarity, however, the virtual and physical worlds diverge considerably in the manner in which innovation happens. In the physical world, efficiency is important and every new solution iteration is expensive, requiring considerable human activity to make things happen. Consequently, it is important that enterprises looking to innovate in the physical world provide those expensive people resources with appropriate innovation efficiency raising skills. Training thousands of people in TRIZ/SI makes sound economic sense in this context.

In the virtual world, where ideas transfer very quickly, there is far less justification for training large numbers of people. 'All' that is required is that a small number of people are skilled in the universal

truths of TRIZ/SI to be able to encode them into systematic creativity algorithms.

There is a considerable irony in this story. TRIZ is and has always been about distilling the 'DNA' of innovation. Altshuller himself published a book called 'The Innovation Algorithm'. Having created at least the start of such an algorithm, it becomes highly code-able. And the moment it does become coded and the IT world is presented with even the start of a meaningful 'computer-aided innovation' capability – especially one also equipped with (highly predictable) 'self-updating' capacity – then it removes the need for thousands of coders to do the creativity and innovation solution generation job manually. Paradoxically, by working out the 'innovation algorithm', TRIZ has ruled out the likelihood of widespread TRIZ deployment. At least from a visible-to-the-lay-person perspective. Most coders will never come to hear about TRIZ, but much of TRIZ will come pre-coded into the software kernels they get to work with. Only an elite few need ever know the 'Innovation DNA' to be able to upload it into

The authors believe that the future of TRIZ in the IT world is assured. Just not through training thousands of coders. But rather by being the first and best to encode the universal truths TRIZ research has revealed into a Systematic Software Innovation algorithm.

The big outstanding challenge in that world is how the inherent (monetary) value that comes through the TRIZ knowledge can be captured. In the physical world, it has been possible to capture at least a part of the monetary value of it through training large numbers of people, publishing books and selling TRIZ-based software tools. These models fundamentally can't and won't work in the virtual world. Millions of software engineers cannot be allowed to continue reinventing the same wheels because customers increasingly cannot afford them. There is, therefore, enormous business pressure to evolve software creation capability in the autopoietic direction. Perhaps we should contemplate inserting that challenge into the Systematic Software Innovation algorithm?

tomorrow's software systems. The IT services sector is already hitting fundamental contradictions associated with increasing competition and reducing margins. In the West, the contradiction has been evident for a number of years already – as evidenced by the extraordinary amount of outsourcing of code development work to the developing parts of the world. But because the contradiction is present and causing pain, there is every incentive to resolve it by innovating the software development process such that, as outlined in the previous section. Software that 'writes-itself', 'maintains itself, and 'updates-itself' solves massive business challenges for western organizations and so they have every incentive to derive and create such solutions. The recent release of TRIZ-based software systems like PanSensic being a case in point. Once a customer has installed a smart PanSensic dashboard, they are already halfway to automatically revealing future innovation opportunities and using the Trends and Inventive Principles to generate solutions. All without any need to teach any of their personnel anything at all about TRIZ.

6. Conclusion

Collaboration between different professionals is more and more necessary now (Khomenko, 2010). Systematic innovation can help in this constructive collaboration. TRIZ is expected to play a major role in the design and development of software systems providing new capabilities that far exceed today's levels of autonomy, functionality, usability and reliability. TRIZ absorption can be accelerated by close collaboration between academics and industry. This review paper provides detailed introduction to systematic innovation followed by brief introduction to TRIZ with a review of key tools inside the framework. An analysis of commercial and academic TRIZ software is presented next followed by a detailed literature review of systematic innovation in software engineering, finally views of subject matter experts in TRIZ area are presented to understand the current state of TRIZ application in software engineering and future scope. The authors hope that the review in this

paper will help academicians, researchers and software companies understand the current industry dynamics and help achieve investments in TRIZ for enhancing their existing and future software development process and products.

7. References

Bradford, G. (2016). *TRIZ is now practiced in 50 countries*. Machine Design, March 21, 2016. http://machinedesign.com/contributing-technical-experts/triz-now-practiced-50-countries, accessed on April 21, 2016.

Domb, E. (2003.) *TRIZ for Non-Technical Problem Solving*, The TRIZ journal Apr.

Domb E and Stamey J. W. (2006). *Describing Design patterns in software engineering*.

Fenn, J. and Rascino, M. (2008). *Mastering The Hype Cycle: How To Chose The Right Innovation At The Right Time*, Harvard Business School Press.

Fulbright, R. (2004). *TRIZ and Software Fin*i.

Goyal, S. B. (2012). *Analyzing Object Model with Theory of Innovative solution*.

Nakagawa T. (2005a). *SOFTWARE ENGINEERING AND TRIZ* (structured programming review with triz),

Nakagawa T. (2005b). *SOFTWARE ENGINEERING AND TRIZ (2)* (step wise refinement and jackson method review),

Ng, D. I. (2013). *Innovation in Service Delivery TRIZ* in IT & Retails.

Odintsov, I. (2009). *TRIZ methods in SW development to enhance the productivity*

Rea, K. C. (1999) *Using TRIZ in Computer Science - Concurrency*, the TRIZ journal

Rea, K. C. (2000a). *TRIZ and Software - 40 Principle Analogies, Part 1*, the TRIZ journal.

Rea, K. C. (2000b). *TRIZ and Software - 40 Principle Analogies, Part 2*, the TRIZ journal.

Rea, K. C. (2002). *Applying TRIZ to Software Problems -Creatively Bridging Academia and Practice in Computing*, TRIZCON2002.

Rea, K. C. (2005). *TRIZ for Software Using the Inventive Principles*.

Sheu, D. D. (2015). *Mastering TRIZ Innovation Tools: Part I,* Agitek International Consulting, Inc, 4th Ed.

Sheu, D. D. and Lee, H.K. (2010). *A Proposed Classification and Process of Systematic Innovation*, International Journal of Systematic Innovation,1(1) , 3-22.

Hartmann H, Vermeulen and Beers M. V. (2004) *Application of TRIZ in Software Development*.

Hyun, J. S. (2009). *A Conflict-Based Model for Problem-Oriented Software Engineering and Its Applications Solved by Dimension Change and Use of Intermediary*.

Ilevbare, M. I, Probert D and Phaal R (2013) *A review of TRIZ*, and its benefits and challenges in practice.

Khomenko, N(2010). *Keynote presentation for 6th TRIZ symposium* in Japan, Tokyo. September 2010

Kluender, D. (2011). *TRIZ for software architecture in ScienceDirect*.

Mann, D. L. (2008) *Systematic(Software) Innovation* IFR Press

Mann, D. L. (2006). *Updating TRIZ: 2006-2008 Patent Research Findings*, keynote address, 4th Japanese TRIZ Symposium.

Mann, D. L. (2007). *Hands-On Systematic Innovation For Business & Management*, IFR Press.

Mann, D. L. (2011). *TRIZ and Software Innovation: Historical Perspective And An Application Case*.

Mann, D. L. (2012) *Innovation Capability Maturity Model: An Introduction*, IFR Press,.

Song, K. K. (2009). *Design of Enhanced software protection architecture by using theory of inventive problem solving*.

Souchkov, V. (2007-2014). *Breakthrough Thinking With TRIZ For Business And Management: An Overview*, ICG Training & Consulting, retrieved from www.xtriz.com.

Stamey, J. W. (2006.). *TRIZ and Extreme Programming*.

Subramanian, R. (2007) *Applying TRIZ in Information Technology outsourcing*.

Tillaart. R.V.D. (2006) *TRIZ and Software - 40 Principle Analogies, a sequel*.

Toivonen, T. (2014). *The continuous innovation model - combining Toyota Kata and TRIZ*.

Wang, S. H. (2011). *Software Development and quality problems and solutions by TRIZ in ScienceDirect*.

Zadesenets, I. (2009). *Using TRIZ to Resolve Software Interface Problems*.

Zlotin B and Zusman A. (2005). *Theoretical and practical aspects of development of TRIZ- based software systems*.

AUTHOR BIOGRAPHIES

Dongliang Daniel Sheu is a Professor at National Tsing Hua University in Taiwan since 1996. Before then, he has 9 years of industrial experience in the electronic industries with Hewlett-Packard, Motorola, and Matsushita. Daniel received his Ph.D. degree in engineering from UCLA and MBA degree from Kellogg Graduate School of Management at Northwestern University. He also holds a B.S.M.E. degree from National Taiwan University and an M.S.M.E. degree from State University of New York at Buffalo. He is currently the President of the International Society of Innovation Methods, Honorary President of the Society of Systematic Innovation, Editor-in-chief of the International Journal of Systematic Innovation, and Area Editor of the Computers and Industrial Engineering Journal. His areas of interest include Systematic Innovation (TRIZ++), Innovation Management, Patent Technical Analysis, Equipment Management, and Factory Diagnosis.

Usharani Hareesh Govindarajan currently teaches in the role of an Adjunct Assistant Professor in National Tsing Hua University, Taiwan. Dr. Hareesh received his Doctorate from the National Tsing Hua University from the Department of Industrial Engineering and Engineering Management, prior to which he received a Master's in Engineering in Computer Science from AUUP (India), and a Master in Science from the Delhi University (India). His research interests include full stack software systems integration research, analytics on patent data for management information support.

Darrell Mann spent 15 years working at Rolls-Royce and ultimately became Chief Engineer of the company. He left the company in 1996 to help set up a high-technology company spin-out from Imperial College, London, before entering systematic innovation research at the University of Bath. He started using Systematic Innovation in 1992 and teaching Systematic Innovation methods in 1998. Darrell has given workshops to over 15,000 delegates. With over 800 systematic innovation-related papers and articles, plus the best-selling 'Hands-On Systematic Innovation' books, Darrell is now one of the most widely published authors on the innovation subject in the world. He is CEO of Systematic Innovation Ltd, a UK-based innovation company with offices and affiliates in India, Malaysia, China, Denmark, Turkey, Australia, US and Austria. Darrell is now recognised as one of the world's most prolific inventors. He is a Professor at the University of Buckingham in the UK, and Taylor's University in Malaysia.