Application of TRIZ in Software Development

By: Herman Hartmann, Ad Vermeulen and Martine van Beers
j.h.hartmann@philips.com

Introduction

TRIZ has become a powerful tool to solve problems and to create inventive solutions. An increasing number of patents are being generated with the help of TRIZ.

TRIZ is mainly used for mechanical problems and, in its current form, not as much for electrical or software problems. Software is a growing part of a product and is becoming the source of many problems. When TRIZ could be applied to software as well, the applicability of TRIZ could be expanded and software-related problems could be avoided.

At first glance TRIZ doesn't seem to apply to software problems: no atoms, no molecules, no layers to touch; no physical, no chemical effect to apply (see [1]). Yet software problems have been successfully solved with the use of TRIZ. But, according to many, still a lot of work needs to be done (see [2], [3], [4]).

This paper is meant to support the discussions on this subject. On some aspects more experience is needed to obtain a clear picture. Any comment is appreciated greatly.

Inventive Principles

The examples in literature of applying TRIZ to software are that of applying the Inventive Principles (see [6] chapter 8 and 10). They are used to solve an algorithmic problem by defining the ideal situation, analyzing the contradictions and using the principles to develop a better algorithm. (An intelligent algorithm is something like the equivalent of a smart construction in Mechanical Engineering.)

The translation from these Inventive Principles into Software is very difficult to use for many; even for very experienced TRIZ users. The translation made by Kevin Rea ([2] and [3]) is very helpful but only if you are working in a certain application area (in this case that of concurrent programming). A better way is to use the methodology of Genrich Altshuller by analyzing the patents in Software Engineering and develop a completely new set of Inventive Principles and Contradiction Matrix.

The main contribution of TRIZ however lies in breaking through the "mental inertia"(see [6], chapter 3). In most cases it is sufficient to solve the problem by systematically analyzing the contradictions and making them visible using a graphical representation. Micheal Schlueter (see [1]) primarily uses this and the IWB software to solve a problem.

Fast algorithms

The examples that are described in the TRIZ database concern the development of a fast and reliable algorithm using limited resources (such as memory size and processor speed).

The development of successful (patented) algorithms or standards can be commercially very interesting. Examples are: MP3 (for sound processing), MPEG (for Video Processing) and fast search algorithms (Google has become the number 1 search engine, mainly because they own the fastest algorithm). I don't know whether TRIZ is used to develop these algorithms and standards.

Graham Rawlinson (see [5]), however, concludes: "TRIZ is useful, but not often mind blowing in the solutions derived". The reason for this might be that the development of fast algorithms has been a subject of research since the early sixties (when computers where expensive and the resources where very limited). Many of the methods developed then are still valid now. Furthermore the use of graphical representations (a major contributing factor of TRIZ in the field Mechanical Engineering) and formal methods to describe Software is quite common.

Moore's law

The observation made in 1965 by Gordon Moore, co-founder of Intel, that the number of transistors per square inch on integrated circuits had doubled every year since the integrated circuit was invented. Moore predicted that this trend would continue for the foreseeable future. In subsequent years, the pace slowed down a bit, but data density has doubled approximately every 18 months, and this is the current definition of Moore's Law, which Moore himself has blessed. Most experts, including Moore himself, expect Moore's Law to hold for at least another two decades. (From: www.webopedia.com)

The result of Moore's Law is that, because the capabilities of hardware are continuously increasing, there is mostly not a great challenge for Software Engineers with respect to speed and availability of memory. When the Software proves to be too slow, or requires too much memory, simply wait for new Hardware and the problem is solved automatically.

Creating fast algorithms is important when the required Hardware is not yet available for a longer period of time. Video processing, search engines and wireless data communication are typical examples of today. In a few years time these will hardly be issues anymore. The main challenge in software development lies in managing the increasing complexity due to the increasing size of the software and software teams (see section "Software size" and "Architecture Development").

In mechanical engineering there is a continuous challenge in creating new products and dealing with increasing contradictions. In many products the technical limits have (almost) been reached. For 100 years, automobiles have driven on combustion engines;

30 years after the Concorde still there isn't commercial supersonic flight and so forth. In other words it becomes more and more difficult to develop new products, and the increase in functionality becomes smaller. Creativity and finding unorthodox solutions while developing new products is essential.

An example: Suppose a mechanical engineer of a car manufacturer goes to his boss, and tells him: "Boss, I am able to make our engine 5 % more efficient". His boss probably replies: "That's astonishing, take all the time you need and don't forget to write a patent". If a software engineer would go to his boss and tells him "Boss, I can make our software work 5 % faster". His boss probably replies, "That's of no importance, next month we will get the new processor which works twice as fast. Get back to your work, you still have 20 bugs to solve before the end of this week and don't waste any more time".

Software size

As the available memory is increasing 2 times in 18 months, so are the lines of code. Therefore, software products are becoming bigger each year and more people are needed to develop this software. There are trends like open systems; subcontracting; reuse, *et cetera* to overcome this. All this creates situations that are more complex, resulting in project overruns, unreliable software and unsatisfied customers.

The main challenge nowadays is to manage this increasing complexity. Since the early nineties there is a strong emphasis on software processes. Most organizations are using the capability maturity model (see [7]) or other frameworks to improve their software development processes.

Architecture Development

Another way to deal with the increasing complexity is to create architecture of the software. Software architecture provides the technical structure for a project. A good architecture makes the rest of the work easy. A bad architecture makes the rest of the work almost impossible (see [8]).

In creating architectures one has to deal with conflicting demands. The Architecture has to fulfill functional and non-functional requirements. Examples of non-functional requirements are: portability, maintainability, flexibility, extendibility and reusability. These non-functional requirements are also know as *Soft Intents*. In some application areas a *softgoal interdependency graph* (see [9]) is used to visualize the conflicts. Mostly solutions are a "best-fit" between these conflicting demands. Hardly ever all demands are fully met.

TRIZ could be very useful in solving these conflicting demands (Contradictions) in a more satisfying manner. Since architectures are also used in other fields of mechanical engineering we can learn from the application of TRIZ in this field.

Trends of Technological Evolution

Due to the rapidly increasing capabilities of software it is hard to tell what future products will look like. It is very difficult to imagine what is possible in the future and what will be successful. (Who could have thought that SMS, downloadable ring-tones and removable covers, have become major selling factors for mobile phones?)

TRIZ focuses on Technological Evolution and this could be used to identify future possibilities. Combined with commercial trends, this helps in defining successful products in an earlier stage. Further research on this is needed.

Conclusions

Although TRIZ *Inventive Principles* have been applied to solve Software problems, they were only used to create faster algorithms. The additional value of this to the software community is limited due to Moore's law.

The TRIZ *Inventive Principles* could be very useful in solving the contradictions in the creation of Software Architectures and the TRIZ *Trends of Technical Evolution* might be useful in identifying future product. The additional value of this to the software community is much bigger. However, on both subjects no examples are described in literature and thus further research is necessary.

Acknowledgements

The authors would like to thank Micheal Schlueter for help in getting us started and reviewing this article and also Graham Rawlinson and Kevin Rea for their previous work and reviewing this article.

References

[1] "Fast Software by TRIZ", Michael Schlueter, ETRIA World Conference - TRIZ Future 2003
[2] TRIZ and Software – 40 Principle Analogies, Part 1, Kevin Rea, TRIZ-journal 2001
[3] TRIZ and Software – 40 Principle Analogies, Part 2, Kevin Rea, TRIZ-journal 2001
[4] Applying TRIZ to Software Problems, Kevin Rea, TRIZCON2002
[5] TRIZ and Software, Graham Rawlinson, TRIZCON2001
[6] Hands on systematic innovation, Darrell Mann 2002
[7] Managing the Software Process, Watts S. Humphrey, 1989
[8] Software Project Survival Guide, Steve McConnel, 1998
[9] Non-Functional Requirements in Software Engineering, L. Chung, 2000

Herman Hartmann
Software Process Improvement Consultant
Philips CFT
P.O.Box 218

5600 MD Eindhoven
The Netherlands
Tel: +31-402737008
mailto:J.H.Hartmann@philips.com

Ad Vermeulen
Innovation Consultant
Philips CFT
P.O.Box 218
5600 MD Eindhoven
The Netherlands

Martine van Beers
Innovation Consultant
Philips CFT
P.O.Box 218
5600 MD Eindhoven
The Netherlands