TRIZ for Software

Using the Inventive Principles

Kevin C. Rea
kevin@trizforsoftware.com OR kevincrea@yahoo.com

1 Introduction

When I first learned TRIZ, I believed that using TRIZ for the software domain was possible. I saw areas within Lucent Technologies whereby TRIZ could make major improvements, and using the 40 principles and other tools of TRIZ, I was able to produce engineering solutions as well as patent submissions, within Lucent.

Although I resigned from Lucent Technologies to pursue another path, I am still bound by intellectual property restrictions. As such, I can't "publish" these solutions. This may partially explain why we have not yet seen published instances of real software problems being successfully solved using either the 40 Principles or other TRIZ tools as mentioned in [1].

I believe there may be three hard truths concerning this:

1. As what happened to me, people are bound by company restrictions not to publish what may be proprietary in terms of Intellectual Property (IP). This is common for many consultants that must adhere to their client's IP wishes,
2. Some are concerned of another person(s) getting their idea and profiting from it in some way before they can get some intellectual protection (a provisional patent maybe),
3. Covetousness – some would rather take than give, holding onto their solutions without sharing with others and furthering TRIZ. Incidentally, oftentimes the opposite can achieve better results – for example, the success of the open-source initiative.

Could there be more reasons than this? probably, however Genrich Altshuller did not fall into concern number three – he gave it ALL away.

Since my intention was to get TRIZ for software on the front burner in peoples' minds, I set out to observe the 40 principles already in practice- this was the primary goal, to spur minds with the hope that others would join together and build upon, such as in [2], to further realize TRIZ for software. My hope is that those of us pursuing this particular area of TRIZ can work "together" and collectively give back to the TRIZ community.

Although there has been progress in TRIZ for software, there is more work to be done. Possible directions could include:

1. Enhancements in ARIZ to support better problem definition for the software domain (not changing ARIZ per say, but abstracting steps of the algorithm in the context of current software methods as mentioned in [4]),
2. Research of physical contradictions in software design and development (i.e., software specification/design and verification),

3. Analysis of software systems before and after invention in the context of TRIZ,
4. Review of existing software engineering topics using TRIZ [5].

2 Using the Inventive Principles

Following is a concept solution to a real-world problem that uses the matrix and inventive principles of TRIZ.

The problem resides within an application group that is building a Windows 32-bit application (the tool). This tool controls, monitors and diagnoses a major piece of telecommunications equipment. The designers of the telecommunications equipment consist of both hardware, firmware and software engineers. Certain protocols are defined that run on the various pieces of equipment that make up this communication system. The tool must communicate to the various sub-systems of this equipment in the appropriate protocol. The tool is written using Microsoft Visual C++ and the protocol specifics of the tool are defined in corresponding header files (.h) whereby the implementation files (.cpp) reference the protocol definitions when building a specific message object. Much care must be taken when defining these protocols in the header files as one bit shifted or masked the wrong way could cause an unseen system failure and/or potentially damage the equipment. The developer must use design documents of the upstream system designer which explicitly specify the protocol message formats. As a result of this dependency on the systems designer's specification document, and the time needed to change the header files and then rebuild, a delay is introduced that could negatively impact schedule deliverables. The following is a typical scenario in the context of this problem: Revision 1.0 of the Win32 app. has been tested and deployed in the field for some time now; a major bug has recently been detected by field personnel and reported back to the designers. The designers noticed a mistake in their message specification for equipment X that uses protocol Y. A revision of the tool must be built and deployed to the field as soon as possible. Although it is a minor change to the specification document, the developers of the tool must retrieve the appropriate header files from their configuration management system, edit the header files, rebuild the application, test it and redeploy it into the field. Therefore we now have two areas where specification data is being changed, one in the designers specification document, and the resultant header file(s) that define the message specification for the tool. So there is a need to improve the **"waste of time"** needed to get a design specification change implemented in the tool and out to the field, yet at the same time we impair **"accuracy of manufacturing"** by introducing the possibility of human error in interpreting that design change and making the appropriate modifications to the header files. **Table-1** below depicts *one* Technical Contradiction (TC) in the context of this problem and the suggested principles used in solving this problem.

The standard TRIZ matrix was used in this problem. The matrix reveals possible inventive principles that could be used to develop a solution concept. *For this particular TC* I used the standard 40 principles and didn't need to use the software-related principles as in [2] or [3]. I believe this demonstrates that the standard matrix and inventive principles can apply to software problems in their current form just fine; however, in some instances, the software analogies ([2],[3]) may be more helpful to those working in the software domain.

Table-1 Use of Standard Matrix

Technical Contradiction	*Coordinates in the Matrix*	*Suggested Principle*	*Name of the Principle (in order of preference)*
Waste of Time .../ Accuracy of Manufacturing	25x29	24 26 28 18	**Mediator** **Copying** Replacement of Mechanical System Mechanical Vibration

The standard matrix suggests four principles that we can use to help solve this problem. The first two suggestions, "Mediator" and "Copying" trigger a solution concept which is explained and depicted below.

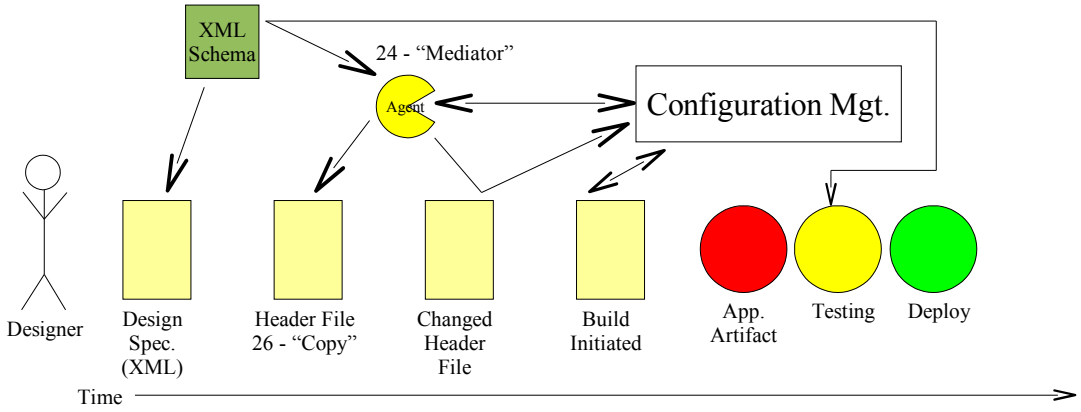
A **"mediator"** will replace the human changing the particular section of code in the appropriate header file. The designer's change in his/her specification document will be **"copied"** in a form suitable for the mediator (**intelligent agent**) to use and make the appropriate change thereby increasing the accuracy of manufacturing. The responsibility is shifted more to the designer in accurately changing the specification document. The specification document will be in some form of XML-based compliant markup language which the intelligent agent will process. Additionally, the agent will also consume other descriptive XML-based schema that describe the particular local configuration management policies:

1. Appropriately retrieve the necessary header files,
2. make the changes,
3. flag for a build process to occur and appropriate testing facilities (manual and/or automated) to test the build,
4. report the results to the designer and product management team.

The responsible authority then gives a go-ahead for deployment to affected consumers (in this case, the remote telecommunications systems) of the change.

Oftentimes you will find that the change will reside in more documents than necessary. Therefore, a possibility exists of errors passed down to the next stage in the process, such as testing. In this solution concept, the testing entity will use the same data that has been "copied" from the original design specification and test against this, not the *developer's* document change. I don't advocate the developer being totally out of the change - they should be flagged as to what the mediator is doing and have a gate in the process to approve or disapprove. **Figure-1** depicts this solution concept.

Figure - 1 – Overall solution concept



3 Summary

The 40 inventive principles (as well as other TRIZ tools) *can* apply to solving software problems. As research and application in this area increases I hope to see this realization of TRIZ for software demonstrated more. In this article, a real-world problem was presented along with the application of using the standard inventive principles and a solution concept formed. In terms of time, this solution concept took 10 minutes to derive, a testimony to Altshuller and those who developed the 40 principles.

4 References

[1] Mann, D.L., TRIZ For Software. *The TRIZ Journal*. Oct. 2004. Internet: http://www.triz-journal.com/archives/2004/10/04.pdf

[2] Fulbright, R., TRIZ and Software Fini. *The TRIZ Journal*. Aug. 2004. Internet: http://www.triz-journal.com/archives/2004/08/02.pdf

[3] Rea, K.C., TRIZ and Software 40 Principles Analogies, Part 2. *The TRIZ Journal.* Nov, 2001. Internet: http://www.triz-journal.com/archives/2001/11/e/

[4] Rea, K.C., Applying TRIZ to Software Problems - Creatively Bridging Academia and Practice in Computing . *TRIZCON 2002 and The TRIZ Journal.* Oct., 2002. Internet: http://www.triz-journal.com/archives/2002/10/c/index.htm

[5] Nakagawa, Toru, Software Engineering and TRIZ (1) Structured Programming Reviewed with TRIZ. *To appear in TRIZCON 2005* - April 2005. Internet: http://www.aitriz.org/

Acknowledgments

Darrell Mann, Ron Fulbright, Toru Nakagawa, Mehdi Akbari, Herman Hartmann, Ad Vermeulen and Martine van Beers, Micheal Schlueter, Graham Rawlinson, and others that have spent and continue to spend time in TRIZ for software.

About The Author

Kevin C. Rea spent over a decade at Lucent Technologies (formerly AT&T) and General Dynamics in various engineering positions to include software engineering and development as well as manufacturing test and measurement. He hopes to see the realization of TRIZ for software - as such, he would really like to work with others to advance this area.

(For those who *need*-to-know, he holds a BS in Electronic Engineering, MS in Computer Science and maybe a PhD someday, Lord-willing)