

Workshop: Communicating Design Patterns with TRIZ

John W. Stamey

Department of Computer Science
Coastal Carolina University
Conway, SC 29568
jwstamey@coastal.edu

Ellen Domb

PQR Group
Upland, CA
ellendomb@trizgroup.com

ABSTRACT

This work will present elements of Genrich Altshuller's Theory of Inventive Problem Solving, also known as TRIZ, and use them to describe the structural patterns found in the Gang of Four's Design Patterns.

Categories and Subject Descriptors

D.2.1 [Software Engineering]:

Requirements/Specifications - Methodologies

General Terms

Design, Documentation

Keywords

TRIZ, Design Patterns

1. INTRODUCTION

Understanding and communication of concepts is always improved when we can describe one framework in terms of another framework. New insight and understanding can be gained from both. This presentation will describe a subset of the twenty-three Design Patterns developed by the Gang of Four [1] as instances of problem solving principles found in Genrich Altshuller's Theory of Inventive Problem-Solving, known as TRIZ. An early observation of the similarity between design patterns and TRIZ is that both systems are sets of heuristics, derived from observations of successful solutions to common problems. Upon further analysis, we find very strong links between the Structural Patterns of the Gang of Four and a specific set of principles from TRIZ.

The 40 Principles were initially developed by analysis of thousands of patents by Altshuller. From this analysis, 39 System Characteristics were determined, such that, when paired with each other in a 39x39 matrix, produced potential situations in which the System Characteristics were in conflict were identified. Altshuller found each pair of conflicting System Characteristics could generally be resolved by applying a core set of forty fundamental principles of TRIZ. [2]

The primary use of the 40 Principles is solving problems that are "technical contradictions" in TRIZ terminology, more commonly called *tradeoffs*. The Contradiction Matrix [3] is constructed with a list of reference to which of the principles are most frequently used to solve which kinds of tradeoff. For example, in a mechanical problem where the strength of a system gets better by adding more material, but the weight gets worse, the matrix tells us that principles 40, 26, 27, and 1 are most frequently used to solve the problem. An excerpt of the 39x39 matrix may be found in the Appendix.

The 40 principles can be used without the matrix, if the analyst can identify the basic assumptions that cause the contradiction to arise, then select one or more principles that solve the problem. For example, in the mechanical example, the strength of the object gets better but the weight gets worse if the strength is improved by adding material. So, solutions that do not involve adding material are desired. The frequently used solutions all predict this:

- Principle 40: Use composite materials (Changes the strength/weight ratio)
- Principle 26: Use an optical copy of some part of the system (Reduces the load, so that the increased strength isn't needed)
- Principle 27: Use cheap disposable copies (Similar)
- Principle 1: Divide the structure into smaller parts (changes the support/surface ratio so that a stronger surface isn't needed)

Computer scientists have long known the forces of time and space (two System Characteristics) as usually being at odds with each other. The general problem is that the as the System Characteristic of time decreases (gets better), the System Characteristic of space increases (gets worse). Conversely, as the System Characteristic of space decreases (gets better), the System Characteristic of time decreases (gets worse).

An interesting example of the time-space tradeoff comes from Jean-Louis Lassez. [4] A problem with databases is that the more information they store, the slower they can become. However, the more information we have, the more likely we are to find the answer to an important question. Identifying “access to more answers and information” as the *Volume of a Static Object* (System Characteristic 8) and identifying “increased the time necessary to find an answer” with *Time Action of a Stationary Object* (System Characteristic 16), we examine the 39x39 contradiction matrix for the solution. The intersection of *Time Action of a Stationary Object* along the horizontal axis (getting worse) and *Volume of a Static Object* along the vertical axis worsening (getting better) leads to three potential solutions from the 40 Principles:

- 35 – Transformation of Properties
- 34 – Rejecting and Regenerating Parts
- 38 – Accelerated Oxidation

Principle 34 includes “After completing its function or becoming useless, an element of an object is rejected (discarded, dissolved, evaporated, etc.) or modified during its work process.” This description contains the essence of the definition of a cache, which is a common solution to the initial problem with databases. Examining the more general issue of more and more time (getting worse) taking to sort through an over-abundance of information (getting better), the TRIZ solution found in Principle 34 nicely describes the idea of a FAQ, or list of Frequently Asked Questions – another example of a cache.

The Gang of Four's 23 Design Patterns have become standard implementation practices in software engineering best practices. This workshop will examine the seven STRUCTURAL DESIGN PATTERNS, at their highest conceptual level, as instances of a number of the 40 Principles of TRIZ.

2. DESIGN PATTERNS AND TRIZ

2.1 Adapter Pattern and the Mediator Principle

The adapter pattern converts the interface of a class into one that an object would expect. In this way, objects can work together because incompatible interfaces are avoided. Suppose we have classes C and D. An object instantiated from class D (d) wishes to use an object instantiated from class C (c). However, the interface to c is not what d expects. The adapter pattern is a way to create a compatible interface for d to c.

TRIZ PRINCIPLE 24 - MEDIATE or negotiate a temporary link between incompatible parties. This is best done when there are conditions that are incompatible or mismatched with related functions, events or conditions.

2.2 Bridge Pattern and the Principle of Extraction

There are many times when we have the notion of an abstract class (such as TVremoteControl) that provides us a way to operate on something with a concrete implementation (such as a TV). The TVremoteControl class has abstract actions (methods) such as on, off, setStation, getNextChannel, etc.). In the real world, we can add, delete or change these TVremoteControl actions. Concrete TV sets (implementations) can change, as well as vary widely across different manufacturers. The bridge pattern class allows us to form a relationship between an abstract class that interacts with a concrete class. We see the "HAS-A" relationship between the abstract class (TVremoteControl) and the concrete class (TV). The reason for this is that there are many different TV implementations. TVremoteControl is in a 1:many relationship with all of the types of TVs (different objects) for which it works.

TRIZ PRINCIPLE 2 - EXTRACTION separates useful or harmful parts or properties of a system. Here, we separate the abstract concept from the physical implementation of a system, so that we can work on each separately.

2.3 Composite and Iterator Pattern and the Principle of Universality

One has several lists to use. Each of the lists can be and most likely are, in different formats. It is useful to create one interface through which items in the different lists can be accessed. A second interface is created to move through lists (in the format provided by the list interface). This interface, through which list items are accessed, is called an *iterator*. The underlying representation (implementation) of the list is of no consequence, the returning of the next list item being of primary importance.

TRIZ PRINCIPLE 6 – UNIVERSALITY promotes uniformity of feature, uniform use of an object for different purposes, or applying the same requirements or features for different objects, situations or actions.

2.4 Decorator Pattern and the Principles of Nesting and/or Flexible Membranes

The decorator pattern dynamically adds functionality to an object. Using the decorator pattern is a way to avoid creating subclasses (child classes) to provide extended functionality. The decorator pattern begins with a superclass of a particular component, C. The actual object we will "decorate"

is a subclass of C, we will call c. The decorator object, d, actually references variables in object c. A subclass of the decorator actually creates the additional functionality provided by the decorator class.

TRIZ PRINCIPLE 7 - NESTING applies where one object fits inside another, or passes through another.

TRIZ PRINCIPLE 30 - FLEXIBLE MEMBRANES OR THIN FILMS prescribes the replacement of traditional constructions with constructions of thin film or flexible membranes. Also, an object may be isolated from its environment with a thin film.

2.5 Facade Pattern and the Principle of Consolidation

Given a collection of interfaces, a client (user) should only have to deal with one unified interface system when trying to access data. The highest level of interfaces is called the facade pattern. It is a direct implementation of the Law of Demeter, which encapsulates the Principle of Least Knowledge ("friends only talk with friends")

TRIZ PRINCIPLE 5 - CONSOLIDATION brings functions or parts of a system into a relationship that creates a new outcome. One example is combining the ability for one player to play multiple formats (DVD, mp3, etc.). For the facade pattern, one interface acts as a gateway to many others.

2.6 Flyweight Pattern and the Principle of Transition into a New Dimension

There are times when you need to have, literally, thousands of instances of an object. Each object is relatively simple, usually calling a few methods. An example of this would be a web-based word processor that has perhaps 2560 locations for characters to be placed. Characters either appear or disappear (two methods). The flyweight pattern prescribes a manager to organize each of the small

objects (configuring the characters into 80 per column and 32 rows per page). The character objects are then responsible for their own management (showing or not showing a letter, number, symbol, or blank).

TRIZ PRINCIPLE 17 - TRANSITION INTO A NEW DIMENSION suggests changing the orientation of a system by adding a dimension. This new dimension can place less strain onto an existing dimension, or it can add a controlling dimension (like the flyweight pattern).

2.7 Proxy Pattern and the Principles of Parameter Change and/or Phase Transition

Like its dictionary namesake, the proxy pattern is used to control access to an object. As an example, there are times when a user might request an image to be displayed. If the size of the image file is quite large, then it might take a while for the image (perhaps a Flash animation) to appear. Good usability practices would suggest that a small "Loading" animated icon show until the larger Flash file is loaded in memory and can play. Enter the proxy pattern to do just that - play the "Loading" icon until the Flash images is ready to display, then display the image.

TRIZ PRINCIPLE 35 - PARAMETER CHANGES suggest changing an object's physical state (e.g. to a gas, liquid, or solid). Change the concentration or consistency. Change the degree of flexibility. Change the temperature. Change any parameter that will improve the situation. (Frequently, identifying the correct parameter is the key to breakthrough problem solving.)

TRIZ PRINCIPLE 36 - PHASE TRANSITION: suggests using phenomena occurring during phase transitions (e.g. volume changes, loss or absorption of heat, etc.) In data analysis, a Fourier transform or other mathematical transformation can be regarded as a phase change.

3. ATTENDEES

All participants of SIGDOC 2006 are welcome to participate in this workshop. Computer scientists and communicators who are interested in using TRIZ to describe existing knowledge frameworks should find the workshop applicable to their research.

4. FUTURE WORK

Research is being done to complete the mapping of the 23 Design Patterns into the 40 Principles of TRIZ. At the same time, we are working to

determine the System Considerations that generate each of the patterns as mapped into the 40 Principles of TRIZ.

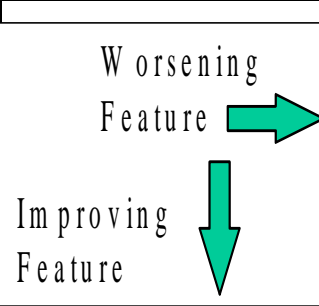
5. ACKNOWLEDGEMENTS

The authors would like to thank Richard Langevin, Executive Director of the Altshuller Institute and Dr. Jean-Louis Lassez for their comments on this workshop.

6. REFERENCES

- [1] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1994.
- [2] Lerner, L. in Altshuller, G. & Clark, D.W. *40 Principles, Extended Edition*. Worcester, MA: Technical Innovation Center, Inc., p. 12.
- [3] Domb, E. Contradiction Matrix and the 40 Principles for Innovative Problem Solving, TRIZ Journal, July 1997, Retrieved from <http://www.trizjournal.com/matrix/>.
- [4] Jean-Louis Lassez, Conway, SC, Personal Communication, August, 2006.

APPENDIX

		Weight of moving object	Weight of stationary object	Length of moving object
		1	2	3
11	Stress or pressure	10, 36, 37, 40	13, 29, 10, 18	35, 10, 36
12	Shape	8, 10, 29, 40	15, 10, 26, 3	29, 34, 5, 4
13	Stability of the object's composition	21, 35, 2, 39	26, 39, 1, 40	13, 15, 1, 28
14	Strength	1, 8, 40, 15	40, 26, 27, 1	1, 15, 8, 35
15	Duration of action of moving object	19, 5, 34, 31	-	2, 19, 9
16	Duration of action by stationary object	-	6, 27, 19, 16	-
17	Temperature	36, 22, 6, 38	22, 35, 32	15, 19, 9