TRIZ and Software Fini

Ron Fulbright
Director, Information Management & Systems
University of South Carolina Spartanburg
Spartanburg, SC 29302
rfulbright@uscs.edu

1 Introduction

In [1] and [2], Rea discusses software engineering analogs to Altshuller's 40 principles of innovation. However, Rea did not list analogs for principles: 29 (pneumatic/hydraulic construction), 31 (porous materials), 36 (phase transition), 37 (thermal expansion), 38 (accelerated oxidation), and 39 (inert environment). This article completes Rea's list by offering software analogs for these six principles and summarizes all 40 software TRIZ principles.

2 Additional Software Analogs

2.1 Hydraulic/Pneumatic Construction (Principle 29)

For mechanical systems, this principle applies to variable-volume parts inflated with liquid or gas. In software, the analog of a container with varying volume is a dynamically allocated data structure. It is often unknown at compile time how much data a program will need to handle. With dynamically allocated data structures, as more elements are required, the data structure's memory footprint expands and then contracts as elements are deleted.

2.2 Porous Materials (Principle 31)

We like to think that our software produces the correct output at all times. However, sometimes this is not always desirable. The suggestion here is that "porous material" be interpreted as intentionally making a software application imperfect. An example is an intelligent tutoring system (ITS). Consider a student learning to play chess from an ITS. The student will become frustrated and less likely to enjoy playing if the computer wins all the time. Also, if the computer plays perfectly all the time, the student will not learn to take advantage of opponents' mistakes—a critical skill in playing chess with human players. Therefore, the ITS needs to be "porous" and intentionally make mistakes to play down to the level of the student. Indeed, the degree to which the ITS does this is can change over time and in concert with monitoring the student's progress via another TRIZ principle, feedback.

2.3 Phase Transition (Principle 36)

Recent research in nonlinear dynamics has exposed an interesting feature of complex adaptive systems. It seems that in a dynamical region just on the controllable side of chaos is a regime called the *emergent regime* in which systems achieve the highest levels of global emergent behavior. Researchers in artificial life have explored this region and coined the phrase "life at the edge of chaos" to describe the sudden onset of complex and sustainable patterns in that region. Others have applied the same idea to natural complex adaptive systems like biology, economics, and markets. Wolfram envisions using the phenomenon as a whole new approach to science.

Researchers liken the sudden shift of a system from controlled behavior to emergent behavior to the change of phase in physical systems—like water changing from solid to liquid as it melts. The degree of randomness in these systems is a key parameter. It seems that given the right amount of randomness, a complex system can be induced to change phase to the emergent regime in which its information processing capability is maximized thereby allowing the system as a whole to achieve more than the sum of its parts. This certainly applies to software systems.

2.4 Thermal Expansion (Principle 37)

Thermal expansion or contraction in physical systems involves a volume change as an object is heated or cooled. A computer's memory space is a combination of active memory (in the CPU) and paged memory (maintained in some nearby storage medium such as cache or virtual memory). The expansion and contraction of this resource, in response to more or less processes requiring varying amounts of memory can be modeled thermodynamically by attaching a metric analogous to *temperature* to the system which would then model the computer's performance at a given time.

2.5 Accelerated Oxidation (Principle 38)

In chemical systems, oxidation is the process of combining with oxygen thereby releasing energy stored in the chemical bonds. This reaction produces heat, a randomized quantity of energy. Obviously, software does not bind with oxygen, but we can abstract the oxidation principle to refer generically to the mixing of something with something else to produce a randomized output. Salted encryption comes to mind as an analog. An encryption algorithm without a random component, "salt", run on some cleartext (say user passwords) will always produce the same encrypted output. A particular password would always be encrypted to the same string on every computer running the unsalted encryption algorithm. If you crack the password once, you can evade security on every other computer employing that algorithm. However, if the encryption algorithm adds a random factor, called "salt", into its calculations, the encrypted text is valid for only the one machine, since, theoretically, all other machines would salt their calculations differently.

2.6 Inert Environment (Principle 39)

An inert environment is one that tends to not react with objects in the environment. A logical connotation is that an inert environment is a benign one. With this interpretation, software test harnesses serve as an analog. In software development, it is often necessary to test the software being developed in a simulated environment providing some, but not all, of the behavior of the actual environment the software will operate in. This artificial construct is generally called a "test harness."

Another analog is benchmark tests, often used to measure hardware and software performance. The environment in which the benchmark is run is carefully controlled to insulate the system from uncontrolled influences while retaining critical characteristics and thus is also an inert environment.

3 Summary of TRIZ Software Analogs

Combining the above analogs to Rea's analogs, and editing for space, results in the condensed summary of TRIZ analogs for software shown in Table 1.

4 References

[1] Rea, K.C., TRIZ and Software 40 Principles Analogies, Part 1. *The TRIZ Journal.* Sep, 2001. Internet: http://www.triz-journal.com/ archives/2001/09/e/index.htm

[2] 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/

1. Segmentation a. Dividing an object into independent parts. b. Make an object modular. c. Increase the degree of fragmentation.	Intelligent Agents C++ templates Confidential Objects
2. Extraction Separate interfering or necessary parts	Extraction of text in images
3. Local Quality a. Change structure from uniform to non-uniform b. Make parts perform different functions	Non-uniform access algorithms Higher levels in a single index tree
4. Asymmetry Change from symmetrical to asymmetrical.	Load balancing, resource allocation
5. Consolidation Make operations contiguous or parallel	Threading, multitasking
6. Universality Perform multiple functions; eliminate parts	Personalization of user interface
7. Nesting Place an object into another	Classes within other classes
8. Counterweight Counter weight with lift	Shared objects in multiple contexts
9. Prior counteraction Preload compensating counter tension	Pre-processing
10. Prior action Fully or partially act before necessary	Pre-compiling
11. Cushion in advance Prepare beforehand to compensate low reliability	Fair scheduling in packet networks
12. Equipotentiality In a potential field, limit position changes	A transparent persistent object store
13. Do it in reverse Invert actions	Transaction rollback
14. Spheroidality Replace linear parts with curved parts	Circular abstract data structures
15. Dynamicity Find an optimal operating condition	Dynamic Linked Libraries (DLLs)
16. Partial or excessive action Use "slightly less" or "slightly more"	Perturbation analysis
17. Transition into new dimension Move in more dimensions	Multi-layered assembly of classes
18. Mechanical Vibration Oscillation	Change the rate of an algorithm
19. Periodic Action Periodic or pulsating actions	Scheduling algorithms
20. Continuity of useful action a. Continue actions b. Eliminate all idle or intermittent actions	Utilizing processor at full load Eliminatel blocking processes

Table 1a – Summary of TRIZ Analogs for Software (1-20)

21. Rushing through Conduct a process at high speed	Burst-mode transmission
22. Convert harm into benefit Eliminate the primary harmful action	Bottleneck DDOS zombies
23. Feedback Introduce feedback	Feedback improving iterations
24. Mediator Use an intermediary	XML-based view generation
25. Self-service Performing auxiliary functions	Periodic auto-update
26. Copying Use simpler and inexpensive copies	Perform a shallow copy
27. Dispose Use multiple inexpensive objects	Rapid prototyping
28. Replacement of Mechanical System Replace mechanical means	Voice recognition/dictation
29. Pneumatic or hydraulic construction Use inflatable gas or liquid parts	Dynamically allocated data structures
30. Flexible films or thin membranes Isolate the object from the environment	Wrapper objects
31. Porous materials Make an object porous	Intelligent tutoring systems
32. Changing the color Change the external view (transparency)	Transparency layers
33. Homogeneity Use same material	Container objects
34. Rejecting and regenerating parts a. Discard portions of an object b. Restore consumable parts	Garbage collection Transaction rollback
35. Transformation properties Change the degree or flexibility	Multi-role objects
36. Phase transition Phase transition phenomenon	Emergent behavior
37. Thermal expansion Use thermal expansion or contraction	System memory
38. Accelerated oxidation Use oxygen-enriched air	Salted encryption
39. Inert Environment Replace normal environment with an inert one	Test harness
40. Composite materials Use composite (multiple) materials	Composite objects

Table 1b – Summary of TRIZ Analogs for Software (21-40)