# 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 Altshuller's 40 principles of innovation and analogs that might be used to apply TRIZ to software engineering. However, Rea did not list analogs for principles 31 (Pourous Materials), 36 (Phase Transition), 37 (Thermal Expansion), 38 (Accelerated Oxidation), and 39 (Inert Environment). This article offers software engineering analogs for these principles and a summary of all 40 principles and software analogs.

# 2 Additional Software Analogs

## 2.1 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. I suggest here that the concept of "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.

## 2.2 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 I call 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, sustained forms 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.

## 2.3 Thermal Expansion (Principle 37)

Thermal expansion or contraction in physical systems involves two key features. First, the volume consumed by a thermally expanded area increases and decreases in the presence of thermal contraction. Second, the degree of randomness increases as an object is heated and decreases as it is 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 thermally. One could attach a metric analogous to *temperature* to the system which would then model the computer's performance at a given time.

In purely software terms, dynamically allocated data structures exhibit analogous expansion and contraction characteristics. As more elements are required, the data structure's memory footprint expands and then contracts as elements are deleted. Rae described this in his article as analog for TRIZ principle 34a (discarding parts).

## 2.4 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 not only produces heat, a randomized quantity of energy, but also degrades the material the oxygen binds with (e.g. rust). 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. Therefore, salted encryption is analogous to a chemical "oxidation" process.

## 2.5 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 following condensed summery of TRIZ analogs for software.

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

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