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, he 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 he 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. Intelligent Agents b. Make an object modular. C-+ templates c. Increase the degree of fragmentation. Confidential Objects 2. Extraction Extraction of text in images 3. Local Quality Extraction of text in images a. Change structure from uniform to non-uniform Non-uniform access algorithms Higher levels in a single index tree Make parts perform different functions 4. Asymmetry Load balancing, resource allocation 5. Consolidation Make operations contiguous or parallel 6. Universality Perform multiple functions; eliminate parts Perform multiple functions; eliminate parts Personalization of user interface 7. Nesting Classes within other classes 8. Counterweight Classes within other classes 9. Prior counteraction Pre-processing 10. Prior action Pre-processing 11. Cushion in advance Transaction rollback 12. Do it in reverse invert actions Transaction rollback 13. Do it in reverse Transaction rollback 14. Spheroidality Replace linear parts with curved parts 15. Dynamicity Find an optim		A (1)	
b. Make an object modular. c. Increase the degree of fragmentation. 2. Extraction Separate interfering or necessary parts 3. Local Quality a. Change structure from uniform to non-uniform b. Make parts perform different functions 4. Asymmetry Change structure from uniform to non-uniform b. Make parts perform different functions 4. Asymmetry Change from symmetrical to asymmetrical. 5. Consolidation Make operations contiguous or parallel 6. Universality Perform multiple functions; eliminate parts Perform multiple functions; eliminate parts Perform multiple functions; eliminate parts Personalization of user interface 7. Nesting Place an object into another 8. Counterweight Counterweight Counterweight Counteraction Preload compensating counter tension Fully or partially act before necessary Pre-processing 10. Prior action Fully or partially act before necessary Prepare beforehand to compensate low reliability 11. Cushion in advance Prepare beforehand to compensate low reliability 12. Equipotential field, limit position changes 13. Do it in reverse Invert actions 14. Spheroidality Replace linear parts with curved parts 15. Dynamicity Find an optimal operating condition 16. Partial or excessive action Use "slightly less" or "slightly more" 17. Transition into ew dimension Move in more dimensions 18. Mechanical Vibration 19. Periodic Action Periodic or pulsating actions 18. Mechanical Vibration a. Continue actions 19. Periodic Action Periodic or pulsating actions 10. Continuit position changes 11. Destination 13. Do it in reverse Invert actions 14. Aspheroidality 15. Dynamicity Find an optimal operating condition 15. Partial or excessive action Use "slightly less" or "slightly more" 15. Dynamic Linked Libraries (DLLs) 16. Partial or excessive action Use "slightly less" or "slightly more" 17. Transition into new dimension Move in more dime	1.	Segmentation	
c. Increase the degree of fragmentation. 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 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 Oniversality Perform multiple functions; eliminate parts Personalization of user interface 7. Nesting Place an object into another Classes within other classes Counterweight			
2. Extraction Separate interfering or necessary parts Extraction of text in images 3. Local Quality a. Change structure from uniform to non-uniform Non-uniform access algorithms 4. Asymmetry Change from symmetrical to asymmetrical. Local 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 Pre-processing Pre-processing 10. Prior action Fully or partially act before necessary Pre-compiling 11. Cushion in advance Pre-processing 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 Faition into new dimension Multi-layered assembly of classes 18. Mechanical Vibration Multi-layered assembly of classes 19. Periodic Action Periodic Action			
Separate interfering or necessary parts Extraction of text in images 3. Local Quality a. Change structure from uniform to non-uniform Non-uniform access algorithms b. Make parts perform different functions Higher levels in a single index tree 4. Asymmetry Load balancing, resource allocation 5. Consolidation Diversality Perform multiple functions; eliminate parts Personalization of user interface 7. Nesting Place an object into another Classes within other classes 8. Counterweight Counterweight with lift Shared objects in multiple contexts 9. Prior counteraction Pre-processing 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 Multi-layered assembly of classes			Confidential Objects
3. Local Quality a. Change structure from uniform to non-uniform Non-uniform access algorithms b. Make parts perform different functions Higher levels in a single index tree 4. Asymmetry 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 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 Partial or excessive action Dynamic Linked Libraries (DLLs) 15. Dynamicity Find an optimal operating condition Dynamic Linked Libraries (DLLs) 15. Dynamicity "sightly less" or "slightly more" Per	2.	Extraction	
3. Local Quality a. Change structure from uniform to non-uniform b. Make parts perform different functions Non-uniform access algorithms 4. Asymmetry Load balancing, resource allocation 5. Consolidation Load balancing, resource allocation 6. Universality Perform multiple functions; eliminate parts Personalization of user interface 7. Nesting Partor multiple functions; eliminate parts Personalization of user interface 7. Nesting Partor multiple functions; eliminate parts Personalization of user interface 9. Prior counterweight Classes within other classes 8. Counterweight Shared objects in multiple contexts 9. Prior action Pre-processing 10. Prior action Pre-processing 11. Cushion in advance Prepare beforehand to compensate low reliability In a potential field, limit position changes A transparent persistent object store 13. Do it in reverse Transaction rollback Invert actions Transaction rollback 14. Spheroidality Dynamicity Find an optimal operating condition Dynamic Linked Libraries (DLLs) 15. Dynamicity "sitightly loss" or "slightly more" 17. Transition intonew dimensi		Separate interfering or necessary parts	Extraction of text in images
a. Change structure from uniform to non-uniform Non-uniform access algorithms b. Make parts perform different functions Higher levels in a single index tree 4. Asymmetry Load balancing, resource allocation 5. Consolidation 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 Cuniter weight with lift Shared objects in multiple contexts 9. Prior counteraction Pre-processing Pre-processing 10. Prior action Fully or partially act before necessary Pre-compiling 11. Cushion in advance Prepreabeforehand to compensate low reliability Fair scheduling in packet networks 12. Equipotential field, limit position changes A transparent persistent object store 13. Do it in reverse 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 Multi-layered assembly of classes 18. Mechanical Vibration Change the rate of an algorithm <th>3.</th> <th>Local Quality</th> <th></th>	3.	Local Quality	
b. Make parts perform different functions 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 Counterweight with lift Shared objects in multiple contexts 9. Prior counteraction Fully or partially act before necessary Pre-processing 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 <th>•.</th> <th></th> <th>Non uniform access algorithms</th>	•.		Non uniform access algorithms
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 In a potential field, limit position changes A transparent persistent object store 13. Do it in reverse Invert actions Transaction rollback Itherations (DLLs) 14. Spheroidality 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. Continue actions Scheduling algorithms			
Change from symmetrical to asymmetrical.Load balancing, resource allocation5. Consolidation Make operations contiguous or parallelThreading, multitasking6. Universality Perform multiple functions; eliminate partsPersonalization of user interface7. Nesting Place an object into anotherClasses within other classes8. Counterweight Counter weight with liftShared objects in multiple contexts9. Prior counteraction Preload compensating counter tensionPre-processing10. Prior action Fully or partially act before necessaryPre-compiling11. Cushion in advance Prepare beforehand to compensate low reliability In a potential field, limit position changesA transparent persistent object store13. Do it in reverse Invert actionsTransaction rollback14. Spheroidality Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continue actionsScheduling algorithms	4		Higher levels in a single index tree
5. Consolidation Threading, multitasking 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 Classes within other classes 9. Prior counteraction Pre-processing 9. Prior action Pre-processing 10. Prior action Pre-compiling 11. Cushion in advance Pre-compiling Prepare beforehand to compensate low reliability Fair scheduling in packet networks 12. Equipotentiality A transparent persistent object store 13. Doi tin reverse Inverse Invert actions Transaction rollback 14. Spheroidality Replace linear parts with curved parts 15. Dynamicity Dynamic Linked Libraries (DLLs) 16. Partial or excessive action Use "slightly less" or "slightly more" 13. Move in more dimensions Multi-layered assembly of classes 16. Partial or excessive action Circular abstract data structures 17. Transition into new dimension Multi-layered assembly of classes 18. Mechanical Vibration </th <th>4.</th> <th></th> <th>Lead below in a second allocation</th>	4.		Lead below in a second allocation
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 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. Continue actions Scheduling algorithms	_		Load balancing, resource allocation
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 algorithm	5.		
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 Faird an optimal operating condition Dynamic Linked Libraries (DLLs) 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 Utilizing processor at full load		Make operations contiguous or parallel	Threading, multitasking
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 Action a. Continuity of useful actions Scheduling algorithms 20. Continuity of useful action a. Continue actions Utilizing processor at full load	6.	Universality	
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 Action a. Continuity of useful actions Scheduling algorithms 20. Continuity of useful action a. Continue actions Utilizing processor at full load		Perform multiple functions; eliminate parts	Personalization of user interface
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 Utilizing processor at full load <th>7</th> <th>· · · ·</th> <th></th>	7	· · · ·	
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 Utilizing processor at full load	••		Classes within other classes
Counter weight with liftShared objects in multiple contexts9. Prior counteraction Preload compensating counter tensionPre-processing10. Prior action Fully or partially act before necessaryPre-compiling11. Cushion in advance Prepare beforehand to compensate low reliabilityPre-compiling12. Equipotentiality In a potential field, limit position changesA transparent persistent object store13. Do it in reverse Invert actionsTransaction rollback14. Spheroidality Replace linear parts with curved partsCircular abstract data structures15. Dynamicity Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load	•		Classes within other classes
9. Prior counteraction Preload compensating counter tension Pre-processing 10. Prior action Fully or partially act before necessary Pre-processing 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 Multi-layered assembly of classes 18. Mechanical Vibration Change the rate of an algorithm 19. Periodic Action Scheduling algorithms 20. Continuity of useful action Scheduling algorithms	8.		
Preload compensating counter tensionPre-processing10. Prior action Fully or partially act before necessaryPre-compiling11. Cushion in advance Prepare beforehand to compensate low reliabilityFair scheduling in packet networks12. Equipotentiality In a potential field, limit position changesA transparent persistent object store13. Do it in reverse Invert actionsTransaction rollback14. Spheroidality Replace linear parts with curved partsCircular abstract data structures15. Dynamicity Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load			Shared objects in multiple contexts
10. Prior action Prepare 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 Multi-layered assembly of classes 18. Mechanical Vibration Change the rate of an algorithm 19. Periodic Action Scheduling algorithms 20. Continuity of useful action Scheduling algorithms	9.	Prior counteraction	
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 Multi-layered assembly of classes 18. Mechanical Vibration Change the rate of an algorithm 19. Periodic Action Scheduling algorithms 20. Continuity of useful action Scheduling algorithms		Preload compensating counter tension	Pre-processing
Fully or partially act before necessaryPre-compiling11. Cushion in advance Prepare beforehand to compensate low reliabilityFair scheduling in packet networks12. Equipotentiality In a potential field, limit position changesA transparent persistent object store13. Do it in reverse Invert actionsTransaction rollback14. Spheroidality Replace linear parts with curved partsCircular abstract data structures15. Dynamicity Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load	10.	Prior action	
11. Cushion in advance Fair scheduling in packet networks 12. Equipotentiality Fair scheduling in packet networks 13. Do it in reverse A transparent persistent object store 13. Do it in reverse Invert actions 14. Spheroidality Transaction rollback 14. Spheroidality Replace linear parts with curved parts 15. Dynamicity Circular abstract data structures 15. Dynamicity Dynamic Linked Libraries (DLLs) 16. Partial or excessive action Use "slightly less" or "slightly more" 17. Transition into new dimension Multi-layered assembly of classes 18. Mechanical Vibration Change the rate of an algorithm 19. Periodic Action Scheduling algorithms 20. Continuity of useful action Scheduling algorithms			Pro compiling
Prepare beforehand to compensate low reliabilityFair scheduling in packet networks12. Equipotentiality In a potential field, limit position changesA transparent persistent object store13. Do it in reverse Invert actionsTransaction rollback14. Spheroidality Replace linear parts with curved partsCircular abstract data structures15. Dynamicity Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load	44		Pie-compiling
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 Multi-layered assembly of classes 18. Mechanical Vibration Change the rate of an algorithm 19. Periodic Action Scheduling algorithms 20. Continuity of useful action Scheduling algorithms	11.		
In a potential field, limit position changesA transparent persistent object store13. Do it in reverse Invert actionsTransaction rollback14. Spheroidality Replace linear parts with curved partsCircular abstract data structures15. Dynamicity Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load		· · ·	Fair scheduling in packet networks
13. Do it in reverse Invert actionsTransaction rollback14. Spheroidality Replace linear parts with curved partsCircular abstract data structures15. Dynamicity Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load	12.		
Invert actionsTransaction rollback14. Spheroidality Replace linear parts with curved partsCircular abstract data structures15. Dynamicity Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load			A transparent persistent object store
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. Continue actions Utilizing processor at full load	13.		
Replace linear parts with curved partsCircular abstract data structures15. Dynamicity Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load		Invert actions	Transaction rollback
Replace linear parts with curved partsCircular abstract data structures15. Dynamicity Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load	14.	Spheroidality	
15. Dynamicity Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load		• •	Circular abstract data structures
Find an optimal operating conditionDynamic Linked Libraries (DLLs)16. Partial or excessive action Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load	15		
16. Partial or excessive action Dynamic Linked Listance (ELLO) 17. Transition into new dimension Perturbation analysis 17. Transition into new dimension Multi-layered assembly of classes 18. Mechanical Vibration Oscillation Oscillation Change the rate of an algorithm 19. Periodic Action Scheduling algorithms 20. Continuity of useful action Utilizing processor at full load	15.		
Use "slightly less" or "slightly more"Perturbation analysis17. Transition into new dimension Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load			Dynamic Linked Libraries (DLLs)
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 Utilizing processor at full load	16.		
Move in more dimensionsMulti-layered assembly of classes18. Mechanical Vibration OscillationChange the rate of an algorithm19. Periodic Action Periodic or pulsating actionsScheduling algorithms20. Continuity of useful action a. Continue actionsUtilizing processor at full load		Use "slightly less" or "slightly more"	Perturbation analysis
18. Mechanical Vibration Oscillation Oscillation Change the rate of an algorithm 19. Periodic Action Periodic or pulsating actions 20. Continuity of useful action Scheduling algorithms a. Continue actions Utilizing processor at full load	17.	Transition into new dimension	
18. Mechanical Vibration Oscillation Oscillation Change the rate of an algorithm 19. Periodic Action Scheduling algorithms Periodic or pulsating actions Scheduling algorithms 20. Continuity of useful action Utilizing processor at full load		Move in more dimensions	Multi-layered assembly of classes
Oscillation Change the rate of an algorithm 19. Periodic Action Periodic or pulsating actions Periodic or pulsating actions Scheduling algorithms 20. Continuity of useful action Utilizing processor at full load	18.	Mechanical Vibration	
19. Periodic Action Periodic or pulsating actions 20. Continuity of useful action Scheduling algorithms a. Continue actions Utilizing processor at full load			Change the rate of an algorithm
Periodic or pulsating actions Scheduling algorithms 20. Continuity of useful action a. Continue actions Utilizing processor at full load 	10		
20. Continuity of useful action utilizing processor at full load a. Continue actions Utilizing processor at full load	13.		Schoduling algorithms
a. Continue actions Utilizing processor at full load			
	20.		
D. Eliminate all idle or intermittent actions Eliminatel blocking processes			
		D. Eliminate all idle or intermittent actions	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		
22. Convert harm into benefit Eliminate the primary harmful actionBottleneck DDOS zombies23. Feedback Introduce feedbackFeedback improving iterations24. Mediator Use an intermediaryXML-based view generation25. Self-service Performing auxiliary functionsPeriodic auto-update26. Copying Use simpler and inexpensive copiesPerform a shallow copy27. Dispose Use multiple inexpensive objectsRapid prototyping28. Replacement of Mechanical System Replace mechanical meansVoice recognition/dictation29. Pneumatic or hydraulic construction Use inflatable gas or liquid partsDynamically allocated data structures30. Flexible films or thin membranes Isolate the object from the environmentWrapper objects		Rurat modo transmission
Eliminate the primary harmful actionBottleneck DDOS zombies23. FeedbackIntroduce feedbackFeedback improving iterations24. MediatorXML-based view generationUse an intermediaryXML-based view generation25. Self-servicePerforming auxiliary functionsPerforming auxiliary functionsPeriodic auto-update26. CopyingPerform a shallow copy27. DisposeRapid prototyping28. Replacement of Mechanical SystemRapid prototyping29. Pneumatic or hydraulic constructionUse inflatable gas or liquid parts30. Flexible films or thin membranesDynamically allocated data structures31. Porous materialsWrapper objects		
23. FeedbackFeedbackIntroduce feedbackFeedback improving iterations24. MediatorXML-based view generationUse an intermediaryXML-based view generation25. Self-servicePeriodic auto-update26. CopyingPeriodic auto-update27. DisposePerform a shallow copy27. DisposeRapid prototypingUse multiple inexpensive objectsRapid prototyping28. Replacement of Mechanical SystemVoice recognition/dictation29. Pneumatic or hydraulic constructionDynamically allocated data structures30. Flexible films or thin membranesWrapper objects31. Porous materialsVoice feedback		Bottleneck DDOS zombies
24. Mediator XML-based view generation 25. Self-service Periodic auto-update 26. Copying Periodic auto-update 27. Dispose Perform a shallow copy 28. Replacement of Mechanical System Rapid prototyping 28. Replace mechanical means Voice recognition/dictation 29. Pneumatic or hydraulic construction Dynamically allocated data structures 30. Flexible films or thin membranes Isolate the object from the environment Wrapper objects Wrapper objects		
24. Mediator XML-based view generation 25. Self-service Periodic auto-update 26. Copying Use simpler and inexpensive copies 27. Dispose Perform a shallow copy 28. Replacement of Mechanical System Rapid prototyping 28. Replace mechanical means Voice recognition/dictation 29. Pneumatic or hydraulic construction Dynamically allocated data structures 30. Flexible films or thin membranes Isolate the object from the environment Wrapper objects Wrapper objects		Feedback improving iterations
Use an intermediaryXML-based view generation25. Self-service Performing auxiliary functionsPeriodic auto-update26. Copying Use simpler and inexpensive copiesPerform a shallow copy27. Dispose Use multiple inexpensive objectsRapid prototyping28. Replacement of Mechanical System Replace mechanical meansVoice recognition/dictation29. Pneumatic or hydraulic construction Use inflatable gas or liquid partsDynamically allocated data structures30. Flexible films or thin membranes Isolate the object from the environmentWrapper objects		
25. Self-service Performing auxiliary functionsPeriodic auto-update26. Copying Use simpler and inexpensive copiesPerform a shallow copy27. Dispose Use multiple inexpensive objectsPerform a shallow copy28. Replacement of Mechanical System Replace mechanical meansVoice recognition/dictation29. Pneumatic or hydraulic construction Use inflatable gas or liquid partsDynamically allocated data structures30. Flexible films or thin membranes Isolate the object from the environmentWrapper objects31. Porous materialsVoice method is the object form the environment		XML-based view generation
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	25. Self-service	
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	Performing auxiliary functions	Periodic auto-update
Use simpler and inexpensive copiesPerform a shallow copy27. Dispose Use multiple inexpensive objectsRapid prototyping28. Replacement of Mechanical System Replace mechanical meansVoice recognition/dictation29. Pneumatic or hydraulic construction Use inflatable gas or liquid partsDynamically allocated data structures30. Flexible films or thin membranes Isolate the object from the environmentWrapper objects31. Porous materialsUse inflatable gas	26. Copying	
27. Dispose Rapid prototyping 28. Replacement of Mechanical System Replace mechanical means Replace mechanical means Voice recognition/dictation 29. Pneumatic or hydraulic construction Dynamically allocated data structures 30. Flexible films or thin membranes Isolate the object from the environment 31. Porous materials Wrapper objects		Perform a shallow copy
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 Use inflatable gas or liquid parts	27. Dispose	
28. Replacement of Mechanical System Voice recognition/dictation Replace mechanical means Voice recognition/dictation 29. Pneumatic or hydraulic construction Dynamically allocated data structures 30. Flexible films or thin membranes Isolate the object from the environment 31. Porous materials Wrapper objects	Use multiple inexpensive objects	Rapid prototyping
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 Mathematical Structures Mathematical Structures	28. Replacement of Mechanical System	
Use inflatable gas or liquid partsDynamically allocated data structures30. Flexible films or thin membranes Isolate the object from the environmentWrapper objects31. Porous materialsWrapper objects		Voice recognition/dictation
30. Flexible films or thin membranes Isolate the object from the environment Wrapper objects 31. Porous materials Wrapper objects		
Isolate the object from the environment Wrapper objects 31. Porous materials Variable		Dynamically allocated data structures
31. Porous materials		
	-	Wrapper objects
IVIAKE AN ODIECT DOTOUS		
		Intelligent tutoring systems
32. Changing the color		
Change the external view (transparency) Transparency layers		Transparency layers
33. Homogeneity		
Use same material Container objects		Container objects
34. Rejecting and regenerating parts		
a. Discard portions of an object Garbage collection		
b. Restore consumable parts Transaction rollback		Transaction rollback
35. Transformation properties		
Change the degree or flexibility Multi-role objects		Multi-role objects
36. Phase transition		
Phase transition phenomenon Emergent behavior	-	Emergent behavior
37. Thermal expansion		
Use thermal expansion or contraction System memory	Use thermal expansion or contraction	System memory
38. Accelerated oxidation		
Use oxygen-enriched air Salted encryption	Use oxygen-enriched air	Salted encryption
39. Inert Environment		
Replace normal environment with an inert one Test harness	39. Inert Environment	
40. Composite materials	39. Inert Environment	one Test harness
Use composite (multiple) materials Composite objects	 39. Inert Environment Replace normal environment with an inert 40. Composite materials 	one Test harness

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