Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 00 (2014) 000–000

www.elsevier.com/locate/procedia

TRIZ FUTURE CONFERENCE 2014 - Global Innovation Convention

List of Inventive Principles for Information Technology and Software

Hartmut Beckmann[a]*

[a]*Giesecke & Devrient GmbH, Prinzregentenstrasse 159, 81677 Munich, Germany*

* Corresponding author.

Abstract

In this paper, a list of 30 Inventive Principles for information technology and software is presented. The list was derived from the original 40 Inventive Principles by using a heuristic transfer method. This transfer method avoids assuming that every original Principle has to result in one Principle for information technology. For doing so, it uses the three characteristics of information technology, objects, data, and algorithms. This also reduces the lateral thinking necessary to apply each Principle. It is advantageous to clarify which objects, data, and algorithms are available in the whole system before applying the Inventive Principles for information technology.

Each given Principle for information technology provides solution models for modifying objects, data, and algorithms. The Principle list given is thought to be used as a checklist not to miss any possible solution. Also, a short abstract and example is given for each Principle to illustrate its primary intention.

The numbering of the Inventive Principles for information technology correlates to the numbering of the original Principles. Nevertheless, in some cases the new Principles are very different from the original ones to conform to information technology in the best way. Also, a proposal for a new Principle is given: “Transmutation”, which suggests converting object, data or algorithm into each other.

Peer-review under responsibility of The International Scientific Committee ETRIA.

Keywords: TRIZ, Inventive Principles, Information Technology, Computer Science

1. Introduction

The Inventive Principles for information technology and software presented in this paper were derived from the original 40 Inventive Principles [1, 2]. In contrary to methods used in the past [3, 4], the derivation method [5] does not rely on the assumption that a one-to-one transfer creates the best results. Furthermore, an additional design goal was to avoid psychological inertia [6] when applying the Principles for information technology. For doing so, the original Inventive Principles were applied to the three characteristics of information technology: objects, data, and algorithms.

- An object is any hardware able to provide or process binary data.
- Data is any information available in binary format.
- Algorithms are any sequences of well-defined steps performed by objects using data.

After that, all ideas generated were regrouped using the original Principles. This caused Principles to be dropped, changed in meaning, or to be split. As a result, 30 Inventive Principles for information technology and one proposal for a new Principle were generated. The complete list is given below.

When using these Principles for information technology, it is best practice to create a detailed collection of all objects, data, and algorithms available inside and outside the system first. This can either be done using a resource analysis like in ARIZ [7] or by creating a model of the whole system first using Function Analysis [8, 9].

Currently, there is no Contradiction Matrix [10] available for these new Principles. In daily work, this is a little drawback only. When TRIZ comes into action, in most cases any other method for solving the problem has failed.

Peer-review under responsibility of The International Scientific Committee ETRIA.

Therefore it is important not to miss any possible solution, so the Principles are used as a checklist.

The numbering used for the Inventive Principles for information technology correlates to the numbering of the original Principles, so the new Principles are more comprehensible.

2. List of Principles

Table 1. Principle #1 “Segmentation”.

Abstract:	Divide up mixed things.
Objects:	Split objects by its functions.
	Split objects into logical parts matching requirements of data and algorithms.
Data:	Split data into self-contained parts.
	Spread data across many objects. This may also be used on segmented data.
Algorithms:	Split algorithms in parts which can be used independently from each other. Spread the parts across many objects.
	Decompose an algorithm into different cases. Handle every case separately.
Example:	When watching YouTube videos in Full HD resolution, video and audio streams are transmitted separately to avoid at least audio stutter if the connection slows down temporarily.

Table 2. Principle #2 “Taking out”.

Abstract:	Keep the necessary only.
Objects:	Reduce the necessary features of an object by using existing features of other objects.
	Replace an object by another one supporting the absolutely necessary features only.
Data:	Remove interfering or unnecessary data as soon as possible.
	Keep the absolutely necessary data only.
	Reduce the redundancy of the data.
	Reduce the information contained in the data.
Algorithms:	Remove unnecessary parts of algorithms.
	Keep the absolutely necessary parts of algorithms only.
	Reduce the precision of the algorithm.
Example:	When patching a big software package, only the parts to be changed are transmitted for reducing data volume and transmission time.

Table 3. Principle #3 “Local quality”.

Abstract:	Match things exactly.
Objects:	Select the object used by evaluating the properties of handled data or executed algorithm.
Data:	Fit the data exactly to the properties of the environment.
Algorithms:	Fit the algorithm exactly to the data processed. Every special property of the data has to be used.
	Use any special feature of an object when executing an algorithm.
Example:	The algorithm for calculating the ‘fast inverse square root’ is using only multiplication and bit-shift operations. See Wikipedia for more information.

Table 4. Principle #4 “Asymmetry”.

Abstract:	Make things more different.
Objects:	Decrease using a feature in an object. For compensation, increase usage of another feature in another object.
Data:	Increase the ratio of amount of data available for different objects.
	Increase the ratio between properties of the data.
Algorithms:	Increase the difference between the algorithms used. Especially increase the difference between the algorithms executed by different objects.
Example:	In cloud gaming, a powerful server creates all audio and video data and transmits it to the gamer's device. So, the gamer's device needs considerably less computing power and energy. As a result, computing power and energy consumption of server and gamer's device are very asymmetric.

Table 5. Principle #5 “Merging”.

Abstract:	Put things together.
Objects:	Merge objects providing similar functions.
Data:	Merge data similar in type, content, or structure.
Algorithms:	Merge multistage algorithms or processes into a single one.
	Enable an algorithm to process multiple datasets at once.
	Make an algorithm interacting with many target objects at once.
Example:	Multipath TCP combines different transmission channels into a single logical one, thus increasing throughput and redundancy especially for wireless connections.

Table 6. Principle #6 “Universality”.

Abstract:	Use one to match all.
Objects:	Introduce multi-purpose objects.
	Make the object's functions more generic.
Data:	Provide data by using a generic format.
	Provide data in different formats for different purposes.
	Provide easy access to the data for other objects.
Algorithms:	Provide generic algorithms usable for multiple purposes.
Example:	Transmission of television, telephone and internet data using one physical medium like DSL, cable, or Wi-Fi. This allows using all services using a single device easily.

Table 7. Principle #7 “Nested doll”.

Abstract:	Put things inside.
Objects:	Put an object inside another object.
	Hide an object inside another object so it is no more accessible from the outside.
	Start a function of an object before another function was completed or received.

Data:	Use hierarchic structures instead of sequences.
	Arrange data using different layers.
Algorithms:	Start executing the next step of an algorithm before the current step is completed.
Example:	Video streaming integrates the steps of downloading and playback, thereby greatly reducing the delay until the video starts.

Table 8. Principle #8 “Anti-weight”.

Abstract:	n/a

Table 9. Principle #9 “Preliminary anti-action”.

Abstract:	Keep trouble out.
Objects:	Ensure an object's function cannot get faulty by a malfunction.
Data:	Ensure that data cannot get harmed, faulty, or wrong.
Algorithms:	Ensure an algorithm works right even in case of malfunctions. The malfunction may also be caused by an external event.
Example:	Solid state drives (SSD) and memory cards use 'wear leveling' for distributing write accesses across all flash pages. This avoids data corruption caused by flash pages written to very often.

Table 10. Principle #10 “Preliminary action”.

Abstract:	Work before work.
Objects:	Perform one or more object's functions as soon as possible.
Data:	Structure data elements in an advantageous order.
	Provide pre-calculated data.
Algorithms:	Perform parts of an algorithm as soon as possible.
Example:	After uploading a video to YouTube or another video-sharing site, the video is re-compressed and stored in different resolutions. This eliminates computing effort when the video is streamed later on.

Table 11. Principle #11 “Beforehand cushioning”.

Abstract:	Keep trouble locked-in.
Objects:	Ensure that an object's malfunction does not bring down other parts of the system.
	Ensure that an excessively used function does not cause harmful side effects.
Data:	Ensure that faulty data or results cannot cause major trouble.
Algorithms:	Ensure that faulty algorithms cannot cause major trouble.
Example:	Solid state drives (SSD) and hard disk drives (HDD) all keep a number of replacement pages/sectors, so faulty ones can be replaced by new ones. Thus errors stay inside the drive and cannot cause further trouble outside.

Table 12. Principle #12 “Equipotentiality”.

Abstract:	Minimize changes.
Objects:	Reduce the difference between input and output data of an object.
	Minimize the changes caused by an object's function in the function target.
Data:	Avoid changes in data content.
	Avoid changes in parameters of data.
Algorithms:	Reduce the number of special cases to be handled.
	Minimize the changes done to data.
Example:	Using homomorphic encryption, it is possible to perform specific computations directly on encrypted data without decrypting the data first. This can be used for secure cloud data processing.

Table 13. Principle #13 “The other way round”.

Abstract:	Use the opposite.
Objects:	Reverse one or more functions of an object.
	Exchange one or more functions between objects.
Data:	Put the opposite information into the data.
	Change the representation of the data's information radically.
Algorithms:	Reverse an algorithm itself.
	Reverse the results of an algorithm.
	Exchange the active and passive part of an algorithm.
Example:	In a peer-to-peer (P2P) network every client downloading data also becomes a server. This is in contrary to the common client-server model.

Table 14. Principle #14 “Spheroidality – curvature”.

Abstract:	n/a

Table 15. Principle #15 “Dynamics”.

Abstract:	Let things change.
Objects:	Enable objects for adding, modifying or removing functions while the object is in use.
	Enable the system to add or remove objects at any time.
Data:	Make static data dynamic.
	Make implicit data explicit, then dynamic.
	Make static data parameters dynamic.
Algorithms:	Enable an algorithm to execute steps in a random order.
	Fit an algorithm to resources changing during runtime.
Example:	In Internet television, the video stream bandwidth is dynamically adapted to the current bandwidth of the user device. By doing so, the video quality may get worse but the video does not stop.

Table 16. Principle #16 “Partial or excessive actions”.

Abstract:	Work more or less.
Objects:	Provide additional functions which can be used optionally. This may happen in special cases.
	Provide minimalist functions only.
Data:	Provide less data or less precise data as necessary. Hand in the missing parts later.
	Provide more data or more precise data as necessary. Drop the additional parts.
Algorithms:	Remove a small part of an algorithm to simplify it greatly.
	Add a function to an algorithm which only requires little additional effort.
	Introduce additional data in an algorithm. By using that data, steps of the algorithm can be simplified or omitted.
Example:	The JPEG image format also defines a progressive mode which provides less precise image data at the beginning of the file. This allows showing a rough preview of the image while being loaded.

Table 17. Principle #17 “Another dimension”.

Abstract:	Use resources from outside.
Objects:	Replace functions of an object by functions executed by supersystem objects.
Data:	Use a reference to the data instead of the data itself.
Algorithms:	Use supersystem components for executing parts of an algorithm.
Example:	Using Apple's 'iWork for iCloud', it is possible to edit documents directly in the web browser. As a result, the external resource iCloud is used for storing and sharing the documents.

Table 18. Principle #18 “Mechanical vibration”.

Abstract:	n/a

Table 19. Principle #19 “Periodic action”.

Abstract:	Wait for the other one.
Objects:	Execute a function when it suits best for the function target.
	Synchronize function carrier and function target.
Data:	Provide data exactly at the right time and place.
Algorithms:	Execute one or more steps of an algorithm when it suits best for the step's target.
Example:	The Transmission Control Protocol (TCP) is part of the Internet protocol suite. Its flow control mechanism causes the sender to wait until the receiver is ready to process further datagrams.

Table 20. Principle #20 “Continuity of useful action”.

Abstract:	Work using parts.
Objects:	Divide big functions into many identical small ones.
	Execute the small functions at maximum speed.
	Execute the triggered functions of the target object using partial function results. Repeat the functions of the target object for each partial result.
Data:	Divide lists of datasets into single elements.
Algorithms:	Divide big parts of an algorithm into many identical small parts. Execute the small parts at maximum speed.
	Execute the next step of an algorithm using partial input data of the current step. Repeat the next step for each new data part available.
Example:	When enabled in the settings, Google search shows intermediate search results after each keyword entered. This means the search is also done using partial data only.

Table 21. Principle #21 “Skipping”.

Abstract:	Hide incomplete things.
Objects:	Ensure that functions of an object always leave the target objects in consistent states.
Data:	Ensure the permanent consistency of any accessible data.
	Ensure intermediate results are not misinterpreted as final results.
	Avoid changing the original data until a result is finalized.
Algorithms:	Ensure each step of an algorithm does not cause inconsistent states.
Example:	When downloading a file using an internet browser, the file has a different name or extension until the download is completed. So, complete and incomplete files cannot get mixed up.

Table 22. Principle #22 “Blessing in disguise”.

Abstract:	Squeeze information out of bad things.
Objects:	Analyze errors or problems to get information about an object's status.
	Use a harmful, insufficient, or excessive function of an object to improve system's reliability.
	Provide precise information on problems or errors a function of an object caused. This information may also be returned to the function carrier object.
	Modify an object so it detects problems and errors in detail.
Data:	Analyze data on inaccuracy and errors.
Algorithms:	Amplify errors when processing faulty data. The result is more incorrect than the input data.
	Reduce the robustness of an algorithm for getting more errors.
	Implement a negative characteristic into an algorithm.
	Provide as much information on errors or problems as possible. Preferably, this information can be processed by another algorithm.
Example:	Mobile network operators keep track of worse connection quality of customer's devices. This information is used to improve the mobile network coverage.

Table 23. Principle #23 “Feedback”.

Abstract:	Close the loop.

Objects:	Modify the functions of an object using results of functions executed in former steps.
	Inform other objects about functions executed. Also inform other objects about results of functions executed.
Data:	Create parts of the new input data from output data.
Algorithms:	Modify the behavior of an algorithm by including former results.
Example:	Some internet search engines come with an autocomplete feature offering additional search keywords. These keywords are taken from other searches executed, so the search engine gives a feedback to all users.

Table 24. Principle #24 “Intermediary”.

Abstract:	Add assistance in between.
Objects:	Absorb harmful functions by using an additional object.
	Improve existing functions by using an additional object.
	Add an object which matches functions to special target properties.
Data:	Insert some kind of redundancy.
Algorithms:	Add an algorithm which handles the details of other algorithms, data, or objects.
	Add an algorithm which handles errors and problems on its own.
Example:	Spam filters are additional components which keep junk mails away.

Table 25. Principle #25 “Self-service”.

Abstract:	Become independent.
Objects:	Change the trigger of an object's function.
	Minimize the number of functions an object receives.
	Remove external objects from the system.
Data:	Remove external dependencies from data.
Algorithms:	Change the trigger starting an algorithm.
	Minimize the input data an algorithm needs.
	Minimize the number of objects an algorithm needs.
Example:	In wireless ad hoc networks, each participating device itself becomes an access point and forwards data of other devices. Thus, no external resource is necessary to build a network.

Table 26. Principle #26 “Copying”.

Abstract:	Introduce many clones.
Objects:	Introduce multiple identical objects into system and supersystem. This also may happen at different places or at different times.
Data:	Keep a copy of existing data for other purposes.
	Provide data at different places.
Algorithms:	Modify an algorithm so it can be executed in parallel and mixed up.
Example:	Google Inc. operates many data centers all over the world for short distance to the users. This reduces traffic acquisition costs and network latency.

Table 27. Principle #27 “Cheap short-living objects”.

Abstract:	Work fast, disappear afterwards.
Objects:	Concentrate execution of an object's function in a very short period. Thus, the object's function only has to be temporarily available.
Data:	Reduce lifetime or validity period of data.
Algorithms:	Concentrate usage of data in a very short period. Thus, the data only has to be temporarily available.
Example:	For each web page requested using Hypertext Transfer Protocol (HTTP), one or more channels are opened retrieve the requested data and closed afterwards. So all channels disappear every time the work is done.

Table 28. Principle #28 “Mechanics substitution”.

Abstract:	Change the connection.
Objects:	Change the kind of connections between objects. This also refers to physical connections.
Data:	Change the coupling of data distributed in the system.
	Change the route of data flows.
Algorithms:	Change the interactions between algorithms. Introduce interactions between algorithms not interacting yet.
Example:	A data diode device modifies a network connection so it becomes unidirectional. This allows data to enter a high security area, but ensures no sensitive data of the high security area can get out.

Table 29. Principle #29 “Pneumatics and hydraulics”.

Abstract:	Make things fuzzy.
Objects:	Use probability, vagueness or many-valued logic for controlling an object's function.
Data:	Introduce probability or vagueness data or use many-valued logic.
	Create additional pseudo-data using statistics.
Algorithms:	Involve probability, vagueness or many-valued logic into an algorithm.
Example:	Using fuzzy logic, it is possible to create control systems and handle problems even if vague facts are available only.

Table 30. Principle #30 “Flexible shells and thin films”.

Abstract:	n/a

Table 31. Principle #31 “Porous materials”.

Abstract:	n/a

Table 32. Principle #32 “Color changes”.

Abstract:	Improve the insight.
Objects:	Modify the functions of an object so they are easier to trace.
	Modify an object so its status is easier to understand.
Data:	Modify data coding and data structures so they become more expressive. Thus, understanding, tracing, and debugging become simpler.
Algorithms:	Modify an algorithm so its steps become easier to trace.
Example:	When watching a YouTube video, the context menu item “Stats for nerds” can be used to get technical detail information on audio and video stream.

Table 33. Principle #33 “Homogeneity”.

Abstract:	Unify different things.
Objects:	Use similar or identical objects in your system.
	Make the object's functions more similar.
	Add a similar function or property to all objects.
Data:	Use identical data structures or coding. Also adjust this to the supersystem.
Algorithms:	Use similar input and output data, data structures or coding. Also adjust input and output to the supersystem.
	Use similar algorithms, preferably adapt the system to algorithms of the supersystem.
Example:	In the emerging Internet of Things, every participating device gets a unique ID and access to the Internet. Thus, very different devices become unified.

Table 34. Principle #34 “Discarding and recovering”.

Abstract:	Do it quick and dirty.
Objects:	Remove the feedback an object receives as a result of a function.
	Ignore problems caused by incoming functions.
Data:	Accept faulty or lost data.
	Retrieve completely new data instead of correcting faulty data.
Algorithms:	Ignore errors occurring during an algorithm.
	Remove all parts of an algorithm checking for and handling errors.
	Replace missing or faulty data by default data.
Example:	The User Datagram Protocol (UDP) is part of the Internet protocol suite. In contrary to the common Transmission Control Protocol (TCP) there is no guarantee that a UDP datagram is delivered. Thus, UDP is used by real-time applications where resending of lost data packages does not make any sense.

Table 35. Principle #35 “Parameter changes”.

Abstract:	n/a

Table 36. Principle #36 “Phase transitions”.

Abstract:	Analyze the changes.
Objects:	Track physical parameters of functions when executing.
	Track parameters of the object itself.
Data:	Add data containing the difference to predecessor data. Also track the difference of parameters.
Algorithms:	Track internal work data of an algorithm.
	Use differences between input and output data.
	Use differences between parameters of data.
Example:	For checking the quality of video compression, the difference between source video and compressed video is calculated for each pixel of each frame.

Table 37. Principle #37 “Thermal expansion”.

Abstract:	n/a

Table 38. Principle #38 “Strong oxidants”.

Abstract:	n/a

Table 39. Principle #39 “Inert atmosphere”.

Abstract:	n/a

Table 40. Principle #40 “Composite materials”.

Abstract:	n/a

Table 41. Proposal for Principle #41 “Transmutation”.

Abstract:	Switch the type.
Objects:	Turn an object into an algorithm or data.
Data:	Turn data into an object or an algorithm.
Algorithms:	Turn an algorithm into an object or data.
Examples:	By using white-box cryptography, the encryption key is hidden in an obfuscated algorithm. Thus, the encryption key is not available as data.
	By using virtualization, a piece of hardware can be replaced by a piece of software.

References

[1] Altshuller, G., 40 Principles: TRIZ Keys to Technical Innovation, Technical Innovation Center, 1997.
[2] Altshuller, G., The Innovation Algorithm, Technical Innovation Center, 2007.
[3] Tillaart, R., TRIZ and Software - 40 Principle Analogies, a sequel., TRIZ Journal, January 2006.

[4] Mann, D., Systematic (Software) Innovation, IFR Press, ISBN 978-1-906769-01-7, 2008.
[5] Beckmann, H., Method for Transferring the 40 Inventive Principles to Information Technology and Software, TRIZ Future Conference, October 2014.
[6] Altshuller, G., Creativity as an Exact Science, CRC Press, 1984, p. 10-15.
[7] Altshuller, G., ARIZ Means Victory. Algorithm for Inventive Problem Solving ARIZ-85-?, Petrozavodsk: Karelia, 1989, p. 11-50.
[8] Kowalick, J., Tutorial: Use of Function Analysis and Pruning, with TRIZ and ARIZ, to solve "impossible - to - solve" Problems, TRIZ Journal, December 1996.
[9] Beckmann, H., Function Analysis integrating Software Applications, TRIZ Future Conference, October 2013.
[10] Fedoseev, U., Altshuller, G., The Contradiction Matrix and its Components: Album (in Russian), Petrazavodsk, 1984