40 Inventive Principles with Computing Examples

Principle 1. Segmentation

A. *Divide an object into independent parts.*

- Divide source code up into modules to be built as separate binary images (Structured Analysis/Structured Design - functional decomposition or Object Oriented Analysis/Object Oriented Design - groups of interacting objects)
- Follow a waterfall product life cycle with independent phases
- Use subroutines libraries of reusable software modules
- Multiple Graphics User Interfaces (GUIs) to access compute engine
- Partition the work into individual assignments with different development teams
- Segment a large computer project into tasks

B. *Make an object easy to disassemble.*

- Mix and match computing services
- Use local variables
- Use of 3rd party computing services
- Use of public domain software
- Modular software development processes
- Modular architecture for control mechanism
- Intermodal execution of control flow

C. *Increase the degree of fragmentation or segmentation.*

- Divide up the software routines into reusable libraries
- Execute small segments of software across internet computers
- Empower entire internet for some functions (ex., cloud computing).
- Push decision making to the lowest level possible.
- Customer requirements suggestion programs
- Large validation customer program
- Multi-source customer validation workshops

Principle 2. Taking out

Separate an interfering part or property from an object, or single out the only necessary part (or property) of an object.

- Eliminate underutilized sections of code
- Shut down unused processing cores
- Single out only the useful information in video (mpeg 4) and audio compression (mp3)
- Just-In-Time data and activity design
- Provide only highest value added services/products through help desk and other contract services

Principle 3. Local quality

A. *Change an object's structure from uniform to non-uniform, change an external environment (or external influence) from uniform to non-uniform.*

- Store data in compressed mode
- Use variable clock rate
- Spend the most time on your highest value code section
- Focus code development efforts on top customers
- Perform high value added activities (value stream mapping) to the maximum achievable level
- Perform low value added activities (value stream mapping) to the minimum acceptable level
- Deliver complementary “lite” versions and sell full-service product versions
- Utilize "coding only days" to eliminate other work place distractions
- Allow for download of custom version instead of delivered generic version

B. *Make each part of an object function in conditions most suitable for its operation.*

- Put all necessary data for computation into readily accessible storage (Cache)
- Allow processor to choose clock speed most suitable for interacting with entities and input/output
- Locate computation engines at the point of need or highest usage

C. *Make each part of an object fulfill a different and useful function.*

- Use Object Oriented Programming with specific functions assigned to each object
- Use sub-system computer controllers for distributed and independent decision making
- Allocated/dedicate centralized computing system resources for various tasks

Principle 4. Asymmetry

A. *Change the shape of an object from symmetrical to asymmetrical.*

- Use unfiltered input and filtered output, conversely use filtered input and unfiltered output
- Optimize code/hardware for most value added activities not for overall capabilities
- Design backups with compression rather than an exact copy of the primary data source
- allocate different amounts of memory (ram, cache, hard, backup, etc.) for different operations/activities
- design multi-core systems with uneven capabilities and assign as appropriate
- Spend more time on validation code than developing it
- Spend more time planning code than developing it

B. *If an object is asymmetrical, change its degree of asymmetry.*

- Change allocation of internal system resources during operations and as required
- Compress data to higher or lower degrees
- Use asymmetrical clock speeds that vary over time or according to task

Principle 5. Merging

A. *Bring closer together (or merge) identical or similar objects, assemble identical or similar parts to perform parallel operations.*

- Use multi-core processing or bi-memories
- Use multi and hyper threading
- Massive parallel computing (mainframe) or server farms
- Personal computer networks
- Products grouped into suites
- Allow compute intensive operations (graphics and processors) to utilize common resources

B. *Make operations contiguous or parallel; bring them together in time.*

- Entire team validates product/code with different perspectives
- Utilize the Internet data for browser merging of cached data
- Prototype while designing (iterative development)
- Constantly build latest versions of products whether or not it is entirely complete
- Build all sections of code in parallel and combine at simultaneous completion
- Run all aspects of computation in parallel and combine at simultaneous completion
- Run back-up continuously and in real time
- Store a web page as separate sections on multiple devices to decrease load time

Principle 6. Universality

A. *Make an object or structure perform multiple functions; eliminate the need for other parts.*

- Design computing system to perform multiple functions (i.e., process control, inventory management, statistical analysis, scheduling, planning, etc.)
- Design single unit for all functions (i.e, memory, I/O, clock, graphics, logic processing, etc.)
- Base functional execution based upon resources and energy available allowing more throughput
- Let customers validate your systems and software
- Merge communication, data, and applications into the Internet cloud eliminating custom clients
- Use multiprocessing operating system to extend your desktop applications onto two screens

Principle 7. "Nested Doll"

A. *Place one object inside another; place each object, in turn, inside the other.*

- Use recursive codes, data and data structures in architecture design
- Use layers of approval to hide complex details of process implementation
- Levels of precision (use nested computational algorithms that stop calculations once precision is sufficient)

B. *Make one part pass through a cavity in the other.*

- Use your browser window to view your document files
- Use a debug port to monitor information flow for errors
- Cross check information or data by viewing the data in different formats - look for patterns

- Utilize short gaps in computation of primary requirements to handle secondary requirements
- Use non-contiguous memory locations for temporary data storage needs

Principle 8. Anti-Weight

A. *To compensate for the weight (downward tendency) of an object, merge it with other objects that provide lift.*

- Use memory caching to merge storage data to improve speed of execution
- Provide software services to make the computing platform more attractive to customers
- Merge email with handheld device to allow for faster response time

B. *To compensate for the weight (downward tendency) of an object, make it interact with the environment (e.g. use global lift forces).*

- Use the applications of the internet instead of using local client resources
- Use 3rd party computer applications and concentrate on the fundamentals of the client application
- Use memory available across internet
- Use computational resources across internet

Principle 9. Preliminary Anti-Action

A. *If it will be necessary to do an action with both harmful and useful effects, this action should be replaced with anti-actions to control harmful effects.*

- Save input data for auto repopulation in case of system failure
- Automatically back up data and machine state in case of future power outage
- Pre-load and pre-arrange
- Use iterative feedback from customer for correcting future product features/flaws

B. *Create beforehand stresses in an object that will oppose known undesirable working stresses later on.*

- Shorten system and coding development cycle to allow for more validation time
- Acquire user input about anticipated, and previous, system shortcomings before system design
- Simulate operating environment for software check out before real computing resources arrive
- Demonstrate accelerated capabilities of system before divulging increase resource requirements
- Train coders in efficiency methods (both administrative and technical) before heavy coding activities

Principle 10. Preliminary Action

A. *Perform, before it is needed, the required change of an object (either fully or partially).*

- Plan project release cycles ahead of time
- Use data gathered for other purposes to help speed subsequent operations (i.e., use zip code from credit card billing information to estimate shipping cost before that information is needed)
- Create a library of common objects for use in coding and system design

B. *Pre-arrange objects such that they can come into action from the most convenient place and without losing time for their delivery.*

- Have system familiarize itself with data storage locations each time the system is started
- Executed handshaking on all interfaces expected to for utilization to speed exchange when needed
- Resolve all IP addresses in the locale of the current browsing
- Make calculations before they are actually needed, at least intensive ones

Principle 11. Beforehand Cushioning

A. *Prepare emergency means beforehand to compensate for the relatively low reliability of an object.*

- Insure all actions can be reversed in case of user error
- Automatic backups of data to a remote location
- Install anti-virus scripts before the virus effects the system
- Allow for automatic need recognition and triggering of timeouts for potential loop hangs
- Plan for regular maintenance updates prior to product release

Principle 12. Equipotentiality

A. *In a potential field, limit position changes (e.g. change operating conditions to eliminate the need to raise or lower objects in a gravity field).*

- Minimize data hierarchy structures to allow for as flat of a data structure as possible
- Understand the cost of storage versus computation power and store computationally "expensive" objects and recomputed large memory objects whenever needed
- Keep high usage data in cache, medium usage data "further" from point if use, and delete low usage data immediately upon completion
- Allow customer to download the latest data at all times

Principle 13. 'The Other Way Round'

A. *Invert the action(s) used to solve the problem (e.g. instead of cooling an object, heat it).*

- Instead of hiding system problems, actively promote/expose the problems to let more ideas solve it
- Code real time as customer require changes
- Stop email overload and organization by storing everything without reading and searching one large flat file
- Customize coding after customer orders it
- Simplify your interface instead of adding features (apple OS)

B. *Make movable parts (or the external environment) fixed, and fixed parts movable.*

- Use shifting solid state memory (bits "rotate" in array)
- Deliver product development platforms (i.e., firmware) at the customer site rather customer ordering
- Code at the customer's site
- Design system for remote access by customers
- Allow firmware/software update in the field based upon customer usage model

- Utilize mobile computational agents that go to the data storage location for interface and computation

C. *Turn the object (or process) 'upside down'.*

- Validate before coding
- Have controls and sensor interface reversed (pull data from sensor instead of having sensor push data, assume a sensor status and then test it, have "sensors" make decisions and control system perform administrative duties
- Store data backwards or mirrored
- Create a system architecture based upon existing low level building blocks
- Use a Last In First Out (LIFO) instead of First In First Out (FIFO) data structure
- Build systems based on customer input

Principle 14. Spheroidality - Curvature

A. *Instead of using rectilinear parts, surfaces, or forms, use curvilinear ones; move from flat surfaces to spherical ones; from parts shaped as a cube (parallelepiped) to ball-shaped structures.*

- Iterative coding development (circular) instead of linear/waterfall development
- Use polar coordinates instead of Cartesian
- Use spherical system design to keep all information and actions as close to the control center as possible
- Eliminate the traditional data hierarchical organization structure and replace with a spoke and wheel organization
- Use circular (ring/queue) data structures instead of linear ones (arrays)

B. *Use rollers, balls, spirals, domes.*

- Use a scroll wheel or ball instead of linear movement
- Have curved user interfaces (input - keyboard, output - monitor)
- 3 dimensional spherical hard drive
- Spherical moving solid state memory

C. *Go from linear to rotary motion, use centrifugal forces.*

- Design web pages to flow in a circular fashion so users can always backtrack easily
- Code in loops only
- Continuously check for security credentials, not just upon log-in

Principle 15. Dynamics

A. *Allow (or design) the characteristics of an object, external environment, or process to change to be optimal or to find an optimal operating condition.*

- Dynamic allocation of memory at run-time vs. static allocation designed into the product
- Change user interface as activities change or as system "learns" user preferences
- Automatic video playing based upon quality of the communication bandwidth
- Dynamic resource allocation between logic processor and graphics engine

B. *Divide an object into parts capable of movement relative to each other.*

- Use Object Oriented programming
- Continuously optimizing data storage (allocation, compression, most accessed data, etc.)
- Use various methods (algorithms) to calculate the same requirement and compare for accuracy

C. If an object (or process) is rigid or inflexible, make it movable or adaptive.

- Change rigid process to an agile module that allow for changing requirements.
- Create dynamic coding that changes its structure or execution order based on computational requirements
- Design transistors that can be reconfigured to function as memory or logic as needed by system

Principle 16. Partial or Excessive Actions

A. *If 100 percent of an objective is hard to achieve using a given solution method then, by using 'slightly less' or 'slightly more' of the same method, the problem may be considerably easier to solve.*

- Use multiple redundant systems and use "voting" between them to determine system reliability
- Where speed and user satisfaction is required, pre-calculate all possible data request so that they are ready when called
- Meet some computing requirements to a minimum level – do not waste time where it is not needed

Principle 17. Another Dimension

A. *To move an object in two- or three-dimensional space.*

- Move data in a two or three dimensional manner and store at the location of next usage.
- Arrange user interface windows in two and three dimensional manner to allow for better tactile relationship to the user
- Use hexadecimal formatting and notations
- Change parameter range (attributes) of input/output to meet needs of user/computational engine/database/etc.

B. *Use a multi-story arrangement of objects instead of a single-story arrangement.*

- Interrelate products or services to form a net or cube of capabilities for the customer
- Use 3rd party computing resources for some requirements
- Hide the interworking of the system (on one level) and show the customer only what they need to see (on another level)

C. *Tilt or re-orient the object, lay it on its side.*

- Allow multiple core processing threads instead of static threads of execution
- Shut down part of the system to increase efficiency of the running parts

D. *Use 'another side' of a given area.*

- Use virtual machine instead of static machines

- Use mirrored memory hard drives
- Quantum Computing

Principle 18. Mechanical vibration

A. *Cause an object to oscillate or vibrate.*

- Use vibration for user feedback (vibrating phone, Wii, computer key board, etc)
- Extend the use of clock cycle to coordinate more activity/functionality

B. *Increase its frequency (even up to the ultrasonic).*

- Shorter/faster development cycles to help incorporate dynamic changes in requirements
- Hurry up and run so you can go to sleep and save power
- Multiple channels for customer feedback (blog, discussion boards, telephone, email, web ex, etc)
- Use excessively fast clock cycles

C. *Use an object's resonant frequency.*

- Time back-up and Maintence with natural lull in organizational quite times
- Estimate filter times by processing time and queue length
- Coordinate processing order with multi-core queue orders so that previous manipulations and calculations can be available for sequenced processing (filters, data tags, etc.)
- Use low frequency sweeping (physical and virtual) searches in data retrieval and storage
- Coordinate system maintenance functions with significant triggering events not standard clock cycles
- Alternate usage of algorithms to vary user experience and effect (gaming, etc.)

D. *Use piezoelectric vibrators instead of mechanical ones* E. *Use combined ultrasonic and electromagnetic field oscillations.*

- Use field driven oscillators to trigger mechanical systems and synchronize mechanical clocks
- Use field driven oscillators to measure relative physical movements between computational systems

Principle 19. Periodic Action

A. *Instead of continuous action, use periodic or pulsating actions.*

- Schedule system events to occur as necessary not on schedule
- Queue up and burst process data during lulls in system usage (short or long cycles)

B. *If an action is already periodic, change the periodic magnitude or frequency.*

- Do quality checks or system operations at irregular intervals
- Reward users for frequent, or infrequent, use of systems
- Provide user feedback/input in real time but do certain internal operations at scheduled times

C. *Use pauses between impulses to perform a different action.*

- Catch up system maintenances activities when resource demands are low
- Product development at one location on one time zone, validation and test at another site and time zone. Allows for 24/7 operations/development

Principle 20. Continuity of Useful Action

A. *Carry on work continuously; make all parts of an object work at full load, all the time.*

- Background multi-tasking
- 24 hours a day by 7 days a week development activities
- Daily or weekly builds of the system for test and development usage
- Incorporate improvement into the process continuously or at the end of each usage cycle
- Use auto completion, spelling checks, and other corrections real time during data entry

B. *Eliminate all idle or intermittent actions or work.*

- Stop processing cores that are not needed
- When system is off, allow for wake up on a communication event
- Use mobile computing during commuting
- Synchronize desk email from handheld smartphone

Principle 21. Skipping

A. *Conduct a process, or certain stages (e.g. destructive, harmful or hazardous operations) at high speed.*

- Perform rapid prototyping of software interfaces with stubs and drivers
- Use an agile development methodology to simplify features and test sooner.
- Design repetitive calculations so that they do not require I/O during the process
- Have data transfers utilize all system resources to minimize activity time
- Disallow all activities except current focus (serial processing)
- Remove up front design and focus on output/user

Principle 22. "Blessing in Disguise" or "Turn Lemons into Lemonade"

A. *Use harmful factors (particularly, harmful effects of the environment or surroundings) to achieve a positive effect.*

- Display advertisements on web sites while waiting for a slow page to load
- Compressing data while taking the extra time to backup will achieve more storage space
- Take advantage of delays in data transfer to perform additional computations
- Perform lengthy calculations in parallel with other user defined activities so that user remains engaged and active
- Utilize information from lengthy data gathering activities to benefit the customer and supplier

B. *Eliminate the primary harmful action by adding it to another harmful action to resolve the problem.*

- Eliminate lengthy back up by backing up only the changes to a computer system
- Allow pre-displaying of instant search results as user types and eliminate misspelling of query
- Computational error reduction through manipulation of additional significant digits
- Use marginal search engine results to generate next level search suggestions (additional time)

C. *Amplify a harmful factor to such a degree that it is no longer harmful.*

- Use random data errors as input to encryption systems to reduce de-encryption success rates
- Utilize vector (IFR indicator?) of analysis "errors" (different from what was expected) as input to new analysis direction
- Use debug information as catalyst for transmission of useful information

Principle 23. Feedback

A. *Introduce feedback (referring back, cross-checking) to improve a process or action.*

- To improve security of a system, introduce feedback to obtain user credentials more often for authentication.
- Change color of light when a software update is available
- Bounce icon on the desktop when an update is complete or user action is required
- Use a busy light when the computer is processing information
- Change the cursor of the mouse when executing tasks and not ready for new input.
- When some calculation or query takes a lot of time display a progress bar.
- Make computations with 2 different, but corresponding, methods and qualify results by comparison – internal feedback
- Prompt users to change input resolution based on initial system output – external feedback
- Statistical Process Control – external feedback to the "machine", internal feedback to the production process
- Customer suggestion programs/serves as input to the requirements gathering phase
- Continuously monitor performance of capacity or performance modeling (theory) in relation to reality (experimental) and adjust input to former

B. *If feedback is already used, change its magnitude or influence*.

- A route planner can display the first results while it tries to optimize the route.
- An approximation algorithm can calculate something to a certain precision and intermediate values are presented on the screen.
- Survey every customer about service and use as live input to Service Direction Processes
- Create an incentive program to award project teams based upon finding and fixing bugs or reducing the complexity of the software program.
- Use Multi-Variant Decision Making to compare unlike scenarios
- Check competing system's accuracy and adjust "your" system accordingly (benchmarking performance results)

Principle 24. 'Intermediary'

A. *Use an intermediary carrier article or intermediary process.*

- Use "standardized" database for use by multiple operational systems
- Use an intermediary segment (buffer) in the process to pass and/or provide data to other parts of the system – file server or web server
- Use a mediator/reconciliation module when allegedly identical input to separate systems is indeed different (external monitoring)
- Use a mediator/reconciliation module when input to allegedly identical systems results in different outputs (internal monitoring)
- Print queue, processing queue

B. *Merge one object temporarily with another (which can be easily removed).*

- Use data compression (image, audio, etc)
- Utilize tagged data packets (from a data base?) as input to specialized analysis algorithms
- ID new data as "new" until qualified or validated
- Use data from multiple data bases for certain analyses
- Linked lists to add, delete or modify data

Principle 25. Self-service

A. *Make an object serve itself by performing auxiliary helpful functions*

- Use excess resources (computational, storage, memory, bandwidth, etc.) to continuously run optimization, security, or error checking processes
- Design the security system to test itself before it allows communication with sources that present a potential threat
- Use results of customer inputs to update internal databases
- Viruses and worms that can copy and spread themselves
- Self repairing memory, memory the deactivates bad sectors, databases that can repair its data
- Timing/clocking processes that synchronize themselves
- Self restoring system or self extracting archive that contains the program needed to extract or restore itself.
- When mechanical systems are replaced by other means also utilize action signal for computing purposes (also see Mechanical Substitution)
- Enable or disable sections of executable software based upon self awareness of its usage (hide features to users)

B. *Use waste (or lost) resources, energy, or substances.*

- Use excess resources (computational, storage, memory, bandwidth, etc.) to support virtualization or an alternative revenue business mode
- Heat employee areas facility with computing equipment exhaust heat
- Use older / out of date computing resources to service lower priority processes
- During low demand periods use excess resources to generate business, provide public services, or support employee needs
- Make base data from monthly accounting process available to other departments to support their decision making

Principle 26. Copying

A. *Instead of an unavailable, expensive, fragile object, use simpler and inexpensive copies.*

- Create a "pointer" (by reference) to data store instead of making a copy that takes up additional memory/storage
- Build an array of logically reconfigurable memory to quickly replicate computational system requirements for certain short duration task – this leaves more resources of primary processor for other uses
- Use existing application components (subroutines and methods) instead of creating entire program from scratch
- Provide partially functional software as a debug/customer service or sales tool

B. Replace an object, or process with optical copies.

- Use data object instead of a data instance
- Use a virtualization strategy instead of a fixed strategy
- Provide results of actual computations of real customer data by way of optical manes only (video monitor) until customer "purchases" data
- Share photos on the internet
- Replace only frames of a video file with the region of the video that has changed for each frame
- Scan documents into PDF files that can be shared
- Sell printable event tickets over the internet

C If optical copies are used, move to IR or UV (Use an appropriate out of the ordinary illumination and viewing situation).

- Use heat trace technology to "print" information onto heat sensitive films
- Use IR or UV "data packets" instead of fiber optics
- Project an optical image of the data using a mirror (Pepper's ghost)

Principle 27. Cheap Short-Living Objects

A. *Replace an expensive object with a multiple of inexpensive objects, compromising certain qualities (such as service life, for instance).*

- Use parallel logic engines instead of expensive processors
- Use a matrix (multiple dimensional as necessary) to provide "calculations" opposed to calculating through formulas
- Use simple logic circuits for initial calculations as starting points for iterative cycles that uses more resource intensive engine.
- Minimize, or eliminate, floating point calculations utilizing integers wherever possible.
- Use initial (low quality) calculations as initial data while more accurate calculations continue
- Provide parts of final data set prior to full calculation is complete
- Use least expensive memory as necessary to application (speed vs. cost)
- Use low cost data storage (tape or disk) to back up expensive solid state storage
- Create copies of the data across virtual cloud based storage devices

Principle 28 Mechanics Substitution

A. *Replace a mechanical means with a sensory (optical, acoustic, taste or smell) means.*

- Use voice recognition for input to a computer or handheld
- Use optical data connections vs. electrical/wired data connections
- Use alternative output systems (thermal, acoustic, smell, etc.)

B. *Use electric, magnetic and electromagnetic fields to interact with the object.*

- Use wireless caching
- Make all computer/system interfaces field activated

C. *Change from static to movable fields, from unstructured fields to those having structure.*

- Change from fixed array/table into a dynamic storage using only what is needed at the time
- Linked list vs. static array of data
- Allow for multiple data storage in a single data store based upon the attributes of that data store

D. *Use fields in conjunction with field-activated (e.g. ferromagnetic) particles.*

- Use GPS with device compass to enable search features and localized data
- Use GPS and compass to align image to location (astronomy app on iPhone)

Principle 29. Pneumatics and Hydraulics (analogy - change the movement of objects)

A. *Use gas and liquid parts of an object instead of solid parts (e.g. inflatable, filled with liquids, air cushion, hydrostatic, hydro-reactive).*

- Uses variables instead of hard coded constants. Initialize these variables at start up with the appropriate initial value.
- Use a dynamic array or a linked list instead of an array with fixed dimensions. Far less growing pains, or buffer overflow errors.
- Use a paging based storage system instead of a large static storage
- Distribute complex data across multiple sites for faster/easier access
- Allow for removal a storage device (hot swap) without disrupting any other storage device

Principle 30. Flexible Shells and Thin Films (analogy - reduce dimensionality)

A. *Use flexible shells and thin films instead of three-dimensional structures*

- The proxy pattern acts as a shield for one or more objects.
- OO Classes with defined (fixed) public interface can change their inner representation.

B. *Isolate the object from the external environment using flexible shells and thin films.*

- Use a Virtual Private Network (VPN) to access corporate storage through public system
- Use an organization within an organization structure
- Create layers of software features enabled as you need them
- Isolate object from the environment – capture exceptions and isolate from system software (exception handling)

Principle 31. Porous Materials (analogous)

A. *Make an object porous or add porous elements (inserts, coatings, etc.).*

- Increase the visibility of internal objects of a software system to allow for visibility for debug in the future (i.e., build in hooks for storing intermediate results)
- Create an operating system environment with placeholders for future expansion (iPhone and the App Store)
- Create a software architecture with prototype interfaces that are empty for future expansion
- Set up processes to absorb information from all sources necessary
- Combine services with other providers to create a complete package to the customer (TV with internet connectivity for future expansion)
- Change the interface of a software class exposing more or less of its internal structure.

B. *If an object is already porous, use the pores to introduce a useful substance or function.*

- Use a system to attract hackers to help identify holes in security (aka, a "honey pot")
- Manage a software project without enough resources by walking around and show visibility of management to help improve morale and productivity.
- Fill holes in organization structure with expanded capabilities (i.e., hire Software Engineer with a marketing background)
- Use holes in organizational structure to be filled by a roving staff member that helps bridge inter organizational gaps and communication flow
- Use a "light weight" or lean organization to support nimbleness and quick reaction times
- Create viewing windows (controlled transparency) into the organization which in turn benefits from being partially open and sharing

Principle 32. Color Changes

A. *Change the color of an object or its external environment.*

- Create a color chart of severity for grading software issues (green, yellow, red)
- Change the display of data based upon the ambient lighting conditions
- Returning search query results with top references highlighted
- Use color changes to show search results that are advertisements
- Use colors to indicate software is not running correctly (red borders on screen, flashing, etc)
- Change the color of the storage media to indicate importance or order for backups
- Indicate software system status or alert level with color

B. *Change the transparency of an object or its external environment.*

- Hide operations from the customer buy showing other useful information while waiting for requested information
- Add copy protection to data to prevent duplication and unintended data corruption
- When email arrives fade in and fade out a summary of the message on the screen
- Flash toolbar when an event occurs (email or instant message arrival)
- Allow a computer operation to track and undo operation and allow for backing up (undo) steps
- Create frequent stand up meetings to discuss software design choices on an ongoing basis.
- Post descriptions of software issues to everyone to allow for group problem solving and improvement (Open Source projects, etc)
- In a file system, provide "user, group, and others" levels of information access
- Sense the ambient lighting to automatically dim the screen
- Change buttons or screen feature to shades of red for nighttime viewing (e.g, for uses in astronomy, etc)

Principle 33. Homogeneity

A. *Make objects interact with a given object of the same material (or material with identical properties).*

- Integrate GUI features of multiple products into one platform product (e.g., social media: Google+ or Facebook)
- Use data partitioning to extract unique data and make global data common
- Use a common object repository (aka 'object library') for all architectural elements
- Use a single programming language instead of multiple programming languages in a software system.
- Reuse software products into a software product line (same base type of program)
- Make objects interact with a different objects of the same architecture/structure.
- Use automatic data back ups to compare and correct corrupt data (voting), example: Apple Time Machine backup of multiple copies of data from different sources.
- Create a standardized software development method across different locations to allow for interchangeable software engineering resources.
- Global software development with same coding standards and methods to allow for a 24 hour, 7 day a week development cycle.
- Have offsite events to find common areas of interest and development opportunities.
- Create a common software architecture library of components and object that have common interfaces for reuse across different software projects and teams.
- Trade software development teams with other organizations
- Create temporary assignments in other departments
- Allow the top performers form different departments work on projects together

Principle 34. Discarding and Recovering

A. *Make portions of an object that have fulfilled their functions go away (discard by dissolving, evaporating, etc.) or modify them directly during operation.*

- Free the memory allocation once the executing software program is done using it.
- Use local variables instead of global variables in order to allow for temporary/local memory allocation

- Use 3rd party service providers for one time or repetitious software maintenance jobs (backup, bug fixes, etc)
- Keep a pool of unused (complex) objects that are frequently created, used and deleted. By keeping a pool of them the 'creation' process is much faster as the objects are fetched from the pool. It also reduces the strain on the garbage collector for deleted objects.

B. *Conversely, restore consumable parts of an object directly in operation.*

- Create and use temporary files for intermediate storage during archive or backup
- Use a linked-list dynamic storage structure vs. an array for operational storage
- Use a software garbage collecting function to free (de-allocate) memory no longer needed.
- Provide mental health periods and rejuvenation services for employees
- Give compensation time for overtime efforts, perks for rejuvenation
- Reenergize employees with celebration of goal orientation events

Principle 35. Parameter Changes

- Merge software applications that have almost identical databases to reduce the amount of redundancy.
- By working with templates an editor can become multipurpose, e.g. loading another syntax
- Highlighter template makes an editor suitable for the other programming language.

A. *Change an object's physical state (e.g. to a gas, liquid, or solid).*

- Merge software applications that have almost identical databases to reduce the amount of redundancy
- Simulate computer operation with an emulator or software simulator
- Release an early/unfinished release of software as “dog food” to the developer and internal users to collect real usage data
- Partner with a lead customer to test/simulate user interface changes.
- Allow for development staff to work alternative schedules and telecommuting.

B. *Change the concentration or consistency.*

- Storage of video files based upon bit level changes in frames instead of entire frames.
- Create smaller product releases more frequently
- Create special versions of products focused on special events or usage models.
- Use experts to develop critical sections of software
- Mix all levels of product development into one meeting (Product Development Team)

C. *Change the degree of flexibility.*

- Change the structure of data storage with redundant and hierarchy (high level secure storage vs. cheap/disposable low level storage)
- Allow for flexible data storage offsite (cloud/virtual based storage)
- Customizable services based on customer needs
- Create a template for documents and software routines that can be reused
- Dynamic software support hours of operation based on requirements
- Allow for flexible software engineering staff working hours
- Allow for relocation/geographical change of staff (sabbaticals, overseas assignements, etc)

D. Change the temperature.

- Create a software heat map showing where the problem areas are in the code
- Create a prioritized list of hot features for fixing or adding features to software systems.
- Create a programming contest to allow for creative expansion of ideas

Principle 36. Phase Transitions

A. *Use phenomena occurring during phase transitions.*

- Use skipping when fast forwarding video data
- Create a bottoms up approach to solving software problems instead of tops down
- When fixing a software problem, add additional features to the code base.
- Use a new or different software development methodologies to handle phase transitions
- Use team building/team event to regroup after major software milestones to recharge the team.
- Release software early with expectations of continuous improvement/customization.

Principle 37. Thermal Expansion

A. *Use thermal expansion (or contraction) of materials.*

- Use background processing capability to perform low priority tasks (continuous backup, spell checking, etc)
- Utilize eager volunteers/interns for project work opposed to unexcited recruits
- Take advantage of popular markets to introduce new features and services – use search to add social media, use social media to add hangouts, use email to introduce applications.
- Use data compression decompression to increase decrease size of data. Can also be seen as change of precision
- When data is compressed (thermal contracted) it can be send faster over a network, or it takes up less storage space. One has to heat it up (decompress) to make it useful/executable again.
- Take advantage of hot/fast moving market conditions to adjust software development life cycles and software development team dynamics
- Use dynamic I/O buffers and cache sizes so they can contain more or less information (more less energy) and allow them to optimally fit the available memory.

B. *If thermal expansion is being used, use multiple materials with different coefficients of thermal expansion.*

- Take advantage of peer-to-peer networking to increase the bandwidth over a network with all client/servers sharing with all other resources across the network
- As a users bandwidth and interaction with the software increases, increase the utilization of the software resources (gmail and dynamic storage space based upon actual usage)
- Use variety of experienced and junior skills to provide a balance of work for a demanding project.
- Mix a variety of personality types and use diversity to your advantage when using an expansion of work.
- Create work teams from individuals with different attributes (skills sets, energy levels, areas of interest)

Principle 38. Strong Oxidants ('Boosted Interactions') - analogous

- Write data on disk in the same order that the processor expects it, e.g. high endian versus little endian integers. (aka Big Endian vs. Little Endian)
- Use binary formats to increase efficiency for processing, storage and network traffic.

A. *Replace common air with oxygen-enriched air* (enriched atmosphere) - analogous

- Use stress testing to test software modules
- Take advantage of boundary condition testing to test software modules
- Team up with a new software organization (internal or external) for development projects
- Provide performance incentives to software development staff
- Use crowd sourcing methods to develop software features
- Write critical sections of code in assembly vs. high level language
- Use "inline" assembly code for faster execution

B. *Replace enriched air with pure oxygen (highly enriched atmosphere).*

- Inject new internal software with dog-food production software (aka, new untested features)
- Tie employee pay levels directly to company performance
- Create an atmosphere of infinite resources to solve problems
- Use debuggers and optimization tools to streamline software files
- Utilize retreats or "off-site" planning sessions – highly focused environment
- Remove all extracurricular job expectations form key personnel's responsibilities
- Move parts of the coding development operation directly into the area it services

C. Expose air or oxygen to ionizing radiation, D. *Use ionized oxygen,* E. *Replace ionized oxygen with ozone (atmosphere enriched by 'unstable' elements)* .

- Add new changes to existing software via usability studies to gauge viability
- Allow for software innovation experiments with an unconstrained research group
- Sequester development teams offsite and use systematic innovation methods to solve difficult problems ☺
- Hire consultants to provide new experiences and perspectives to the software organization.
- Accelerate bonuses with accelerated performance based upon meritocracy
- Utilize out of order execution and instruction pipelining
- Embrace the hacker's way: better done now then perfect
- Balance high risk software features with stable, low risk features (legacy)
- Embrace agility and velocity in the development environment

Principle 39. Inert Atmosphere

A. *Replace a normal environment with an inert one.*

- Use stubs and drivers (dummy interfaces and empty procedures/methods) as placeholders for architecture components
- Use dummy placeholder objects in an object oriented system
- Create a shell environment for simulated execution
- Create a completely flat software development organization with outside leadership
- Install a dummy PC on a network to detect early instances of computer viruses or worms (aka Honey Pot)

B. *Add neutral parts, or inert additives to an object.*

- Add a Virtual Private Network (VPN) to a computer system to isolate resources from rest of the network
- Create a two-in-the-box software leadership organization to train replacement and add redundancy
- Add exception handling code section to trap faults that can cause harm
- Create test variable to track software execution during debugging
- Use an emulator to simulate software execution with breakpoints
- Add a buffer to email systems to delay sending email in order to reduce impulsive email
- Add advertisements to the display while the user waits for results

Principle 40. Composite Structures

A. *Change from uniform to composite (multiple) structures. (Awareness and utilization of combinations of different skills and capabilities.)*

- Use object inheritance to create classes only as needed
- Combine older architectures with newer architectures (combined structured analysis/design with Object Oriented Analysis/Design)
- Use composite data structures (use records vs. arrays) for storage
- Use a spiral software development model instead of structured waterfall
- Use agile methodology on top of a structured waterfall methodology
- Hire software developers with hardware design skills
- Use a product development team approach to software development: the entire team (business, test, development, management, finance, legal, marketing, etc) is equally important to delivery
- Use one software source code for multiple platform targets