Principles of Language Design

1. ***Abstraction:*** Avoid requiring something to be stated more than once; factor out the recurring pattern.
2. ***Automation:*** Automate mechanical, tedious, or error-prone activities.
3. ***Defense-in-Depth:*** Have a series of defenses so that if an error isn't caught by one, it will probably be caught by another.
4. ***Information Hiding:*** The language should permit modules designed so that (1) the user has all the information needed to use the module correctly, and nothing more; (2) the implementor has all of the information needed to implement the module correctly, and nothing more.
5. ***Labeling:*** Avoid arbitrary sequences more than a few items long; do not require the user to know the absolute position of an item in a list. Instead, associate a meaningful label with each item and allow the items to occur in any order.
6. ***Localized Cost:*** Users should only pay for what they use; avoid distributed costs.
7. ***Manifest Interface:*** All interfaces should be apparent (manifest) in the syntax.
8. ***Orthogonality:*** Independent functions should be controlled by independent mechanisms.
9. ***Portability:*** Avoid features or facilities that are dependent on a particular machine or a small class of machines.
10. ***Preservation of Information:*** The language should allow the representation of information that the user might know and that the compiler might need.
11. ***Regularity:*** Regular rules, without exceptions, are easier to learn, use, describe, and implement.
12. ***Security:*** No program that violates the definition of the language, or its own intended structure, should escape detection.
13. ***Simplicity:*** A language should be as simple as possible. There should be a minimum number of concepts with simple rules for their combination.
14. ***Structure:*** The static structure of the program should correspond in a simple way with the dynamic structure of the corresponding computations.
15. ***Syntactic Consistency:*** Similar things should look similar; different things different.
16. ***Zero-One-Infinity:*** The only reasonable numbers of zero, one, and infinity.

MacLennan, B. J., *Principles of Programming Languages: Design, Evaluation, and Implementation*, Holt, Rinehart and Winston, 1983.

Principles of Software Engineering Environment Design

1. ***Methodology Support:*** A software engineering environment should be designed to support a specific engineering methodology.
2. ***Design Documentation:*** A software engineering environment should support a design documentation system that is (1) complete, (2) capable of representing appropriate levels and types of abstractions with fine granularity, (3) universal, and (4) economical.
3. ***Enforcement/Aid:*** A software engineering environment should enforce technical correctness and conformance with management policies while aiding the user in maintaining these standards.
4. ***Consistency Principle:*** "Permanent" alteration of one view of a design should not be "accepted" by the environment until all related views are made to be consistent with the change.
5. ***Structure Manipulation:*** The user of a software engineering environment should be able to create, reference, locate, alter, and delete structures defined within the environment. He should also be able to display meaningful representations of them within the context of any applicable type or level of abstraction.
6. ***Static Analysis:*** A software engineering environment should (1) allow the user to make assertions about the static structure of a module, program, or system of programs and then report back to the user which assertions are not valid and why, and (2) allow the user to request certain information about the static structure and then report this information back to the user in a lucid format.
7. ***Dynamic Analysis:*** A software engineering environment should (1) allow the user to make assertions about the dynamic structure of a module, program, or system of programs and then report back to the user which assertions are not valid and why, and (2) allow the user to request certain information about the dynamic structure and then report this information back to the user in a lucid format.
8. ***Data Collection:*** A software engineering environment should provide for the unobtrusive collection of software management data.
9. ***Audit Trail:*** A software engineering environment should maintain audit trails showing the relationships between requirements, specifications, designs, implementations, maintenance and enhancements, and the resources expended in these activities. Further, such audit trails should be navigable in both directions from any point.
10. ***Information Organization:*** The information gleaned from the various documentation and data collection efforts should be organized for easy reference and maintenance.
11. ***Management Control:*** A software engineering environment must be controllable by the management of the organization it serves.

12. ***Organizational Structure:*** A software engineering environment should be partitionable along the structural lines of the organization so that individuals and groups may have the necessary levels of privacy and isolation from other individuals and groups, and so the communications among them may be reasonably controlled by forcing them through established interfaces. However, it should also be possible to allow information to flow across organizational boundaries, when needed, through specially designated and controlled interfaces.
13. ***Interface Language Design:*** The language of interaction between a user and an interactive application should be designed according to the principles of good programming language design.
14. ***Multi-level Help:*** For any interactive tool, there should be a hierarchy of "help" levels ranging from no prompts at all to on-line instruction. Furthermore, within an environment, the user should be able to set the "help level" on a tool-by-tool basis (fine granularity of help levels).
15. ***Configurability:*** The user should be able to configure his interface to the environment (tool) within the constraints mandated by higher management. This capability should display a fine granularity.
16. ***Meaningful Response:*** The appearance of the display and the text of messages in response to user actions must be appropriate to those actions.
17. ***Rapid Response:*** Simple and frequently used functions should have an immediate (in terms of human reflexes) effect upon the display.
18. ***Status Reporting:*** Lengthy functions should periodically update the display to assure the user that progress is being made in carrying out the requested function.
19. ***Display Inertia:*** The display should change by the least amount possible in response to a user action. The display should not, however, violate the Meaningful Response principle.
20. ***Extensibility:*** An environment should be extensible in the sense that it must be possible to add tools at any time and at any level which enjoy the same level of control and integration as the original tools.
21. ***Reversibility:*** Any action taken by a user must be reversible for some period of time or number of subsequent actions.

Frost, J. R., *Principles of Software Engineering Environment Design*, master's thesis, Naval Postgraduate School, Monterey, CA, 1984.