Software design

3.1 Software design is the process by which an agent creates a specification of a software artifact, intended to accomplish goals, using a set of primitive components and subject to constraints. Software design may refer to either "all the activities involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems" or "the activity following requirements specification and before programming, as ... [in] a stylized software engineering process."

Software design usually involves problem solving and planning a software solution. This includes both low-level component and algorithm design and high-level, architecture design.

3.2 Overview

Software design is the process of implementing software solutions to one or more set of problems. One of the important parts of software design is the software requirements analysis (SRA). It is a part of the software development process that lists specifications used in software engineering. If the software is "semi-automated" or user centered, software design may involve user experience design yielding a story board to help determine those specifications. If the software is completely automated (meaning no user or user interface), a software design may be as simple as a flow chart or text describing a planned sequence of events. There are also semi-standard methods like Unified Modeling Language and Fundamental modeling concepts. In either case, some documentation of the plan is usually the product of the design. Furthermore, a software design may be platform-independent or platform-specific, depending on the availability of the technology used for the design.

Software design can be considered as creating a solution to a problem in hand with available capabilities. The main difference between Software analysis and design is that the output of a software analysis consist of smaller problems to solve. Also, the analysis should not be very different even if it is designed by different team members or groups. The design focuses on the capabilities, and there can be multiple designs for the same problem depending on the environment that solution will be hosted. They can be operations systems, webpages, mobile or even the new cloud computing paradigm. Sometimes the design depends on the environment that it was developed for, whether it is created from reliable frameworks or implemented with suitable design patterns.

When designing software, two important factors to consider are its security and usability.

Software Design

Software design is both a process and a model. The design process is a sequence of steps that enable the designer to describe all aspects of the software to be built. It is important to note, however, that the design process is not simply a cookbook. Creative skill, past experience, a sense of what makes "good" software, and an overall commitment to quality are critical success factors for a competent design. The design model is the equivalent of an architect's plans for a house. It begins by representing the totality of the thing to be built (e.g., a three-dimensional rendering of the house) and slowly refines the thing to provide guidance for constructing each detail (e.g., the plumbing layout). Similarly, the design model that is created for software provides a variety of different views of the computer software. Basic design principles enable the software engineer to navigate the design process. Davis [DAV95] suggests a set of principles for software design, which have been adapted and extended in the following list:

- **The design process should not suffer from "tunnel vision."** A good designer should consider alternative approaches, judging each based on the requirements of the problem, the resources available to do the job.
- **The design should be traceable to the analysis model.** Because a single element of the design model often traces to multiple requirements, it is necessary to have a means for tracking how requirements have been satisfied by the design model.
- **The design should not reinvent the wheel.** Systems are constructed using a set of design patterns, many of which have likely been encountered before. These patterns should always be chosen as an alternative to reinvention. Time is short and resources are limited! Design time should be invested in representing truly new ideas and integrating those patterns that already exist.
- **The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world.** That is, the structure of the software design should (whenever possible) mimic the structure of the problem domain.
- **The design should exhibit uniformity and integration.** A design is uniform if it appears that one person developed the entire thing. Rules of style and format should be defined for a design team before design work begins. A design is integrated if care is taken in defining interfaces between design components.

- **The design should be structured to accommodate change.** The design concepts discussed in the next section enable a design to achieve this principle.
- **The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.** Well- designed software should never “bomb.” It should be designed to accommodate unusual circumstances, and if it must terminate processing, do so in a graceful manner.
- **Design is not coding, coding is not design.** Even when detailed procedural designs are created for program components, the level of abstraction of the design model is higher than source code. The only design decisions made at the coding level address the small implementation details that enable the procedural design to be coded.
- **The design should be assessed for quality as it is being created, not after the fact.** A variety of design concepts and design measures are available to assist the designer in assessing quality.
- **The design should be reviewed to minimize conceptual (semantic) errors.** There is sometimes a tendency to focus on minutiae when the design is reviewed, missing the forest for the trees. A design team should ensure that major conceptual elements of the design (omissions, ambiguity, inconsistency) have been addressed before worrying about the syntax of the design model.

Design Concepts

The design concepts provide the software designer with a foundation from which more sophisticated methods can be applied. A set of fundamental design concepts has evolved. They are:

1. <u>Abstraction</u> - Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose.
2. <u>Refinement</u> - It is the process of elaboration. A hierarchy is developed by decomposing a macroscopic statement of function in a step-wise fashion until programming language statements are reached. In each step, one or several instructions of a given program are decomposed into more detailed instructions. Abstraction and Refinement are complementary concepts.
3. <u>Modularity</u> - Software architecture is divided into components called modules.

4. Software Architecture - It refers to the overall structure of the software and the ways in which that structure provides conceptual integrity for a system. A good software architecture will yield a good return on investment with respect to the desired outcome of the project, e.g. in terms of performance, quality, schedule and cost.
5. Control Hierarchy - A program structure that represents the organization of a program component and implies a hierarchy of control.
6. Structural Partitioning - The program structure can be divided both horizontally and vertically. Horizontal partitions define separate branches of modular hierarchy for each major program function. Vertical partitioning suggests that control and work should be distributed top down in the program structure.
7. Data Structure - It is a representation of the logical relationship among individual elements of data.
8. Software Procedure - It focuses on the processing of each modules individually
9. Information Hiding - Modules should be specified and designed so that information contained within a module is inaccessible to other modules that have no need for such information

Design considerations

There are many aspects to consider in the design of a piece of software. The importance of each should reflect the goals the software is trying to achieve. Some of these aspects are:

- **Compatibility** - The software is able to operate with other products that are designed for interoperability with another product. For example, a piece of software may be backward-compatible with an older version of itself.
- **Extensibility** - New capabilities can be added to the software without major changes to the underlying architecture.
- **Fault-tolerance** - The software is resistant to and able to recover from component failure.
- **Maintainability** - A measure of how easily bug fixes or functional modifications can be accomplished. High maintainability can be the product of modularity and extensibility.
- **Modularity** - the resulting software comprises well defined, independent components. That leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a

desired software system. This allows division of work in a software development project.

- **Reliability** - The software is able to perform a required function under stated conditions for a specified period of time.
- **Reusability** - the software is able to add further features and modification with slight or no modification.
- **Robustness** - The software is able to operate under stress or tolerate unpredictable or invalid input. For example, it can be designed with a resilience to low memory conditions.
- **Security** - The software is able to withstand hostile acts and influences.
- **Usability** - The software user interface must be usable for its target user/audience. Default values for the parameters must be chosen so that they are a good choice for the majority of the users.[3]
- **Performance** - The software performs its tasks within a user-acceptable time. The software does not consume too much memory.
- **Portability** - The usability of the same software in different environments.
- **Scalability** - The software adapts well to increasing data or number of users.

Modeling language

A modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure. A modeling language can be graphical or textual. Examples of graphical modeling languages for software design are:

- Business Process Modeling Notation (BPMN) is an example of a Process Modeling language.
- EXPRESS and EXPRESS-G (ISO 10303-11) is an international standard general-purpose data modeling language.
- Extended Enterprise Modeling Language (EEML) is commonly used for business process modeling across a number of layers.
- Flowchart is a schematic representation of an algorithm or a step-wise process,
- Fundamental Modeling Concepts (FMC) modeling language for software-intensive systems.
- IDEF is a family of modeling languages, the most notable of which include IDEF0 for functional modeling, IDEF1X for information modeling, and IDEF5 for modeling ontologies.

- Jackson Structured Programming (JSP) is a method for structured programming based on correspondences between data stream structure and program structure
- LePUS3 is an object-oriented visual Design Description Language and a formal specification language that is suitable primarily for modelling large object-oriented (Java, C++, C#) programs and design patterns.
- Unified Modeling Language (UML) is a general modeling language to describe software both structurally and behaviorally. It has a graphical notation and allows for extension with a Profile (UML).
- Alloy (specification language) is a general purpose specification language for expressing complex structural constraints and behavior in a software system. It provides a concise language based on first-order relational logic.
- Systems Modeling Language (SysML) is a new general-purpose modeling language for systems engineering.

Design patterns

A software designer or architect may identify a design problem which has been solved by others before. A template or pattern describing a solution to a common problem is known as a design pattern. The reuse of such patterns can speed up the software development process, having been tested and proven in the past.

Technique

The difficulty of using the term "design" in relation to software is that in some sense, the source code of a program *is* the design for the program that it produces. To the extent that this is true, "software design" refers to the design of the design. Edsger W. Dijkstra referred to this layering of semantic levels as the "radical novelty" of computer programming, and Donald Knuth used his experience writing TeX to describe the futility of attempting to design a program prior to implementing it:

TEX would have been a complete failure if I had merely specified it and not participated fully in its initial implementation. The process of implementation constantly led me to unanticipated questions and to new insights about how the original specifications could be improved.

Usage

Software design documentation may be reviewed or presented to allow constraints, specifications and even requirements to be adjusted prior to computer programming. Redesign may occur after review of a programmed simulation or prototype. It is possible to design software in the process of programming, without a plan or requirement analysis, but for more complex projects this would not be considered feasible. A separate design prior to programming allows for multidisciplinary designers and Subject Matter Experts (SMEs) to collaborate with highly skilled programmers for software that is both useful and technically sound. Making of robots is also a huge use of software design