The Nature of Software

The nature of the software medium has many consequences for systems engineering (SE) of software-intensive systems. Fred Brooks has famously observed that four properties of software, taken together, differentiate it from other kinds of engineering artifacts (Brooks 1995). These four properties are:

1. complexity,
2. conformity,
3. changeability,
4. invisibility.

Brooks states:

> *Software entities are more complex for their size than perhaps any other human construct because no two parts are alike (at least above the statement level). If they are, we make the two similar parts into a subroutine — open or closed. In this respect, software systems differ profoundly from computers, buildings, or automobiles, where repeated elements abound.* (Brooks 1995, p 82)

□

Contents

Complexity

The complexity of software arises from the large number of unique interacting parts in a software system. The parts are unique because they are encapsulated as functions, subroutines, or objects, and invoked as needed rather than being replicated. Software parts have several different kinds of interactions, including serial and concurrent invocations, state transitions, data couplings, and interfaces to databases and external systems.

Depiction of a software entity often requires several different design representations to portray the numerous static structures, dynamic couplings, and modes of interaction that exist in computer software. Complexity within the parts and in the connections among parts requires that changes undergo substantial design rigor and regression testing. Software provides functionality for components that are embedded, distributed and data centric. Software can implement simple control loops as well as complex algorithms and heuristics.

Complexity can hide defects that may not be discovered easily, thus requiring significant additional and unplanned rework.

Conformity

Software, unlike a physical product, has no underlying natural principles which it must conform to, such as Newton’s laws of motion. However, software must conform to exacting specifications in the representation of each of its parts, in the interfaces to other internal parts, and in the connections to the environment in which it operates. A missing semicolon or other syntactic error can be detected by a compiler, but a defect in the program logic or a timing error may be difficult to detect until encountered during operation.

Unlike software, tolerance among the interfaces of physical entities is the foundation of manufacturing and assembly. No two physical parts that are joined together have, or are required to have, exact matches. There are no corresponding tolerances in the interfaces among software entities or between software entities and their environments. There are no interface specifications for

software stating that a parameter can be *an integer plus or minus 2%.* Interfaces among software parts must agree exactly in numbers, types of parameters and kinds of couplings.

Lack of conformity can cause problems when an existing software component cannot be reused as planned because it does not conform to the needs of the product under development. Lack of conformity might not be discovered until late in a project, thus necessitating the development and integration of an acceptable component to replace the one that cannot be reused. This requires an unplanned allocation of resources (usually) and can delay project completion.

Changeability

Software coordinates the operation of physical components and provides most of the functionality in software-intensive systems. Because software is the most malleable (easily changed) element in a software-intensive system, it is the most frequently changed element. This is particularly true during the late stages of a development project and during system sustainment. However, this does not mean that software is easy to change. Complexity and the need for conformity can make changing software an extremely difficult task. Changing one part of a software system often results in undesired side effects in other parts of the system, requiring more changes before the software can operate at maximum efficiency.

Invisibility

Software is said to be invisible because it has no physical properties. While the effects of executing software on a digital computer are observable, software itself cannot be seen, tasted, smelled, touched, or heard. Software is an intangible entity because our five human senses are incapable of directly sensing it.

Work products such as requirements specifications, design documents, source code and object code are representations of software, but they are not the software. At the most elemental level, software resides in the magnetization and current flow in an enormous number of electronic elements within a digital device. Because software has no physical presence, software engineers must use different representations at different levels of abstraction in an attempt to visualize the inherently invisible entity.

Uniqueness

One other point about the nature of software that Brooks alludes to but does not explicitly call out is the uniqueness of software. Software and software projects are unique for the following reasons:

- Software has no physical properties;
- Software is the product of intellect-intensive teamwork;
- Productivity of software developers varies more widely than the productivity of other engineering disciplines;
- Estimation and planning for software projects is characterized by a high degree of uncertainty, which can be at best partially mitigated by best practices;
- Risk management for software projects is predominantly process-oriented;
- Software alone is useless, as it is always a part of a larger system; and
- Software is the most frequently changed element of software intensive systems.

References

Works Cited

Brooks, F. 1995. *The Mythical Man-Month,* Anniversary Edition. Boston, MA, USA: Addison Wesley Longman Inc.

Fairley, R.E. 2009. *Managing and Leading Software Projects.* Hoboken, New Jersey: John Wiley and Sons.

Primary References

Bourque, P. and R.E. Fairley (eds.). 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. Los Alamitos, CA, USA: IEEE Computer Society. Available at: http://www.Swebok.org.

Brooks, F. 1995. *The Mythical Man-Month,* Anniversary Edition. Boston, MA, USA: Addison Wesley Longman Inc.

Fairley, R.E. 2009. *Managing and Leading Software Projects.* Hoboken, New Jersey: John Wiley and Sons.

< Previous Article | Parent Article | Next Article >
SEBoK v. 2.10, released 06 May 2024