BACS3003 SOFTWARE EVOLUTION AND MAINTENANCE

CHAPTER 1

Fundamental of Software Maintenance and Evolution

History

- Mark Halpern established the notion of software evolution in 1965 to characterize the growth characteristics of software.
- Swanson coined the term "maintenance" by classifying maintenance tasks into three categories: corrective, adaptive, and perfective (in 1967).
- People who make deliberate changes to software code that they did not write are referred to as "maintenance engineers" or "maintainers" by IBM.

Software Maintenance and Software Evolution

- **software maintenance** means "*preventing software from failing to deliver the intended functionalities by means of bug fixing*".
- **software evolution** means "*a continual change from a lesser, simpler, or worse state to a higher or better state*".

Bennett and Rajlich (2000)

Software Maintenance and Software Evolution

- All support activities carried out *after* delivery of software are put under the category of *maintenance*.
- All activities carried out to effect changes in requirements are put under the category of *evolution*.

Bennett and Xu(2003)

Software Systems Evolution

- Taking existing designs and creating new ones that are related to them.
- Supporting additional functionalities, improving system performance, and allowing the system to run on a different operating system are among the goals.
- Essentially, as time passes, stakeholders and users have a better understanding of the system
- .“Over time what evolves is not the software but our knowledge about a particular type of software”- Mehdi Jazayeri

Challenges of Software Evolution

- Software Aging
- Incompatibility (Operating environment)

Can you think of any other challenges?

Extra !!!

Challenges of Software Evolution (Mens et al., 2005)

Software organisation productivity and quality in general continue to fall short of expectations, and software systems continue to show signs of ageing as they adapt to new requirements.

One of the fundamental causes of this issue is that in traditional software development methods, software maintenance and adaptation are still devalued.

Challenges of Software Evolution (Mens et al., 2005)

1. Preserving and improving software quality

In all industries, the negative impacts of software ageing can and will have a significant economic and social impact. As a result, developing tools and approaches to reverse or avoid the inherent difficulties of software ageing is critical. As a result, the goal is to develop tools and processes that retain or even improve a software system's quality features, regardless of its size or complexity.

Challenges of Software Evolution (Mens et al., 2005)

2. A common software evolution platform

•When attempting to solve the prior problem, scalability becomes a huge issue. The goal is to provide solutions that can be used in long-term, industrial-scale software systems. Many of the tools needed to handle the complexity inherent in software evolution are far too sophisticated for single research groups or individuals to develop.

•As a result, Michele Lanza has proposed a similar challenge: developing and supporting a single application framework for conducting cooperative software evolution research. This problem raises challenges like tool integration and interoperability, as well as common exchange formats and standards.

Challenges of Software Evolution (Mens et al., 2005)

3. Supporting model evolution

It has been found that practically all available software evolution tool assistance is largely geared at applications (i.e., source code). Typically, the design and modelling phases (as aided by UML CASE tools, for example) give far less support for software evolution. Taking refactoring as an example, we found no modelling tool that provided acceptable means for refactoring design models.

Challenges of Software Evolution (Mens et al., 2005)

4. Supporting co-evolution

The necessity of achieving co-evolution across distinct types of software artefacts or different representations of them is an issue connected to the prior one. That is, to ensure consistency of all relevant software artefacts, changes in one representation should always be reflected by matching modifications in others.

Challenges of Software Evolution (Mens et al., 2005)

5. Formal support for evolution

Even if tiny localised changes to a program's specification are made with current verification methods, the complete programme must be evaluated again. As a result, the cost of verification is proportional to the system's size. The ideal situation is for it to be proportional to the size of the units of change.

Formal techniques must accept change and evolution as a natural part of existence in order to be acknowledged as useful tools for software developers.

Software Evolution

What are the factors that influence software evolution?

Laws Explaining Software Evolution (Cook et.al, 2006)

Software System Evolution Characteristics.

1. *Continuing change (1974).*

E-type systems must be constantly (continuously) updated or they will become increasingly unsatisfactory.

2. *Increase complexity (1974).*

Maintenance-related changes will make a system increasingly complicated unless further work is done to explicitly reduce its complexity.

Laws Explaining Software Evolution (Cook et.al, 2006)

Software System Evolution Characteristics.

3. *Continuing growth(1980)*

The functional content of a system is continually upgraded (grow) over time to meet user needs.

4. *Declining quality (1996)*

Unless a system's design is meticulously fine-tuned and adapted to new operational circumstances, the system's attributes (quality) will be seen as deteriorating over time.

Laws Explaining Software Evolution (Cook et.al, 2006)

Organizational, economic resources constraints

5. *Conservation of organizational stability (1980).*

Over the life of a product, the average effective global activity rate in an evolving E-type system remains constant.

6. *Conservation of familiarity(1980).*

The average content of subsequent releases is invariant during the active life of an evolving E-type system.

Laws Explaining Software Evolution (Cook et.al, 2006)

Meta-Laws

7. Self-regulation (1974).

The evolution of E-type systems is self-regulating, with a close-to-normal distribution of product and process measures over time.

8. Feedback system (1996).

E-type systems' evolution processes are multi-level, multi-loop, multi-agent feedback systems that must be treated as such in order to achieve significant improvements over any plausible baseline.

Software Systems Maintenance

- Software maintenance is essential to ensure the long-term viability of software products throughout their lifecycle. i.e. from conception to completion.
- Changes to the software product are documented, the effects (impacts) of the changes are discovered, artefacts are modified, testing is performed, and a new software release is prepared.
- Users are educated, and assistance is available at all times.
- The term "maintainer" refers to a company or organisation that provides maintenance work.

Source : http://swebokwiki.org/Chapter_5:_Software_Maintenance

Software Development vs Software Maintenance

Software Development

- requirements driven
- Process begins with the objective of designing and implementing a system to deliver certain functional and nonfunctional requirements.
- Primary/First Implementation.
- Development a software from new.
- Deliver new software.

Software Maintenance

- event driven
- scheduled in response to an event.
- on-going administration/support.
- Modifying or adding new features based on existing software.(May include developing software)
- Prevent software from failure.

Reasons of Maintenance

- Prevent software operation from failure.
- Fix discovered software bugs.
- Improve software. Eg. to link or access to other new devices.
- Ensure software works in different environment (if required)
- Provide better experiences to users.
- Comply with organizational goal (if changes)

References

- 1. Blokdyk. G. ((2020). Maintenance of software : a complete guide : practical tools for self-assessment. Art of Service.
- 2. Blokdyk. G.. (2020). Software change management : a complete guide : practical tools for self-assessment. Art of Service.
- 3. Maxim. B. R., Pressman. R. S. (2020) Software Engineering: A Practitioner's Approach. 9th Edn. McGraw-Hill Education

Other references

- 1. E. Varga. (2017). *Unraveling software maintenance and evolution : thinking outside the box*. Springer.
- 2. F. Tsui. O. Karam. B. Bernal. (2018). *Essentials of Software Engineering*. 4th Edition. Jones & Bartlett Learning.
- 3. Tripathy, P., Naik K. (2014). *Software Evolution and Maintenance – A Practitioner's Approach*. John Wiley and Son. Springer Vieweg.
- Cook, S. , Harrison, R. , Lehman, M.M. , Wernick, P. , 2006. Evolution in software systems: foundations of the spe classification scheme. J. Softw. Maintenance 18 (1), 1–35 .
- http://swebokwiki.org/Chapter_5:_Software_Maintenance
- T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld and M. Jazayeri, "Challenges in software evolution," Eighth International Workshop on Principles of Software Evolution (IWPSE'05), 2005, pp. 13-22, doi: 10.1109/IWPSE.2005.7.