Software maintenance

Software maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes.[1]

A common perception of maintenance is that it merely involves fixing defects. However, one study indicated that over 80% of maintenance effort is used for non-corrective actions.[2] This perception is perpetuated by users submitting problem reports that in reality are functionality enhancements to the system. More recent studies put the bug-fixing proportion closer to 21%.[3]

Contents

History

Software maintenance and evolution of systems was first addressed by Meir M. Lehman in 1969. Over a period of twenty years, his research led to the formulation of Lehman's Laws (Lehman 1997). Key findings of his research include that maintenance is really evolutionary development and that maintenance decisions are aided by understanding what happens to systems (and software) over time. Lehman demonstrated that systems continue to evolve over time. As they evolve, they grow more complex unless some action such as code refactoring is taken to reduce the complexity.

In the late 1970s, a famous and widely cited survey study by Lientz and Swanson, exposed the very high fraction of life-cycle costs that were being expended on maintenance. They categorized maintenance activities into four classes:

- Adaptive – modifying the system to cope with changes in the software environment (DBMS, OS) [4]
- Perfective – implementing new or changed user requirements which concern functional enhancements to the software
- Corrective – diagnosing and fixing errors, possibly ones found by users [4]
- Preventive – increasing software maintainability or reliability to prevent problems in the future [4]

The survey showed that around 75% of the maintenance effort was on the first two types, and error correction consumed about 21%. Many subsequent studies suggest a similar magnitude of the problem. Studies show that contribution of end user is crucial during the new requirement data gathering and analysis. And this is the main cause of any problem during software evolution and maintenance. So software maintenance is important because it consumes a large part of the overall lifecycle costs and also the inability to change software quickly and reliably means that business opportunities are lost. [5] [6] [7]

Importance of software maintenance

The key software maintenance issues are both managerial and technical. Key management issues are: alignment with customer priorities, staffing, which organization does maintenance, estimating costs. Key technical issues are: limited understanding, impact analysis, testing, maintainability measurement.

Software maintenance is a very broad activity that includes error correction, enhancements of capabilities, deletion of obsolete capabilities, and optimization. Because change is inevitable, mechanisms must be developed for evaluation, controlling and making modifications.

So any work done to change the software after it is in operation is considered to be maintenance work. The purpose is to preserve the value of software over the time. The value can be enhanced by expanding the customer base, meeting additional requirements, becoming easier to use, more efficient and employing newer technology. Maintenance may span for 20 years, whereas development may be 1–2 years.

Software maintenance planning

An integral part of software is the maintenance one, which requires an accurate maintenance plan to be prepared during the software development. It should specify how users will request modifications or report problems. The budget should include resource and cost estimates. A new decision should be addressed for the developing of every new system feature and its quality objectives. The software maintenance, which can last for 5–6 years (or even decades) after the development process, calls for an effective plan which can address the scope of software maintenance, the tailoring of the post delivery/deployment process, the designation of who will provide maintenance, and an estimate of the life-cycle costs. The selection of proper enforcement of standards is the challenging task right from early stage of software engineering which has not got definite importance by the concerned stakeholders.

Software maintenance processes

This section describes the six software maintenance processes as:

1. The implementation process contains software preparation and transition activities, such as the conception and creation of the maintenance plan; the preparation for handling problems identified during development; and the follow-up on product configuration management.
2. The problem and modification analysis process, which is executed once the application has become the responsibility of the maintenance group. The maintenance programmer must analyze each request, confirm it (by reproducing the situation) and check its validity, investigate it and propose a solution, document the request and the solution proposal, and finally, obtain all the required authorizations to apply the modifications.
3. The process considering the implementation of the modification itself.
4. The process acceptance of the modification, by confirming the modified work with the individual who submitted the request in order to make sure the modification provided a solution.
5. The migration process (platform migration, for example) is exceptional, and is not part of daily maintenance tasks. If the software must be ported to another platform without any change in functionality, this process will be used and a maintenance project team is likely to be assigned to this task.
6. Finally, the last maintenance process, also an event which does not occur on a daily basis, is the retirement of a piece of software.

There are a number of processes, activities and practices that are unique to maintainers, for example:

- Transition: a controlled and coordinated sequence of activities during which a system is transferred progressively from the developer to the maintainer;
- Service Level Agreements (SLAs) and specialized (domain-specific) maintenance contracts negotiated by maintainers;
- Modification Request and Problem Report Help Desk: a problem-handling process used by maintainers to prioritize, documents and route the requests they receive;

Categories of maintenance in ISO/IEC 14764

E.B. Swanson initially identified three categories of maintenance: corrective, adaptive, and perfective.[8] The **IEEE 1219** standard was superseded in June 2010 by **P14764**.[9] These have since been updated and ISO/IEC 14764 presents:

- Corrective maintenance: Reactive modification of a software product performed after delivery to correct discovered problems.
- Adaptive maintenance: Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment.
- Perfective maintenance: Modification of a software product after delivery to improve performance or maintainability.
- Preventive maintenance: Modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults.

There is also a notion of pre-delivery/pre-release maintenance which is all the good things you do to lower the total cost of ownership of the software. Things like compliance with coding standards that includes software maintainability goals. The management of coupling and cohesion of the software. The attainment of software supportability goals (SAE JA1004, JA1005 and JA1006 for example). Note also that some academic institutions are carrying out research to quantify the cost to ongoing software maintenance due to the lack of resources such as design documents and system/software comprehension training and resources (multiply costs by approx. 1.5-2.0 where there is no design data available).

Maintenance Factors

Impact of key adjustment factors on maintenance (sorted in order of maximum positive impact)

Maintenance Factors	Plus Range
Maintenance specialists	35%
High staff experience	34%
Table-driven variables and data	33%
Low complexity of base code	32%
Y2K and special search engines	30%
Code restructuring tools	29%
Re-engineering tools	27%
High level programming languages	25%
Reverse engineering tools	23%
Complexity analysis tools	20%
Defect tracking tools	20%
Y2K “mass update” specialists	20%
Automated change control tools	18%
Unpaid overtime	18%
Quality measurements	16%
Formal base code inspections	15%
Regression test libraries	15%
Excellent response time	12%
Annual training of > 10 days	12%
High management experience	12%
HELP desk automation	12%
No error prone modules	10%
On-line defect reporting	10%
Productivity measurements	8%
Excellent ease of use	7%
User satisfaction measurements	5%
High team morale	5%
Sum	**503%**

Not only are error-prone modules troublesome, but many other factors can degrade performance too. For example, very complex “spaghetti code” is quite difficult to maintain safely. A very common situation which often degrades performance is lack of suitable maintenance tools, such as defect tracking software, change management software, and test library software. Below describe some of the factors and the range of impact on software maintenance.

Impact of key adjustment factors on maintenance (sorted in order of maximum negative impact)

Maintenance Factors	Minus Range
Error prone modules	-50%
Embedded variables and data	-45%
Staff inexperience	-40%
High code complexity	-30%

No Y2K of special search engines	-28%
Manual change control methods	-27%
Low level programming languages	-25%
No defect tracking tools	-24%
No Y2K “mass update” specialists	-22%
Poor ease of use	-18%
No quality measurements	-18%
No maintenance specialists	-18%
Poor response time	-16%
No code inspections	-15%
No regression test libraries	-15%
No help desk automation	-15%
No on-line defect reporting	-12%
Management inexperience	-15%
No code restructuring tools	-10%
No annual training	-10%
No reengineering tools	-10%
No reverse-engineering tools	-10%
No complexity analysis tools	-10%
No productivity measurements	-7%
Poor team morale	-6%
No user satisfaction measurements	-4%
No unpaid overtime	0%
Sum	**-500%**

[10]

See also

- Application retirement
- *Journal of Software Maintenance and Evolution: Research and Practice*
- Long-term support
- Search-based software engineering
- Software archaeology
- Software maintainer
- Software development

References

[11]

1. "ISO/IEC 14764:2006 Software Engineering — Software Life Cycle Processes — Maintenance" (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39064). Iso.org. 2011-12-17. Retrieved 2013-12-02.
2. Pigoski, Thomas M., 1997: Practical software maintenance: Best practices for managing your software investment. Wiley Computer Pub. (New York)
3. Eick, S., Graves, T., Karr, A., Marron, J., and Mockus, A. 2001. Does Code Decay? Assessing Evidence from Change Management Data. IEEE Transactions on Software Engineering. 27(1) 1-12.
4. *Software Maintenance and Re-engineering* (http://www.csse.monash.edu.au/~jonmc/CSE2305/Topics/13.25.SWEng4/html/text.html), CSE2305 Object-Oriented Software Engineering
5. Lientz B., Swanson E., 1980: Software Maintenance Management. Addison Wesley, Reading, MA
6. Lehman M. M., 1980: Program, Life-Cycles and the Laws of Software Evolution. In Proceedings of IEEE, 68, 9,1060-1076
7. Penny Grubb, Armstrong A. Takang, 2003: Software Maintenance: Concepts and Practice. World Scientific Publishing Company

8. "E. Burt Swanson, The dimensions of maintenance. Proceedings of the 2nd international conference on Software engineering, San Francisco, 1976, pp 492 — 497" (http://portal.acm.org/citation.cfm?id=359522). Portal.acm.org. doi:10.1145/359511.359522 (https://doi.org/10.1145%2F359511.359522). Retrieved 2013-12-02.
9. Status of 1219-1998 (http://standards.ieee.org/cgi-bin/status?1219-1998) by IEEE Standards
10. "The Economics Of Software Maintenance In The Twenty First Century" (https://web.archive.org/web/20150319075401/http://www.compaid.com/caiinternet/ezine/capersjones-maintenance.pdf) (PDF). Archived from the original (http://www.compaid.com/caiinternet/ezine/capersjones-maintenance.pdf) (PDF) on 2015-03-19. Retrieved 2013-12-02.
11. Pigoski, Thomas. "Chapter 6: Software Maintenance" (http://sce.uhcl.edu/helm/SWEBOK_IEEE/data/swebok_chapter_06.pdf) (PDF). *SWEBOK*. IEEE. Retrieved 5 November 2012.

Further reading

- *ISO/IEC 14764 IEEE Std 14764-2006 Software Engineering — Software Life Cycle Processes — Maintenance*. 2006. doi:10.1109/IEEESTD.2006.235774 (https://doi.org/10.1109%2FIEEESTD.2006.235774). ISBN 0-7381-4960-8.
- Pigoski, Thomas M. (1996). *Practical Software Maintenance*. New York: John Wiley & Sons. ISBN 978-0-471-17001-3.
- Pigoski, Thomas M. *Description for Software Evolution and Maintenance (version 0.5)*. SWEBOK Knowledge Area.
- April, Alain; Abran, Alain (2008). *Software Maintenance Management*. New York: Wiley-IEEE. ISBN 978-0-470-14707-8.
- Gopalaswamy Ramesh; Ramesh Bhattiprolu (2006). *Software maintenance : effective practices for geographically distributed environments*. New Delhi: Tata McGraw-Hill. ISBN 978-0-07-048345-3.
- Grubb, Penny; Takang, Armstrong (2003). *Software Maintenance*. New Jersey: World Scientific Publishing. ISBN 978-981-238-425-6.
- Lehman, M.M.; Belady, L.A. (1985). *Program evolution : processes of software change*. London: Academic Press Inc. ISBN 0-12-442441-4.
- Page-Jones, Meilir (1980). *The Practical Guide to Structured Systems Design*. New York: Yourdon Press. ISBN 0-917072-17-0.

External links

- Journal of Software Maintenance (http://www3.interscience.wiley.com/cgi-bin/jhome/5391/)
