Establishing Economic Effectiveness through Software Health-Management

Markus Pizka and Thomas Panas, *itestra GmbH*

***Abstract*—More than two thirds of the annual software budget of large-scale organizations dealing with complex software systems is spent on the perfection, correction, and operation of existing software systems. A significant part of these running costs could be saved if the software systems that need to be constantly extended, maintained and operated were in a better technical condition. This paper proposes Software Health-Checks as a method to assess the technical condition of existing software systems and to deduce measures for improving the health of software in a structured manner. Since 2006 numerous commercial software systems with a total of 30 MLOC[1], implemented in various technologies, were already checked with this method. The actions suggested as a result of these Software 'Health-Checks', repeatedly yielded dramatic performance improvements, risk reductions and cost savings between 30% and 80%.**

***Index Terms*—software quality, software maintenance, software management, software economics**

I. INTRODUCTION

Integrated Systems Health Management is used in complex heterogeneous physical systems to continuously monitor the state of (sub-)systems and to take appropriate actions in case of anomalies. Unfortunately, there are only few and barely mature techniques to monitor the health of software systems in a similar way. The closest matches are presumably the results of research on fault tolerant systems [1] on the one hand and commercial systems management solutions, such as IBM Tivoli [2] for large scale information systems on the other hand.

Besides the absence of health management facilities similar to those found in systems management, we strongly argue that a proper health management of software systems should not solely focus on correct operation of the software system but on overall short-, mid-, and long-term economic effectiveness of software. In accordance with the optimization strategy described in Software Reengineering Assessment Handbook of the US Department of Defense [8], the intrinsic goal of software health management has to be the optimization of the overall economic effectiveness and strategic suitability of software systems.

M. Pizka and T. Panas are with itestra GmbH, 85748 Garching, Germany (corresponding author: M. Pizka, phone: +49 (179) 2108101; e-mail: pizka@itestra.de).

[1] Million lines of code

A. Situation

More than 70% of the overall software budgets of larger organizations are spent on maintaining and operating existing software systems [8], [3]. At the same time large-scale software systems are known to suffer from a gradual quality decay over time if no pro-active countermeasures are taken [4], [5]. This decay affects all of the quality attributes defined with the ISO 9216 software quality standard: reliability, functionality, efficiency, portability, usability, maintainability [7], and security.

Consequently, poorly performing, unstable, misaligned and inflexible systems cause enormous annual costs. Although there is a correlation between the age of a system and the degree of decay, there are numerous other reasons for decreasing reliability and performance besides the age of a software system. As described in [13], many software systems show severe signs of decay causing excessive cost of ownership right after and sometimes even before the first release.

B. Requirement Software Health-Management

Gradual and even rapid decay, along with the increasing risk and cost of ownership, can be mitigated effectively by

a) performing health-checks (structured assessments) of the state of the software system on a regular basis and

b) taking immediate action to remove the signs of decay detected during these health-checks.

As our experience shows, implementing health-checks and subsequently enforcing actions to eliminate the effects of decay reduces costs and risks. This thereby extends the life-time of software systems. Based on D.L. Parnas' seminal work on "software aging" [4], we call this iterative process software health management.

Note that this view on software health management is deliberately not restricted to a particular quality attribute (such as correctness or reliability during operation) but aims to increase overall economic effectiveness. Therefore it would be unrealistic to assume that this kind of long-term oriented software health management could be performed automatically or even built into systems. Instead, software health management, as described in this paper, is based on a combination of tool-supported analyses, expert reviews and manually performed counter-actions.

Outline

Section II explains the quality model and the analysis process that we use to assess the health of software systems. Thereafter, this paper focuses on our experiences performing software health management in practice. We show key findings from software health checks on more than 30 MLOC and outline actual improvements achieved in Section III.

II. Health Check Model and Process

A. Software Quality Equals Economic Effectiveness

In order to assess the health condition of a software system, one needs to establish a proper software health model, which in turn requires software quality to be measured. As stated in [10] and others the frequently used term software quality has many different meanings.

The most commonly used definition of software quality is "conformance to a specification". However, this entails that quality measurement results are meaningless if the initial specification is incomplete or weak by itself. Since most specifications in practical settings are indeed weak, virtually every software system was deemed high quality under this definition, which is certainly untrue.

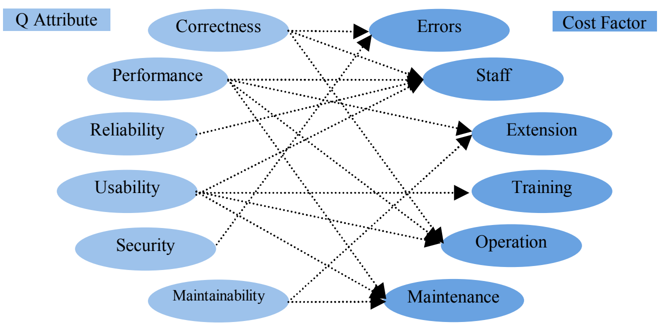

Fig. 1. Practically relevant software quality equals economic effectiveness.

There are, of course, numerous other definitions for software quality besides "conformance" as in [7]. However, the ISO9126 standard as well as many other definitions from research on software metrics fall short of explaining the actual importance of software quality defects. For instance, although the cyclomatic complexity metric (CC) [14], that tries to compute program complexity by counting call graph dependencies, is agreed upon to be important, the actual effect of an increased or decreased CC for a particular software system is rather unclear.

In contrast to this, the improvement process, described in the Software Reengineering Assessment Handbook [8], relates technical properties of software systems with economic effects and strategic considerations to generate a complete, actionable and enforceable view on the state of a system and options to deal with it.

We strongly support this integrated view, particularly the strong combination of technical observations and economic impacts, and therefore propose a value-based view on software quality as sketched in Figure 1. With this quality model in mind, a software system has high quality (i.e. is healthy) if and only if its costs are low.

Tough this economically-based definition of quality might initially sound exaggerated it simply reflects practically reality. For example, the only reason to increase the performance of a software system is to reduce costs through a weighted combination of a) reduced waiting cycles of users, b) reduced resource consumption, c) reduced testing effort, and d) decreased risk of failure (e.g. due to buffer overruns). Sometimes, building the business case for a particular quality attribute is certainly challenging but nevertheless indispensable because virtually every statement about quality will be ignored in the long run if only the costs of achieving it are quantified but not it benefits. One has to accept the reality that this even holds for properties such as safety and security. If the sum of the expected disadvantages and penalties of a security flaw is lower or equal the cost of avoiding or fixing it, it will most likely be ignored.

Among the consequences of this model are: First, it guarantees that everything that is regarded during quality analysis is relevant to the owner of the system, because everything gets mapped onto the actual cost structure. Second, it allows assessing the quality of a software system from two different perspectives, i.e. economics and technical properties. E.g. a system that does not cause any maintenance costs is by definition highly maintainable. There is hardly any need for a sophisticated technical analysis of the maintainability metrics such as the SEI maintainability index in this case. At the same time, a system that handles large volume of data with inadequate algorithms, such as bubble-sorts or in single linked lists, will be unnecessarily expensive for its owner. Hence, defining cost effectiveness as the quality goal allows combining economic and technical data during quality analysis which produces highly relevant health-check results in a very efficient way.

B. Economics-based two-dimensional software quality model

Based on this notion of software quality, we developed a quality model that organizes the criteria that need to be assessed during software health-checks into two dimensions, see figure 2.

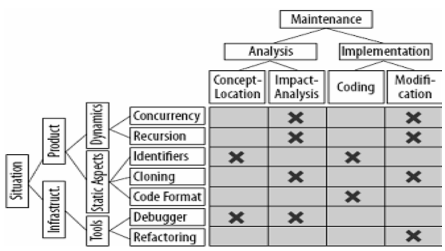

Fig 2. Two-dimensional quality model

At the top of this model is the breakdown structure of all activities that are performed on a software system and represent the major costs. The activities used in a certain setting depend on the organization that owns the software, its processes and its strategy. Typical relevant activities are maintenance (as shown as an example in fig. 2), development, and operations but also repairing damages due to software failure.

At the left of the models are the facts that describe the technical state of the software system including properties of the organization such as it process maturity. The technical facts are more or less context-independent with minor variations between different technologies (e.g. COBOL, Java).

The current version of our quality model encompasses 260 criteria that were chosen because of their strong impact on certain activities and therefore costs. Selected examples are:

- Sample facts about the code: cloning ratio, unused code, number of workarounds, conditional ratio, architectural violations, quality of naming
- Sample facts about the documentation: homonym ratio, synonym ratio, completeness, actuality
- Sample facts about the organization: CMMi level, number of employees with process know-how, number of employees with system know-how

For more information about our quality-model, please refer to [6][9][12].

C. Health Management Process

During our Health-Check, we initially determine costs (top) and then the technical properties (left) of a software system. The analysis of technical properties consists of the following steps:

- Retrieve artefacts, i.e. code, documentation, execution profile, economic statements (invoices, etc.)
- Perform interviews with key stakeholders to collect facts about the organization, processes, etc.
- Tool-supported static analysis of the code base with ConQAT [11]; i.e. analysis of the size of the system, cloning ratio, loop nesting, comment ratio, and other metrics
- Manual inspection of the code and documents
- Review of the analysis results with technical experts of system (e.g. former developers)
- Design of improvement actions if indicated
- Planning and ROI (return on investment) estimation for all improvement actions
- Presentation of the results to the owners of the software system, who will decide whether optimizations are executed

Note, the analysis uses a tool (ConQAT) only to collect some but important facts about the code and its documentation and to guide manual inspection. All other facts are analysed either through manual inspection or through interviews. However, in practice, the complete analysis phase takes only 5 to 15 man-days, depending on the size of the system under consideration.

From the initial Health-Check, Health Management proceeds with executing appropriate actions to eliminate the quality defects detected. The time and effort needed to implement these actions clearly depends on the number and complexity of the selected actions. However, most times improvement actions are only performed if they are completed and yield a positive ROI within less than 12 months

Despite of numerous technical challenges, the biggest challenge to successfully improving the health of software systems is an organizational and psychological issue: i.e. how to gain and preserver acceptance and trust from the stakeholders of the system. Original developers commonly do not understand the need for change, and managers responsible for such systems are also averse to change. This is mainly because managers are afraid that they could be made responsible for actions that they incorrectly or insufficiently supervised in the past. Our Health-Check uses, amongst others, two essential techniques to overcome these problems:

1. We structure the presentation of health-check results according to importance so that management can easily be convinced of problems inherent in their software systems. For this, our initial slide shows the economic potential of improvements followed by an overview of health-check results. Thereafter, we present more details, down to code fragments showing the weaknesses. Interestingly, showing code repeatedly proved to be the most convincing information – even to top managers.
2. We take full responsibility for our actions. Our funding and success is dependent on the success of our optimizations. Our customers pay based on our performance and results we achieve and not according to our initial projections.

III. Experiences

itestra has applied its health-check model to real world systems implemented in PL/I, C, COBOL, Java, Matlab/Simulink and PHP, since 2006. The total size of all systems analyzed exceeds 30 MLOC (million lines of code). These systems are worth about $500 million in assets and create $50 million in annual costs for development and maintenance. Our health-check of these systems indicated that these annual costs can be reduced by at least 30% within one year. Figure 3 shows the distribution of an analysis of 20 systems.

Every dot in the graph represents an individual software system. The horizontal position of the dot corresponds with the actual total annual costs in thousand Euros of this system. E.g., the bullet at the top right corner of the graph corresponds to a system with annual costs of almost 2 Mio EUR for development, maintenance, operation, and hard-/software resource consumption. The vertical position, i.e. the technical quality index (TQI), indicates the aggregated technical abnormality of the system relative to the average of all systems assessed (higher ~ worse, lower ~ better). The middle of the vertical axis marked with “0”, corresponds to the average of all measurements. The TQI is computed by first mapping every measurement of a quality attribute on a scale from – 3 (best) to +3 (worst) according to the deviation of the measurement result for the particular system from the average of all systems. Second, these mappings from measurement results to the [-3;+3] penalty scale are summed up per system.

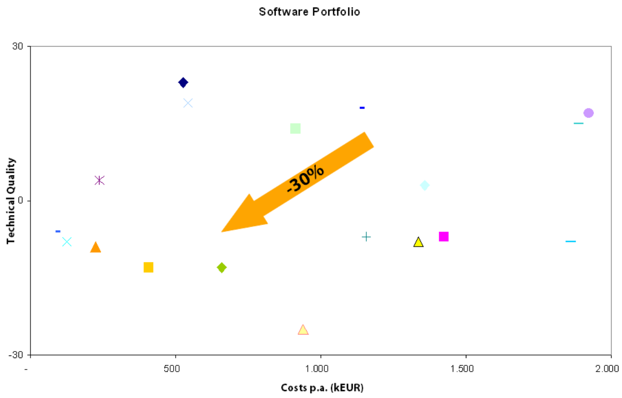

Figure 3: Health-Check of 20 software systems.

The results of this approach clearly expose systems that are both expensive and technically insufficient. I.e. it can be expected that the costs of systems in the top-right quadrant (e.g. the bullet at the top right corner) can be significantly reduced by fixing the technical quality defects detected.

In this particular analysis of 20 systems it allowed us to estimate that 30% of the annual costs of $50 million could be economized.

As a matter of fact, the improvements achieved so far during implementation span from at least 35% up to 80% of the annual development, operational and maintenance costs.

Within this large-scale assessment, we also noticed growing demand for proper software health management. Our customers quickly understood and adopted software health management practices, which is the primarily result of our strong efforts to communicate our analyses and findings in the most effective ways possible.

IV. Conclusion

Software health checks are essential, especially for aging systems. This is comparable to humans that perform preventive health checks, thereby lowering their risk of diseases and treatment costs. Our health check detects crucial weaknesses and risks in software. These checks have a profound influence on the running costs of such systems. Even if not broken, such systems function less efficiently and are more prone to failures.

We learned that many systems are in astonishingly poor technical condition and because of this software health management is crucial – not only to correct software and hence reduce system downtime, but primarily to drastically reduce software operation and maintenance costs.

Today, companies are still forced to spend large sums to keep these systems running just because the causes of failure and inefficiency are not understood.

Our health check helps to discover software weaknesses and allows to drastically cut running costs. Besides, healthy software has the advantage of longer live expectancy which means that risky legacy migration scenarios can be avoided or at least vastly deferred.

References

[1] Brian Randell. System Structure for Software Fault Tolerance. IEEE Transactions on Software Engineering, vol. 1, 1975, pages 220 – 232.

[2] IBM. IBM Tivoli software. http://www-01.ibm.com/software/tivoli/, 2009.

[3] Accenture. Editorial - only 40% of the IT budget for new solutions. IS report, June 2003.

[4] David Lorge Parnas. Software aging. In Proc. International Conference on Software Engineering (ICSE ’94), pages 279–287. IEEE Computer Society, 1994.

[5] Stephen G. Eick, Todd L. Graves, Alan F. Karr, J. S. Marron, and Audris Mockus. Doescode decay? assessing the evidence from change management data. IEEE Trans. Softw. Eng., 27(1):1–12, 2001.

[6] Manfred Broy, Florian Deißenböck, and Markus Pizka. Demystifying maintainability. In Proc. of the 2006 Int. Workshop on Software Quality. Shanghai, China, 2006.

[7] ISO 9126-1 Software engineering - Product quality - Part 1: Quality model. International standard, ISO, 2003.

[8] STSC. Software Reengineering Assessment Handbook v3.0. Technical report, STSC, U.S. Department of Defense, Mar. 1997.

[9] Florian Deißenböck, Markus Pizka et al. Tool Support for Continuous Quality Control. IEEE Software, vol. 25, 2008, pages 60 – 67.

[10] Barbara Kitchenham and Shari Lawrence Pfleeger. Software quality: The elusive target. IEEE Software, 13(1):12–21, 1996.

[11] ConQAT. http://conqat.cs.tum.edu/.

[12] Florian Deissenboeck, Markus Pizka, and Tilman Seifert. Tool support for continuous quality assessment. In Proc. IEEE International Workshop on Software Technology and Engineering Practice (STEP), pages 127–136. IEEE Computer Society, 2005.

[13] Benedikt Mas y Parareda and Markus Pizka. Web-based and other young legacy-systems. information Management & Consulting, 22(2), June 2007.

[14] T. McCabe. A Complexity Measure. International Conference on Software Engineering, pages 308 – 320, 1976.