IBIMA Publishing

mobile

Journal of Software and Systems Development

Vol. 2014 (2014), Article ID 904507, 43 minipages.

DOI:10.5171/2014. 904507

www.ibimapublishing.com

Research Article

Software Ageing Measurement Framework Based on GQM Structure

Authors

Aziz Deraman[1], Jamaiah H. Yahaya[2], Zaiha Nadiah Zainal Abidin[3] and Noorazean Mohd Ali[4]

[1] Universiti Malaysia Terengganu, Terengganu, Malaysia

[2, 3, 4] Universiti Kebangsaan Malaysia, Bangi, Selangor, Malaysia

Received date: 22 April 2013;
Accepted date: 28 November 2013;
Published date: 6 January 2014

Academic Editor: Jan Mendonça Corrêa

Cite this Article as: Aziz Deraman, Jamaiah H. Yahaya, Zaiha Nadiah Zainal Abidin and Noorazeean Mohd Ali (2014), "Software Ageing Measurement Framework Based on GQM Structure," Journal of Software and Systems Development, Vol. 2014 (2014), Article ID 904507, DOI: 10.5171/2014.904507

Abstract

Previous studies have indicated that application software behaves in a manner that is almost similar to the process of human ageing. Similar to human beings getting old, it is believed that software too, can age in terms of its quality, usage and relevancy to the users and environment. Even though ageing is unavoidable, it is possible to understand the reasons for its occurrence so that steps can be taken to control its impact. Software product does not age in the physical sense like human being. However, in certain situations, the software loses its significance and quality to the users and environment. This manifestation can be illustrated as the process of ageing or ageing phenomenon. Our research group has classified the software ageing factors in terms of its product profile or contour, the environment, human aspect and the system functional. The factors are then identified through literature, expert interview and brainstorming amongst members of the research team. Inspired by earlier works concerning software certification and quality, we have led to develop a software ageing model together with its associated domain such as ageing factors and the rejuvenation index. In order to realize the model, a framework for measurement and classification of the software ageing factors is designed and developed. The design is carried out using Goal Question Metrics (GQM) approach. This paper presents the framework and the mapping of the associate metrics to the relevant objective and classifying of the ageing factors.

Keywords: Software Ageing, Software Ageing Measurement Framework, GQM, Ageing Factors and Classification

Introduction

Previous works in related areas in the field of software engineering have highlighted that application software shows the behaviour that is similar to ageing in human beings. It exhibits that software also ages and the process can be delayed and prolonged. However, it is possible for us to understand the underlying reasons and factors for the occurrence of ageing and thus actions can be taken to control its impact. Past research identified two categories of software ageing: 1) the type that is due to the inability of the products to adjust to the forces at work in the surroundings, and 2) the type that occurs as a consequence of modifications (Parnas, 1994). Software is considered as a logical product therefore it does not age in the physical sense. However, in certain situations, the software is losing its significance and quality to its environment. This can relates to the process of ageing. The ageing process in software differs from that in humans in the sense that the former can be expressed in terms of various factors such as its significance, frequency of failure, technology, environment, and so on. At the same time, software rejuvenation encompasses the construction of the software and the characteristics of the software that can be adapted and revised; the changes that must necessarily take place in the environment so the software can maintain its 'youth' and health. If the software ageing factors can be ascertained and distinguished, then the software can be rejuvenated and its ageing can be delayed.

In the field of software engineering, software ageing is a term that is used to describe the gradual deterioration of the functioning or condition of the software with the progress of time. However, time is not the only cause of software ageing, which is also strongly influenced by the quality of the software and whether that specific software can retain its quality during the whole of its lifetime. As the quality of the software tends to deteriorate with time, it is necessary to rejuvenate it in order to enhance the software and sustain its quality, while at the same time manage the factors influencing ageing. It has been indicated in previous researches that certification is one way in which software ageing can be controlled. The procedure for certification can be introduced at any point in its lifecycle in order to gauge its quality so as to be able to predict the deterioration of quality of the software (Yahaya, Deraman, & Hamdan, 2010; Voas, 1999).

We have managed to identify and explore the causes of software ageing and these are illustrated in this paper. Furthermore, we also examine in this paper the ageing measurements and classification developed using the Goal Question Metrics (GQM) approach. In the rest of the paper, we discuss the research within the context of software quality and certification, as well as matters concerning software ageing. A framework of software ageing measurement using GQM approach is presented and a conclusion is given in the final section of this paper.

Issues in Software Quality and Certification

It has been shown in earlier studies that the significance of the software at any point during its lifecycle is determined by its quality, and this characteristic is not presented in the available software quality models and preservation processes. A substantial factors, measurements and metrics that are encountered are attributed to the measurement of quality from external views and perspectives (Yahaya & Deraman, 2010). Several quality models have been developed from previous works such as McCall, Boehm, FURPS, ISO9126 and PQF model. With the current demand from users, technology and ecosystem, the quality model needs to be more simple, precise and practical so that it can be assessed by non-experts, users, customers, designers or investors. This is linked to the common meaning of quality as "fitness for use" and "conformance to requirements". Fitness for use is often taken to refer to features such as practicality, ease of use and maintenance, and the ability to be used repeatedly; whilst "conformance to requirements" signifies that the software is worth something to the user. Very few of the quality models of the past include or focus much on these requirements (Voas, 1999; Tervonen 1996). The PQF model highlights human and behavioural (or technical) characteristics. These give a more balance assessment of human needs or for meeting the expectations of users (Yahaya and Deraman, 2010).

Certification normally refers to the procedure whereby a value is attached to an item, person or organisation, and this value is validated by the issuance of a certificate, which can be produced as evidence of its authenticity. A software certification as a fact sheet presenting the proven external behaviours of the software (and possibly proven internal behaviours as well) (Voas, 1999). Stanfford and Wallnau (2001) also share the same view of certification as a procedure whereby the value given to a particular asset is validated by the issuance of a certificate which can be produced as evidence of its authenticity. This is the latest software engineering theory which is fast gaining recognition in the software industry. The IT Times (2011) referred to Good Software as a quality certification programme designed by the Koreans and which has been in use over the past ten years.

Most of the software certification methods depend mainly on official validation, professional and developer evaluations, and software measurements to ascertain the quality of an item. Another method involves measuring quality against the ISO9126 model (Lee, Ghandi, & Wagle, 2007; Welzel & Hausen, 1997; Heck et al., 2010). Some examples of this method are the Good Software mentioned in the IT Times, the Requirement-driven Workbench and the SCM-Prod (IT Times, 2011; Lee, Ghandi & Wagle, 2007; Yahaya, Deraman & Hamdan, 2008). Such models are appropriate for the normal evaluation of software with constant features such as portability, usability, dependability, maintainability, practicality and competence. The above mentioned studies are mainly aimed at validating software items from the viewpoint of developers, dealers and examiners and pay little attention to the viewpoint and participation of the users.

Earlier studies conducted by us concerning software product certification applications have proven that SCM-Prod model, which uses the product quality method for software certification, is suitable for maintaining the quality of the product during its lifecycle (Yahaya, Deraman & Hamdan, 2008; 2010). This model uses PQF model as its standard of quality certification. Our software certification model pays more attention to the participation of users in the assessment process. Data concerning certification and quality from 2007 until 2011 was analysed and has revealed that software measurements can be used to constantly observe quality. The quality of the software can be determined at any point in its lifecycle and therefore this can assist experts and developers to check the deterioration in the value and quality of the software. Once the software starts losing its value and quality in the environment, it is deemed to be shifting into the ageing phase of its lifecycle. Therefore, it is vital that this ageing phase be checked and prolonged by keeping the risks and conditions to a minimum. By carrying out the certification procedure on specific software items some important information can be acquired as to the quality of the software in the environment, which can then be linked to the ageing phase in the lifecycle (Yahaya & Deraman, 2012). This manifestation is illustrated in Fig. 1.

Fig 1 illustrates that software should follow the ideal curve to gain maximum quality throughout its life cycle. In reality and under some circumstances the ideal curve of quality may not be achievable by the software product. It may fall down under the ideal curve as shown in fig 1. There will be some affected reasons for this occurrence. With quality monitoring process such as Software Quality Assurance activity, Software Certification Method and etc. the quality of the software product can be improved. Based on this assumption and believe, software product can be better improved too and its quality can be enhanced if we can rejuvenate it based on anti-ageing actions. Further studies may be conducted to investigate and identify the affected reasons and factors that may cause this scenario and lead to earlier ageing of the software.

Software Ageing and Rejuvenation

Past researches investigated the issue and vocabulary of software ageing, which concern the gradual decrease of operating system resources, data corruption and the accumulation of statistical errors. Some examples of such software ageing include the bloating and leaking of memory, restricted file-locks, storage space, etc. These studies analyse the ageing of the Linux OS and operating system software (Cotroneo, Natella & Pietrantuono, 2010; Cotroneo et al., 2010; Wah, 2008; Grottke, 2008). Preliminary and basic software rejuvenation models were designed and they comprised figures for the transition state of system software (Wah, 2008; Huang et al, 1995). Grottke et al., (2008) suggested a fault tolerance method utilising a range of settings to lessen the impact of ageing on a system software. This study concentrated on the impact of ageing on system software that included the internal condition of the system and trends with regard to the consumption of resources, and also researched further into the appearance of bugs connected to ageing (Cotroneo, Natella & Pietrantuono, 2010; Cotroneo et al, 2010).

Please see figure 1 in the PDF version

The earliest work related to application software ageing was carried out by Parnas (1994). He suggested perspectives on software ageing compared to human ageing process. Software seemed to age in the same way as human beings with the passage of time. He claimed that software may not age in the physical sense, but under certain conditions it may gradually lose its significance and value to the environment. In such a case the software is said to be ageing (Deraman, 2009; 2010). There are two types of software ageing: 1) software ageing that is due to the failure on the part of the owner of the product to alter the product so as to enable it to adjust to various needs and a vibrant environment, and 2) software ageing that is the outcome of alterations that are made (Constantinides & Arnaoudova, 2009; Deraman, 2010). Software ageing is comparable to human ageing in terms of the following steps and causes: inactivity, ill-informed surgery and kidney failure (Parnas, 1994). At the same time an earlier study revealed that software failure today can be traced mainly to software error (40%), hardware error (15%), human error (40%) and others (5%). Software ageing is crucial in many applications software as it has been proven that the failure of the software is mainly caused by software error (Thein, 2011).

Software quality and software ageing are tightly coupled as the former may serve to determine the age of the software. The quality of the software can be maintained within a particular setting, the ageing of the software may be prolonged, while at the same time a system, procedures and factors can be put in place to sustain this. Software rejuvenation is a pre-emptive way of handling software ageing (Yahaya, Deraman & Hamdan, 2010; Yahaya & Deraman, 2010). Therefore, it is highly important and necessary to design software ageing measurement model and a rejuvenation index. In this model, the ageing factors will be used to develop a rejuvenation guide by identifying and controlling the factors that prevent ageing.

In previous years, numerous methods and techniques were developed to determine and evaluate software product quality.

Past studies revealed that the precise internal measurements that were used with complicated and large programmes in the earlier years are no longer required with the technology that is available today, while the external measurements are gaining in importance and significance. The external measurements are obtained by determining the software quality characteristics through the experiences of those who design and use the software. The external characteristics will be linked to the measurement of the internal features to enable an impartial judgment to be made of the software. Cotroneo, Natella and Pietrantuono (2010) explored the connection between constant software measurements and software ageing. In their research, software can be categorised into two well-defined groups (littleAging and bigAging) according to constant software measurements with regard to ageing principles. As this research is incomplete, further studies must be carried out to examine the effects of software ageing on software measurements.

Establishing Software Ageing Measurement Framework Using GQM

Preliminary studies reveal that some of the factors that relates to software ageing are: changing environment, operational failure, technological challenges (hardware and software), competition, commercial compatibility, etc. (Yahaya & Deraman, 2012). In order to sustain and maintain the high quality of the software through the rejuvenation process several measures should be taken including maintenance (to correct, adapt, perfect and prevent), reorganisation, repositioning and redesigning. These measures should be performed throughout the life cycle of the software or until a new system is introduced to replace the old software (Vliet, 2008).

While the software is being designed, the system requirements will most probably not remain the same because the environment and the ecosystem are constantly changing. As such, the final software that is produced will not meet its requirements. If software is to be practical it must be able to adapt to the environment at any point in its lifecycle. This is one of the rules or observations of evolution dynamics that can be applied to the issue of software ageing and its related measurement. The other rules of evolution dynamics are continuous growth, increased complexity, organisational stability and feedback system (Sommerville, 2011). These rules will be taken into consideration in order to determine the factors that affect software ageing and will be applied in the construction of the ageing model for application software.

In the same way, in software engineering and computer studies, our research group intend to observe and implement the issue of ageing in application software. Although software ageing is an innovative concept, the preliminary study, carried out earlier, has indicated that the issue is a significant one that needs to be investigated and examined. It might be beneficial to understand human ageing if it can be applied in the field of software ageing. Several factors have been identified as being related to software ageing. The preliminary study conducted by us has revealed that some significant ageing factors that have been obtained from the industry include advancement in requirements, technological challenges, commercial compatibility and consistency, design complications, deteriorating quality and changes to the environment and ecosystem. Around 30 people from various backgrounds in the industry participated in this study, which was carried out in Malaysia (Yahaya & Deraman, 2012).

GQM focuses on the data gathering and support the interpretation process (Basili & Rombach, 1988). In basic GQM, there are three levels of GQM structure. The first structure is the conceptual level or known as goal which specifies the object, purpose, quality focus, viewpoint and environment of the study. The second level is operational level that contains all sets of questions that relate to the goal. The third level is measurement level (metrics). Thus, we can see that goals contribute to the creation of several sets of questions. Generated questions also contributed to the metric to be used as measures for the questions that have been created (Gray & MacDonell, 1997).

As shown in Fig. 2, the basic GQM structure is adopted in developing software ageing measurement framework structure. At the first level, the Goal is mapped to Factor that determines the various issues affecting the software ageing process. The Question in GQM level is used to represent various objective of measurement in order to quantify the ageing factors. By setting the objective, various questions could be raised and developed in quest for the measures. At the Metric level, the proposed framework will also use the same structure to list all the possible measurable metrics that could be captured from the real environment.

Please see Figure 2 in the PDF version

Software Ageing Factors

From an extensive study on software quality, software certification and software maintenance, we have discovered that there are four main factors that may influence the age of software. The factors are realised and identified through literature study (Parnas, 1994; Vliet, 2008; Yahaya et. al, 2008; Constantinide & Arnaoudova, 2009; Cotroneo et al., 2010), expert interview and survey (Yahaya & Deraman, 2012; Yahaya et al., 2006), and brainstorming approach (Paulus & Brown, 2003; Isaksen, 1998). We have conducted series of brainstorming sessions and workshop to discuss on this issue. During the brainstorming sessions, we provide a free and open environment that encourages every member in the workshops to participate. Quirky ideas are welcomed and built upon, and all participants are encouraged to contribute fully and create solutions. The brainstorming sessions increase the richness of ideas explored, which means that we found better solutions to the problems identified.

The identified ageing factors are:

- functional,
- environment,
- human,
- product profile.

Functional factor is related to the usefulness of software. For example, if software can no longer function as it used to, the software is considered as ageing. The second factor is environment factor. Environment factor is the external factor involving accessories, alternative and the change of technology. For example, software is considered as ageing because of environment factor if it cannot accommodate the need for new technology in its environment. The third factor of software ageing is human factor. In human factor we found that the related sub factors are environment, staff, user, education, training and popularity. For example, a software is considered as ageing because of human factor if its users are using it less frequently because they prefer to use other alternative software that is more popular. Finally, the fourth factor of software ageing is product profile or contour. The aspects that should be considered in this factor are the acquisition, purchase date, produce date, technology and the age of software. For example, a software is considered as ageing related to this factor if software is originally acquired because of company policy and the users are using it less frequently because of the technology that supports the software is outdated.

The Objective

At objective level of the framework, we propose nine objectives that will serve all the four software ageing factors identified earlier. The nine objectives are shown in Table 1.

Please see table 1 in the PDF version

The Quantitative Metrics

Within the scope of software ageing and software quality issue, we have identified 27 metrics which represented by M1 to M27. All the metrics can be captured from the actual environment and easily quantified either direct or indirect measurement. These metrics are shown in Table 2.

Please see table 2 in the PDF version

The Mapping and Classification

The nine objectives mentioned above are then mapped into factor at the first level. The objectives (O1 to O9) are then broken down into several metrics at quantitative level as shown in Table

2. Table 3 shows the software ageing measurement framework discussed in this paper.

Please see table 3 in the PDF version

Conclusion

This paper has presented and discussed the issues in software ageing and the identified factors and measurements. This work was motivated from previous studies in software quality and certification which were carried out by our research group centred in Universiti Kebangsaan Malaysia. Four main ageing factors for application software have been identified, classified and mapped into nine objectives and twenty seven metrics. The basic structure, the classification and mapping are implemented using GQM approach which leads to systematic structure of the measurement framework. The proposed measurement framework of software ageing can be used in many aspects of quality measurements. For future research, we intend to measure the logical age of the software and delay the ageing by introducing the rejuvenation index which can guide the practitioners on the rejuvenation actions to be implemented.

Acknowledgement

This research is funded by Fundamental Research Grant Scheme, Ministry of Higher Education Malaysia.

References

1.Basili, V.R. and Rombach, H.D. (1988), 'The TAME Project: Towards Improvement-Oriented Software Environments,' *IEEE Transactions on Software Engineering*, 14(6), 58-773.

2. Constantinide, C. and Arnaoudova, V. (2009), 'Prolonging the Aging of a Software Systems,'
[online]. *Encyclopedia of Information Science and Technology, Second Edition.* [Retrieved January 18, 2012]. http://www.igi-global.com/viewtitlesample.aspx?id=14041.

3.Cotroneo, D., Natella, R. And Pietrantuono, R. (2010), 'Is Software Aging Related to Software Metrics?' Proceeding of the 2nd IEEE International Workshop on Software Aging and

Rejuvenation (WoSAR 2010) in conj. with International Symposium on Software Reliability Engineering (ISSRE) 2010, San Jose CA, USA.

4.Cotroneo, D., Natella, R., Pietrantuono, R. and Russo, S. (2010), 'Software Aging Analysis of the Linux Operating System,' Proceeding of IEEE 21st International Symposium on Software Reliability Engineering IEEE Computer Society Washington, DC, USA.

5.Deraman, A. (2009), 'Software Certification: The Way Forward (keynote),' Proceeding of The 5th Malaysian Software Engineering Conference (MySec2011), Johor Bharu.

6.Deraman, A. (2010), Memburu Kualiti Perisian (Inaugural Speech), UKM Publisher. ISBN 978-967-942-967-1.

7.Gray, A. and MacDonell, S.G. (1997), 'GQM++ A Full Life Cycle Framework for the Development and Implementation of Software Metric Programs,' Proceeding of ACOSM, 22-35.

8.Grottke, Jr. M., Matias, R. and Trivedi, K.S. (2008), 'The Fundamentals of Software Aging,' Proceedings of the 1st International Workshop on Software Aging and Rejuvenation, IEEE.

9.Heck, P., Klabbers, M. and Eekelen, M. (2010), 'A Software Product Certification Model,' *Software Quality Journal*, 18, 37-55.

10.Huang, Y., Kintala, C., Kolettis, N. and Fulton, N.D. (1995), ‘Software Rejuvenation: Analysis, Module and Applications,’ [online]. *IEEE.* [Retrieved Feb, 9 2012] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=466961.

11. Isaksen, S. G. (1998), ’A Review of Brainstorming Research: Six Critical Issues for Research,’ *Monograph #302*, June 1998. [Retrieved Nov, 27 2013] http://www.cpsb.com/resources/downloads/public/302-Brainstorm.pdf

12.IT Times, (2011), ‘30 Korean Software Worthy of Global Recognition in 2012,’ [online], [Retrieved January 22 , 2013], http://www.koreaittimes.com/story/15607/30-korean-software-worthy-global-recognition-2012.

13.Lee, S.W., Ghandi, R.A. and Wagle, S. (2007), 'Towards a Requirements-driven Workbench for Supporting Software Certification and Accreditation,' Proceeding of the Software Engineering for Secure Systems 2007 SESS'07 IEEE,[online]. [Retrieved February 17, 2012], http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4273334.

14.Parnas, D.L. (1994), 'Software Aging,' Proceeding of ICSE '94 The 16th International Conference on Software Engineering.

15.Paulus, P. B. & Brown, V. R. (2003), 'Enhancing Ideational Creativity in Groups,' In Paulus, P. B.; Nijstad, B. A. Group Creativity: Innovation through Collaboration. Oxford, UK: Oxford University Press, 110–136. doi:10.1093/acprof:oso/9780195147308.003.0006. ISBN 9780195147308.
Sommerville. (2011), Software Engineering, 9th edition, Pearson Education: Boston.

16.Sommerville. (2011), Software Engineering, 9th edition, Pearson Education: Boston.

17.Stanford, J. and Wallnau, K. (2001), 'Is Third Party Certification Necessary?' Proceeding of the 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction.

18.Tervonen, I. (1996), ‘Support for Quality-Based Design and Inspection,’ *IEEE Software*, 44–54.

19.Thein, T. (2011), ‘Proactive Software Rejuvenation Solution for Software Aging,’ [online], [Rretrieved February 9, 2012]. http://eurosoutheastasia-ict.org/wp-content/plugins/alcyonis-event-agenda//files/Thandar-Thein.pdf.

20.Vliet, H.V. (2008), Software Engineering: Principles and Practices, Chichester: John Wiley & Sons.

21.Voas, J. (1999), ‘Certifying Software for High Assurance Environments,’ *IEEE Software*, 22-25.

22.Wah, B. (2008), Software Aging and Rejuvenation, Wiley Encyclopedia of Computer Science and Engineering, John Wiley & Son, Inc.

23.Welzel, D. and Hausen, H. (1997), ‘Practical Concurrent Software Evaluation for Certification,’ *Journal System Software*, 38, 71-83.

24.Yahaya, J.H., Deraman, A. and Hamdan, A.R. (2006), ‘Software Quality and Certification: Perception and Practices in Malaysia,’ *Journal of ICT (JICT)*, 5(Dec), 63-82.

25.Yahaya, J.H., Deraman, A. and Hamdan, A.R. (2008), ‘Software certification model based on product quality approach,’ *Journal of Sustainability Science and Management*, 3(2), 14-29.

26.Yahaya, J.H. and Deraman, A. (2010), ‘Measuring the unmeasurable characteristics of software product quality,’ *International Journal of Advancements in Computing Technology (IJACT)*, 2(4), 95-106.

27.Yahaya, J.H., Deraman, A. and Hamdan, A.R. (2010), ‘Continuosly ensuring quality through software product certification: A case study,’ Proceedings of the International Conference on Information Society (i-Society 2010), London, UK.

28.Yahaya, J.H. and Deraman, A. (2012), ‘Towards a Study on Software Ageing for Application Software: The Influential Factors,’ *IJACT: International Journal of Advancements in Computing Technology*, 4(14), 51-59.