Towards the Anti-Ageing Model for Application Software

Jamaiah H. Yahaya and Aziz Deraman

***Abstract*— Software products do exhibit a behaviour that closely resembles human ageing. Like people, software too gets old and similar to human we can't prevent ageing, but we can understand its causes and take steps to limit its effects. As a logical product, software is not getting older physically, but in some circumstances the relevance and importance of it is getting lesser and lesser to its environment. Thus, this is the phenomena of getting older. Unlike human ageing, software ageing function can be formulated by relevant, failure, cost, technology and etc. Identifying and detecting these factors will help to rejuvenate the software and delays the ageing. In this paper, the background research in software quality and certification are discussed and explored. The previous works motivated and led us to the development of software anti-ageing model and its related areas such as ageing factors and rejuvenation index.**

***Index Terms*—Ageing factors, Software anti-ageing model, Software quality, Software evolution**

I. INTRODUCTION

In software engineering, application software is observed as exhibiting a behaviour that closely resembles human ageing. Like people, software too gets old and similar to human we can't prevent ageing, but we can understand its causes and take steps to limit its effects. Two types of software ageing by David Parnas: 1) caused by the failure of the product's to adapt with the dynamic of the environment and 2) is the result of the changes that are made [10]. As a logical product, software is not getting older physically, but in some circumstances the relevance and importance of the software is getting lesser and lesser to its environment. Thus, this is the phenomena of getting older. Unlike human ageing, software ageing function can be formulated by relevant, failure, cost, technology and etc. The way software is built and the nature that software can be modified and updated; the requirement change in dynamic environment, give flexibility and enable it to stay 'young'. This process is called software rejuvenation. Identifying and detecting these factors will help to rejuvenate the software and delays the ageing.

Previous studies have indicated that the relevant of the software at any time throughout its life span depends on the quality of the software, which is not included in current quality models and maintenance practices. Our previous research revealed that a significant contribution of the experienced ageing is due to the external quality measurement. The main objective of this research is to develop a software anti-ageing model based on software evolution dynamic environment.

Manuscript received March 6, 2012; revised April 14, 2012. This work is supported in part by the Malaysia Ministry of Higher Education under the Fundamental Research Grant Scheme (FRGS/1/2012/SG05/UKM/02/10).

Jamaiah H. Yahaya is with the School of Computer Science, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Bangi, 43600, Selangor, Malaysia (phone: 60389216181; fax: 60389256732; e-mail: jhy@ftsm.ukm.my).

Aziz Deraman is with the School of Computer Science, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Bangi, 43600, Selangor, Malaysia (e-mail: ad@ftsm.ukm.my).

II. RESEARCH BACKGROUND

In software engineering, software ageing refers to progressive performance degradation or the state of the software degrades with time. However, time is not the absolute factor in software ageing. Software ageing is very closely related to the quality of the software and whether the particular software can maintain its quality through-out its life cycle. The quality status of the software might degraded overtime and a rejuvenation processes are needed to improve the software and maintain the quality as well as the ageing factors.

One approach for managing ageing of a software is through certification. Software certification process can be applied at anytime during its life span, measure its quality and thus can anticipated the decay of quality [5].

A. Software Quality

Software product quality can be evaluated via three categories of evaluations: internal measures, external measures and quality in use measures [11][16]. Internal measuring is the evaluation based on internal attributes typically static measures of intermediate products and artifacts and external measuring is based on external attributes typically measuring the behaviour of the code when executed. While the quality in use measures includes the basic set of quality in use characteristic that affect the software. This characteristic includes effectiveness, productivity, safety and satisfaction. This measurement is an on-going research of SQuaRE which is the next generation of ISO 9126 but yet not fully published currently. SQuaRE quality model consists of internal and external measures that include quality in use aspects. It presents similar concept of characteristics and subcharacteristics as in ISO 9126 approach [17. Several studies in software quality have reveal the important of

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

software measurements that relies on software external metrics [6][17][18].

If we looked back to previous software quality models namely were the McCall model (1977), Boehm model (1978), FURPS model (1987), ISO9126 (1991), Dromey model (1996), Systemic Quality model (2003), and PQF model (2007), the software quality characteristics found in most models were: efficiency, reliability, maintainability, protability, usability and functionality. In PQF model the new human aspects in measuring quality was included which was not introduced in previous models. Measuring quality based on human aspects which relates to user's perspective and expectations are demanding today[6].

Self-assessment or user centric assessment approach is increasing demanded in software engineering principle and practice. In this environment users are able to assess the software products and artifacts within their own environment. User-centric can be defined as a process in which the requirements and limitations of end-users of a software product are given extensive attention at each stage of the processes. The requirements todays demand the quality model to be simple, specific and practical to be measured by layman, users, customers, developers or stakeholders. This relates back with the general definition of quality, quality is defined as "fitness for use" and "conformance to requirements". The term "fitness of use" usually means characteristics such as functionality, usability, maintainability, and reusability and "conformance to requirements" means that software has value to the users [4][20]. Previous quality models do not accommodate and focus to this requirement.

B. Software Certification Process

The term certification in general is the process of verifying a property value associated with something, and providing a certificate to be used as proof of validity. A software certification is defined by Jeffry Voas [21] as a fact sheet that spells out known software output behaviours (and it could also spell out known internal behaviours. Stanfford and Wallnau (2001) [24] define certification as a process of verifying a property value associated with something, and providing a certificate to be used as proof of validity. It is a new concept in software engineering but increasingly acceptable to the software industry. Korea is among the most leading country that requires certification of software product in their industry. The quality certification program is called Good Software and was implemented for the last 10 years [23]. Malaysia is in the phase of developing Malaysian Certification Program which involve several groups that consists of professional and academician from different organisations and industries. Results from certification will contribute a valuable recognition on the quality of the software orgnanisation which can support the buoyancy and trustworthiness of the organisation.

Many approaches to software certification mostly rely on formal verification, expert reviews, developer assessment and software metrics to determine the product quality as described in Welzel and Hausen [22], Voas [21], Lee, Ghandi & Wagle [30] and Heck, Klabbers & Eekelen [18]. Another approach is by integrating ISO9126 model as the certification quality benchmark. Example is Good Software [23], Requirement-driven Workbench [30] and SCM-Prod [28]. These models are suitable for general software assessment with static attributes such as portability, usability, reliability, maintainability, functionality and efficiency. Different approach for certification is using function point [31]. Function point is a standard metric for the relative size and complexity of a software system, originally developed by the IBM in the late 1970s. Most of the studies mentioned above focused on certify software artifacts from develops, suppliers and auditors perspectives and do not emphasis much on user's perspective and involvement.

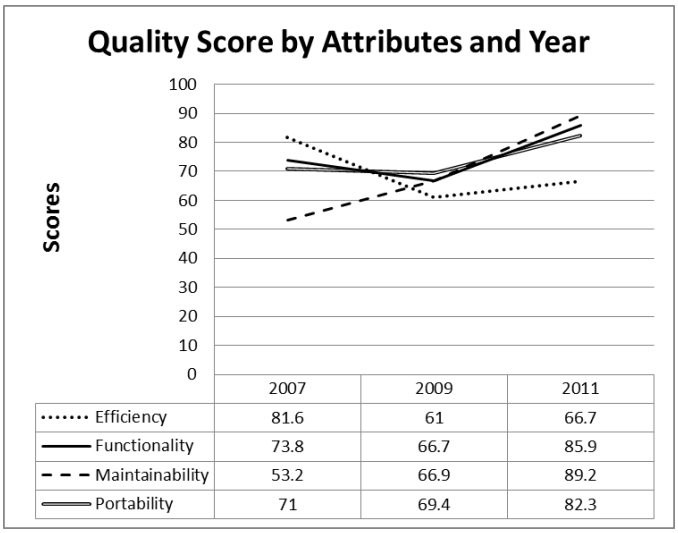
Many related researches have been actively conducted in The National University of Malaysia or UKM in the area of software certification and quality [25][26][27]. Azrina et. al (2001) started their works in software certification by investigated certification issues and perception by Malaysian organisations [25]. This work was then continued by Jamaiah et. al [6][27][28] and Fauziah et. al [27][29] with the development of software certification framework. The framework consists of two distinct certification models: SPAC and SCM-Prod model. The main objective of these fundamental researches was to develop software certification models that could provide quality assurance and warranty to the users on the status of software products and artifacts at any time during its lifecycle. SPAC is a certification model based on process approach while SCM-prod model is a certification model that focuses on end-product quality approach. SCM-Prod model has a balance quality measurement by integrating technical aspects of software together with the human aspects in the quality assessment model. The quality model is known as Pragmatic Quality Factor (PQF). The certification model, SCM-prod model, has been tested and implemented in real environment. It has been verified for practical working model and proved by case studies [3][5].

Our previous experience in software product certification exercises have demonstrated that software certification by product quality approach using our model, SCM-prod model, can be used as a quality control mechnism throughout the life cycle of the product. At any stage of its life cycle the quality of the software can be measured and thus will help the practitioner and developer to prevent the software of getting less important and less quality. If the software is getting less important and less quality in the environment, it is said as moving to the ageing stage of its life cycle. Fig. 1 and Fig. 2 show examples of score obtained in the certification exercise done in one case study carried out collaboratively with one large organisation in Malaysia.

The charts in Fig.1 and Fig. 2 show that the quality of software products can be measured and transformed into a more meaningful presentation. With the line chart presentation, the quality can be monitored regularly at some time intervals. The application software applied in this exercise was a large Hospital Information System which was developed by in-house developers and expertises, and already under operationed for 5 years in the environment. Therefore, we can observe that the quality of the software is at higher level in

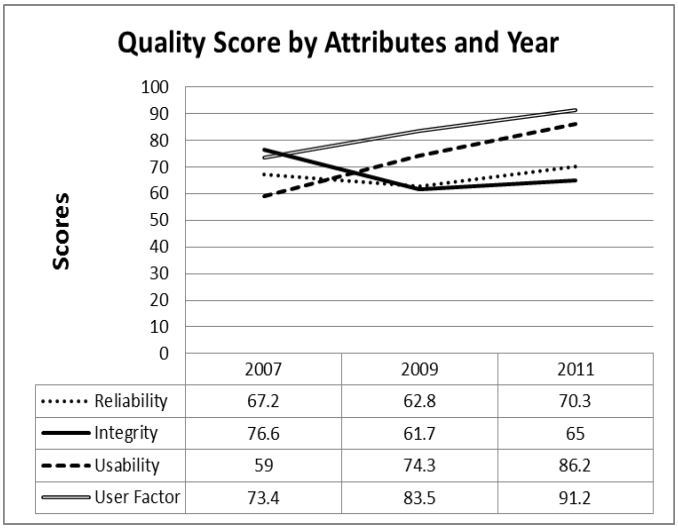
ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

year 2011 compares to the previous assessments in year 2007 and 2009. In the near future, the performance in term of the quality of this software can be monitored again. This will help the developer and the stakeholder to ensure that their product is at high level of quality based technical and user perceptions that included in the PQF quality benchmark used in the certification process.



	2007	2009	2011
Efficiency	81.6	61	66.7
Functionality	73.8	66.7	85.9
Maintainability	53.2	66.9	89.2
Portability	71	69.4	82.3

Fig. 1. Quality score by attributes (Efficiency, Functionality & Maintainability) in years 2007,2009 and 2011



	2007	2009	2011
Reliability	67.2	62.8	70.3
Integrity	76.6	61.7	65
Usability	59	74.3	86.2
User Factor	73.4	83.5	91.2

Fig. 2. Quality score by attributes (Reliability, Integrity, Usability & User Factor) in years 2007,2009 and 2011

III. Software Ageing Issues

Previous studies on software ageing phenomenon dealt with the degradation of the exhaustion of operating system resources, data corruption, and numerical error accumulation [1][2]. Examples of software ageing in this context are memory bloating and leaking, unreleased file-locks, storage space and etc. This work relates to analysis of software ageing in the Linux OS kernel. Several previous studies in software ageing and rejuvenation approach focus on operating systems, systems software and hardware[1][2][8][9]. Huang et.al developed the basic software rejuvenation model which included the state transition diagram for software system [2][15]. Grottke et al. proposed the fault tolerance technique using environmental diversity to mitigate the ageing effect of a system software [9]. This research too focused on the ageing effect on system software that involved internal system state and resource consumption trends and further studied on the manifestation of ageing-related bugs [12].

Other approach and view on software ageing was proposed as ageing phenomena similar to human ageing. Software behaves similar to human being where it gets older and older as the time passes by[10]. Thus, we can say that human ageing factors can be manifested in time or years. Even though software is not getting older physically, but in some circumtances the relevant and important of the software is getting lesser and lesser to its environment. Thus, we can say that its' getting older [3][4]. Two types of software ageing can be categorised as the caused by the failure of the product's owner to modify it to meet changing needs and dynamic environment; and secondly is the result of the changes that are made [10][13]. Furthermore, certain measures and causes of software ageing similar to human ageing according to Parnas(1994) are: lack of movement, ignorant surgery and kidney failure. Whilst the causes of software failure today are more toward software error (40%), hardware error (15%) , human errors (40%) and others (5%). Software ageing phenomena mentioned earlier becomes critical in most applications software [8] because the finding shows that the software failure comes mostly from the software faults. Few major causes that may cause ageing are dealing with the quality of documentation, design for change in software code, unimposing standard, cost, technology, environment and etc. The limitation here is that there is no such software anti-ageing model for application software which can guide the software owners and developers on the ageing prevention.

Software ageing and software quality are closely related. Ensuring continuous quality of software operating in certain environment may delay the ageing of the software [5] and at the same time a mechanism, guidelines and indicators are needed to support this prevention. A proactive method to deal with software ageing phenomenon is software rejuvenation. Thus, development of software anti-ageing model and rejuvenation are very significant and demanded. The anti-ageing model will become the input to the rejuvenation index construction by formulating and regulating the anti-ageing factors.

Several software quality assessment and measurements methods and techniques have been introduced since earlier years of software development. The direct internal measurements used in those years such as compexity and large size of the program are no longer necessities in the current technology while the external measurements are getting more important and relevant [3][6]. The external measurements are related to measuring software quality attributes through the users and developers experience the software. They are subjective factors and the measurement of internal attributes of the software that relates to the external attributes will enable the objective judgement about these attributes. The

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

relationship between software static metrics and software ageing were investigated and found the encouraging result as discussed in section 2. Furthermore, Cotroneo, Natella and Pietrantuono (2011) discovered that the software applications can be categorised into two distinct groups (littleAging and bigAging) based on software static metrics toward ageing criteria [12]. The limitation of this work is that it still requires further investigations on the software ageing effects related to software metrics.

IV. THE AGEING FACTORS

The initial literature study has discovered that some of the factors that may cause software ageing are: environment change, functional failure, technology challenges (hardware and software), competition, business compatibility, process ontology change and etc. The actions needed to maintain and keep the high quality of the software via rejuvenation process includes the maintenance activities (corrective, adaptive, perfective, preventive), restructuring, reallignment, redesign and etc [14]. These actions will be carried out continuosly till the software life ends and terminates, and replace with a new system.

The system requirements are likely to change while the software is being developed because the environment is changing. Therefore, a delivered software won't meet its requirements. Software must be changed if they are useful in an environment. This is one of the law or observation according to evolution dynamic that can be used as the software ageing phenomena and its associated measurement. The other evolution dynamic laws are continuing growth, increasing complexity, organisational stability and feedback system [11]. These will be considered in identifying factors that influence software ageing and use as the formation of anti-ageing model for application software.

The human ageing is defined as "the gradual changes in the structure and function of humans and animals that occur with the passage of time, that do not result from disease or other gross accidents, and that eventually lead to the increased probability of death as the person or animal grows older" [19]. In human ageing, the ageing function can be formulated as:

Human Ageing =
f(time, physical, psychological, social change)

Similarly, the ageing in human can be categorised into three levels: young old (age 65-74), middle old (age 75-84) and oldest old (age 85+). In [31] discusses the first formal studies of human ageing, which appear to be those of Muhammad ibn Yusuf al-Harawi (1582) in his book Ainul Hayat, published by Ibn Sina Academy of Medieval Medicine and Sciences. The original manuscript of Ainul Hayat was scribed in 1532 by the author Muhammad ibn Yusuf al-Harawi and was translated by Hakim Syed Zillur Rahman (2007). The book discusses behavioural and lifestyle factors putatively influencing ageing including diet, environment and housing conditions [32] Also discussed are drugs that may increase and decrease ageing rates. In recent years, studies are being conducted to investigate the ageing factors and study the rejuvenation activities to delay the human ageing.

Application software ageing is still a new innovation topic. Understanding the human ageing might useful to explore it into a new software domain. The ageing function of software can be formulated as:

Software Ageing = f(Requirment evolution, failure, cost, business compatibility, environment dynamic ...)

Various factors have been recognised as the software ageing factors. Table 1 shows the software ageing factors for application software and the prevention actions. The actions listed in the table are the possible activity to prevent the software from reaching the ageing phenomena and the actions can be considered as rejuvenation activity of the software. The rejevenation actions will ensure the software stay young and healthy.

TABLE 1
SOFTWARE AGEING FACTORS AND PREVENTION ACTIONS

Ageing Factors	**Environment dynamic** **Failure** **Technology challenges (Hardware and Software)** **Competition** **Business compatibility & stability** **Requirement evolution** **High cost** **Ontology change** **Declining quality** **Feedback system** **Increasing complexity**
Prevention Actions	Corrective Perfective Adaptive Restructuring Redesign Reallignment Redeployment

More works need to be conducted to identify factors that contributes to software ageing. Correlation between the factors might be needed to identify the main and vital factors as well as to prioritise them. Our research group will continue and enhance this reseach to develop the anti-ageing model and further to deliver a rejuvenation index based on the proposed idea in this paper.

V. CONCLUSIONS

The initial works related to software ageing has been presented in this paper. Our previous research experiences in software quality and certification have motivated us to investigate and explore further on the issues of software ageing. Analysis of certification and quality data gathered from year 2007 till 2011 has demonstrated that quality can be monitored continuously through software metrics. The symptoms of degradation in term of quality of software operating in certain

ISBN: 978-988-19252-1-3
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

environment can be observed to prevent or delay the ageing. The potential ageing factors were presented and further study and exploration are needed to confirm the correlation of the factors and develop the anti-ageing model.

REFERENCES

[1] D. Cotroneo, R. Natella, R. Pietrantuono and S. Russo, "Software aging analysis of the Linux operating system," in *Proc. IEEE 21st International Symposium on Software Reliability Engineering IEEE Computer Society*, Washington, DC, USA, 2010.

[2] B. Wah, *Software aging and rejuvenation*. Wiley Encyclopedia of Computer Science and Engineering, John Wiley & Son, Inc, 2008.

[3] A. Deraman. *Memburu kualiti perisian (Inaugural Speech)*. UKM Publisher. ISBN 978-967-942-967-1.A. 2010.

[4] A.Deraman, "Software certification: The way forward (keynote)", in *Proc. The 5th Malaysian Software Engineering Conference (MySec2011)*, Johor Bharu., 2011.

[5] J.H. Yahaya, A. Deraman and A. R. Hamdan, "Continuosly ensuring quality through software product certification: A case study," in *Proc. The International Conference on Information Society (i-Society 2010)*, London, UK, 28-30 June 2010.

[6] J.H. Yahaya and A. Deraman, "Measuring the unmeasurable characteristics of software product quality," *International Journal of Advancements in Computing Technology (IJACT)* vol.2, no. 4, pp. 95-106, 2010.

[7] P.J. Denning. "What is software quality?" *ACommentary from Communications of ACM*, January 1992.

[8] T. Thein. (2011). "Proactive software rejuvenation solution for software aging," [Online]. Available: http://eurosoutheastasia-ict.org/wp-content/plugins/alcyonis-event-agenda//files/Thandar-Thein.pdf

[9] M. Grottke, R. Matias Jr. and K. S. Trivedi. "The Fundamentals of Software Aging", in *Proc. The 1st International Workshop on Software Aging and Rejuvenation, IEEE*, 2008.

[10] D. L. Parnas. (1994). "Software aging", in Proc. *ICSE '94 Proceedings of the 16th international conference on Software engineering, ACM DL* [Online]. Available: http://dl.acm.org/citation.cfm?id=257788

[11] Sommerville, *Software Engineering*, 9th edition, Pearson Education, Boston, USA, 2011.

[12] D. Cotroneo, R. Natella and R. Pietrantuono. "Is software aging related to software metrics?" in *Proc. The 2st 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, November 2010.

[13] C. Constantinides and V. Arnaoudova. (2009). " Prolonging the aging of aoftware systems," *Encyclopedia of Information Science and Technology* [Online]. *Second Edition (8 Volumes)*. Available: http://www.igi-global.com/viewtitlesample.aspx?id=14041

[14] H. V. Vliet, *Software engineering: principles and practices*. third edition. Chichester: John Wiley & Sons, 2008.

[15] Y. Huang, C.Kintala, N. Kolettis and N.D. Fulton. (1995). "Software rejuvenation: Analysis, module and applications," *IEEE* [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=466961

[16] W. Suryn, A. Abran, P. Bourque and C. Laporte, "Software Product Quality Practices: Quality Measurement and Evaluation Using TL9000 and ISO/IEC9126," in *Proc. The 10th International Workshop, Software Technology and Engineering Practice (STEP)*, 2002.

[17] W. Suryn, A. Abran and A. April. (2003). ISO/IEC SQuaRE: The Second Generation of Standards for Software Product Quality [Online], Available: http://www.lrgl.uqam.ca/publications/pdf/799.pdf

[18] P. Heck, M. Klabbers and M. Eekelen, "A Software Product Certification Model", *Software Quality Journal* vol. 18, pp. 37-55, 2010.

[19] Biology-online (2005). "Aging" [Online], Available: http://www.biology-online.org/dictionary/Aging

[20] I. Tervonen, "Support for Quality-Based Design and Inspection", *IEEE Software*, pp. 44–54, January 1996

[21] J. Voas, "Certifying software for high assurance environments", *IEEE Software*, pp. 22-25, July/August 1999.

[22] D. Welzel and H. Hausen, "Practical concurrent software evaluation for certification," *Journal System Software*, vol. 38, pp. 71-83, 1997.

[23] IT Times. (2011). "30 Korean Software Worthy of Global Recognition in 2012" [Online]. Available: http://www.koreaittimes.com/story/15607/30-korean-software-worthy-global-recognition-2012

[24] J. Stanford and K. Wallnau, "Is third party certification necessary?" in *Proc. The 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction*, 2001.

[25] A. Kamaruddin, A. Deraman, S. Yahya, H. Selamat and H. Zulzalil, "Perception on Software Certification Model: An Empirical Study," in *Proc. The International Conference on Information Technology and Multimedia (ICIMU, 2001)*, Uniten, Malaysia, pp.464-469, August 2001.

[26] A. F. Amalina Farhi and A. Deraman, "Measuring the usability of software applications: metrics for behaviorness," in *Proc. The 2007 International Conference in Computational Science and Its Applications*, vol. Part II, Springer-Verleg Berlin, Heidelberg, 2007.

[27] J. H. Yahaya, F. Baharom, A. Deraman and A. R. Hamdan, "Software Certification from Process and Product Perspectives", *International Journal of Computer Science and Network Security*, vol. 9, no. 3, March 30 (2009). ISSN: 1738-7906.

[28] J. H. Yahaya, A. Deraman and A. R. Hamdan. "Software certification model based on product quality approach," *Journal of Sustainability Science and Management*, vol. 3, No. 2, pp. 14-29. Dec. 2008, ISSN: 1985-3629.

[29] F. Baharom, J. H. Yahaya, A. Deraman and A. R. Hamdan, "SPQF: Software process quality factor for software process assessment and certification," in *Proc. The IEEE International Conference on Electrical Engineering and Informatics*, Bandung, Indonesia, 17-19 July 2011.

[30] S.W. Lee, R.A. Ghandi and S. Wagle. (2007). "Towards a requirements-driven workbench for supporting software certification and accreditation", *Software engineering for secure systems, 2007. SESS'07:ICSE Workshops 2007. Third International Workshop. IEEE* [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4273334

[31] IFPUG. (2008). "International Function Points Users Group" [Online]. Available: http://www.ifpug.org/certification/software.htm

[32] Wikipedia. (2012), "Ageing" [Online] Available: http://en.wikipedia.org/wiki/Ageing