ORIGINAL RESEARCH ARTICLE UHD JOURNAL OF SCIENCE AND TECHNOLOGY

A Simple Software Rejuvenation Framework Based on Model Driven Development

Hoger Mahmud

Department of Computer Science, College of Science and Technology, University of Human Development, Iraq

ABSTRACT

In the current dynamic-natured business environment, it is inevitable that today's software systems may not be suitable for tomorrow's business challenges which indicate that the software in use has aged. Although we cannot prevent software aging, we can try to prolong the aging process of software so that it can be used for longer. In this paper, we outline a conceptual software rejuvenation framework based on model **driven** development approach. The framework is simple but effective and can be implemented in a recursive five step process. We have illustrated the applicability of the framework using a simple business case study which highlights the effectiveness of the framework. This work adds to the existing literature on software aging and its preventative measures. It also fills in the research gap which exists about software aging caused by changing requirements.

Index Terms: Model Driven Development, Software Aging, Software Rejuvenation Framework

1. INTRODUCTION

In this paper, we propose a rejuvenation framework that addresses software aging on abstract level. The changing world demands faster and better alignment of software systems with business requirements to cope with the rising demand for better and faster services. This simply means that a perfectly untouched functioning software ages just because it has not been touched. The aging phenomenon occurs in software products in similar ways to human; Parnas [1] draw correlations between the aging symptoms in human and software. As demands for functionality grow software complexity rises, and as a result software, underperformance and malfunctioning became apparent [2]. Software aging is a known phenomenon with recognized symptoms such as increase in failure rate [3]. Researchers have identified a number of causes of software aging, for example, accumulation of errors over time during system operation. One other cause is "weight gain" as in human, software gains weight as more codes are added to an application to accommodate new functionalities, and consequently, the system loses performance. There are numerous examples where software aging has caused electronic accidents in complex systems such as in billing and telecommunication switching systems [4]. Beside the causes researchers in the field have identified a number of aging indicators such as increased rate of resource (e.g., memory) consumption [5]. Another aging indicator is how robust a system is against security attacks if observed over time. This is because security attack techniques are becoming more sophisticated by day. However, more is needed to be done to address the aging phenomena. Grottke *et al.* [5] claim that the conceptual aspect of software aging has not been paid adequate attention by researchers to cover the fundamentals of software aging.

Currently, addressing software aging is mostly done using reengineering techniques such as:

Access this article online

DOI: 10.21928/uhdjst.v1n2y2017.pp37-45

E-ISSN: 2521-4217
P-ISSN: 2521-4209

Corresponding author's e-mail: hoger.mahmud@uhd.edu.iq

Received: 02-07-2017 Accepted: 22-08-2017 Published: 30-08-2017

1. Forward engineering concerns with moving from high-level abstraction to physical implementation of a system
2. Reverse engineering concerns with analyzing a system to identify components and connectors of that system to represent the system in a different form or higher level of abstraction
3. Redocumentation deals with creation or revision of semantically equivalent representation within the same abstract level
4. Design recovery concerns with reproducing all required information about a system so that a person can understand what the program does
5. Restructuring concerns with transforming a representation of a system to a different one, without any modification to the functionality of the system.

Reengineering can facilitate the examination of a system and learn more about it so that appropriate changes can be made. However, it is not the ideal solution for software upgrade as the process is extremely time-consuming and resource expensive. In this paper, we present a conceptual software rejuvenation framework based on model driven development (MDD) techniques capable of addressing software aging with less time and resource. The framework is most effective where the software aging is due to changing business requirements which in effect requires the addition or omission of functionalities. We have illustrated the applicability of the framework through a simple business case study which supports the effectiveness of the framework. This work contributes to the field of software aging by presenting a novel conceptual framework to software developers that can be utilized to dilute software aging.

The rest of this paper is organised as follows, in Section 2 we provide a brief background about software aging and rejuvenation and in Section 3 we present some related works. In Section 4, we outline the frame work and in Section 5, we illustrate the applicability of the framework using a simple business case study. In Section 6 and 7, we discuss, conclude, and provide some recommendations.

2. BACKGROUND

In this section, we provide a brief background to both software aging and software rejuvenation with the aim to provide better understanding of the proposed framework later in Section 4.

A. Software Aging

Software aging was first introduced by Huang *et al.* [6] and since then the interest in the topic has risen among academics and industries. Complex systems rely on an intricate architectural setup to function, if the structure is slowly destroyed by maintaining and updating the system software aging becomes inevitable [7]. It is a known fact that a system maintainer can mess up perfectly fine functioning software through changing codes or inserting incorrect codes which is known as "ignorant injection" [8]. To provide a focus view of research areas on software aging Cotroneo *et al.* [9] have analyzed more than 70 papers in which they have concluded that overall there are two major categories of research into understanding software aging the first is model-based analysis and the second is measurement-based analysis. Several measureable techniques have been proposed to detect software aging such as "aging indicators" and "time series analysis." The techniques are used to collect data about resources used in a system, and then, analyze it to see if the consumption rate has increased over time which is a sign of aging [3]. As for the causes of software aging, there are two major classes, the first is known as "ignorant surgery" and the second is known as "lack of movement." Fig. 1 shows the major contributors to the two classes of software aging causes.

B. Software Rejuvenation

To keep critical systems functioning correctly software rejuvenation is recognized as an effective technique [10]. The objective of software rejuvenation is to rollback a system continuously to maintain the normal operation of the system and prevent failures. According to Cotroneo *et al.* [3] application-specific and application-generic are two main classes of software rejuvenation techniques in which the former works on specific system features and the latter works on the whole system (e.g., system restart).

To further elaborate on the two main classes, researchers have provided a number of examples for both; flushing of kernel, file system defragmentation and resource reprioritization are examples of application specific rejuvenation and application restart, cluster failover, and operating system reboot are examples of application generic rejuvenation [3]. Fig. 2 illustrates the two classes of software rejuvenation techniques.

3. RELATED WORK

There have been a number of attempts to tackle software aging similar to what we propose here. The authors of Huang *et al.* [6] present a model-based rejuvenation approach for billing applications and Okamura and Dohi [10] proposes dynamic software rejuvenation policies by extending models presented in Pfening *et al.* [11]. The approach is case

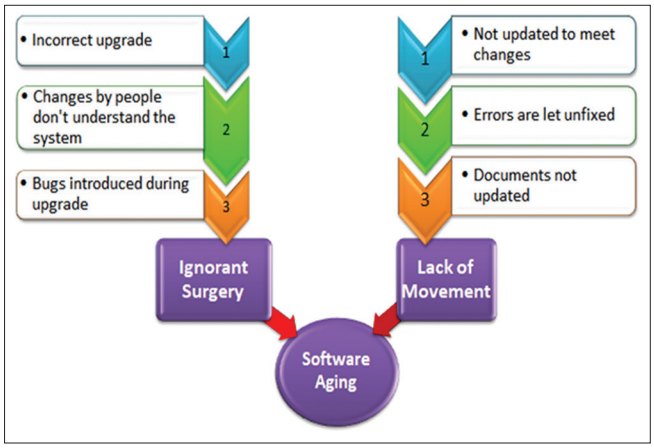

Fig. 1. Major causes of software aging

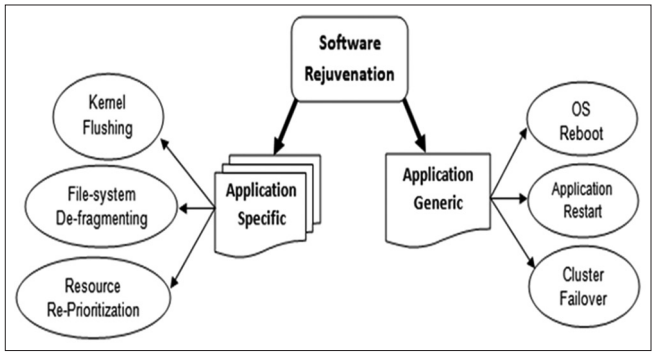

Fig. 2. Software rejuvenation techniques

specific and cannot be applied to a domain; this, however, has similarities with what we are proposing since they also use models to rejuvenate software. Saravakos *et al.* [12] proposes the use of continuous time Markov chains to model and analyze software aging and rejuvenation to better understand causes of aging which helps putting in place mitigating measures. This approach is suitable to treat symptoms of aging that happens for technical reasons rather than changes in requirements. Dohi *et al.* [13] models optimal rejuvenation schedule using semi-Markov processes to maximize availability and minimize cost. The focus here is aging caused due to processing attributes; however, unlike this work we focus on the functionality attributes of a system Garg *et al.* [14]. Adopts the periodic rejuvenation technique proposed by Huang *et al.* [6] and uses stochastic petri net to model stochastic behavior of software aging. Beside modeling techniques, others have used techniques such a time triggered rejuvenation technique used by Salfner and Wolter [15] and software life-extension technique used by Machida *et al.* [16] to counteract software aging in which they take preventative measures to ease software aging and allow more time for system rethink. Huang *et al.* [6] proposes a proactive technique to counteract software aging with the aim to prevent failure using periodic preemptive rollback of running applications. To detect symptoms of aging techniques such as machine learning is used to analyze data through adopting artificial intelligent algorithms (e.g., classifiers) [17]. Garg *et al.* [18] discuss measures for software aging symptom detection with the aim to diagnose and treat the aging taking place, others have used pattern recognition techniques to detect aging symptoms [17]. These works propose how to detect symptoms of software aging without proposing a suitable mechanism to treat the symptoms.

All the related works presented so fare address software aging from technical and performance viewpoint and none consider aging caused as a result of changing requirements. This allows us to claim that our framework contributes to the software aging and rejuvenation literature by filling in this gap and take a new direction in tackling software aging.

4. FRAMEWORK OUTLINE

The base of our conceptual rejuvenation framework is MDD technique [19], [20]. France *et al.* [21] claim that abstract design languages and high-level programming languages can provide automated support for software developers in terms of solution road map that fast-forward system developments. Following their direction we use Model Driven Development (MDD) techniques to design a rejuvenation framework to tackle requirement-based software aging. MDD simply means constructing a model of the system with fine details before transferring it into code. It provides the mapping functions between different models for integration and model reusing purposes [22]. MDD is a generic framework that can accommodate both application specific and application generic classes of software rejuvenation. Mayer *et al.* [23] states MDD is ideal for visualizing systems and not losing the semantic link between different components of the system at the same time. It is inevitable that extensive manual coding in developing a system escalates human errors in the system; this issue can be addressed through code automation which is the ultimate aim of MMD. Building and rebuilding system is an expensive process that requires time and resource; model driven aims at using, weaving and extending models to maintain, develop and redevelop systems. Experts in the field claim that MDD improves quality as models are continuously refined and reduce costs by automating the development process [22]. This process changes models from being expenses to

important assets for businesses. Researchers have identified the conceptual gap between problem domain and implementation as a major obstacle in the way of developing complex systems. Models have been utilized to bridge the gap between problem domain abstractions and software implementation through code generation tools and automated development support [2]. Models can serve many purposes such as:

1. Simplifying the concept of a complex system to aid better understanding of the problem domain and system transformation to a form that can be analyzed mechanically [24]
2. Models are platform and language independent
3. Automatic code generation using models reduce human errors
4. For new requirements only the change in model is required this reduces the issue explained previously known as weight gain.

A. Framework Steps

We propose a five step recursive software rejuvenation framework to address the issue of software aging. As mentioned the framework is based on model driven software development which is implemented in the following steps:

1. First developers gather system requirements which is one of the must do tasks in every software development
2. Developers design the entire system in great details using tools such as unified modeling language (UML)
3. The complete design is fed into code generators such as Code Cooker (http://codecooker.net) and Eclipse UML to Java Generator to generate system codes
4. Software codes are integrated, tested, and finalised, this step is necessary since a code generator tool capable of generating 100% of the code is yet to exist. This limitation is discussed in Section 6
5. In the final step where the new product is delivered and installed.

Fig. 3 illustrates the five steps explained in a recursive setting, i.e., when a new feature is required to be added to the system to address a new requirement the system is upgraded through the model rather than though code injection. The models are kept as assets and refined as new requirements come in, the next section provide more inside as to how the framework works.

5. CASE STUDY

To illustrate the applicability of the framework we present a simple none-trivial business case study specific to Kurdistan region.

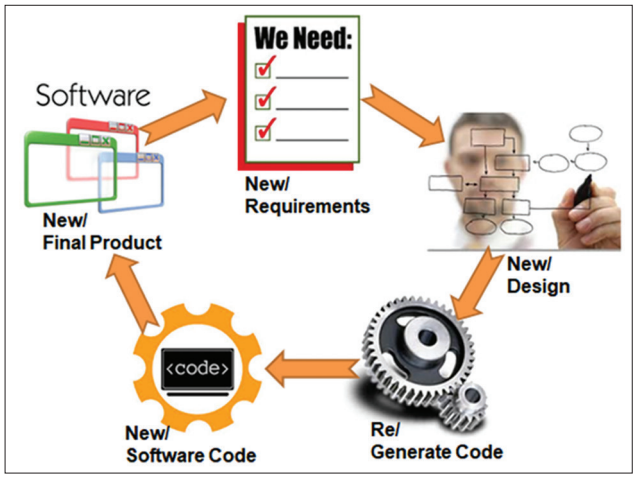

Fig. 3. Model driven development-based software rejuvenation framework

Mr. X is a supermarket owner in the city of Sulaymaniyah who sells domestic goods and he employs 10 people in his supermarket. Currently, his shop is equipped with electronic point of sale (ePOS) systems to record transactions and the form of payment by customers is cash only. Electronic payment is not feasible due to unavailability of electronic payment systems in the region's banks. His current ePOS system is capable of performing the following functionalities:

1. Store individual item details such as name, price, barcode, and expiry dates
2. Store information about employees such as name, address, date of birth, and telephone numbers
3. Retrieve and match barcodes on products to display and record item details
4. Calculate total price and print out customer receipts
5. Record all transactions and generate various reports such as daily sales report, weekly sales report, and sale by item report
6. The administration side of the system is managed through a user management subsystem which allows adding, deleting, updating, and searching on users. The system also contains a product management subsystem that allows managing products through adding, deleting, updating, and searching on item.

We make an assumption that in the next 6 months electronic payment systems (ePayment) will become available in Kurdistan for businesses to use. Now Mr. X would like to gain an edge over his competitors and add ePayment system to his current ePOS system. Fig. 4 is the UML use case diagram for the current ePOS system in Mr. X's supermarket which shows the use cases than can be performed by each actor.

Fig. 5 is the future use case diagram for the new system which shows the addition of a new actor called "customer" and a new use case called "pay electronically" coted in yellow. Now, we assume developers of the system had the framework in mind when they developed the system and have kept a design model of the system similar to the one illustrated in Fig. 6 which shows a UML class diagram design model of the ePOS system. Mr. X now goes back to them and request that the new functionality (electronic payment) to be added to the system. Using the framework the developers refine the UML class diagram model (new classes coted in yellow) to accommodate the new requirement and produce a new design similar to the one shown in Fig. 7. The new design is now ready to be fed into code generators to generate the codes for the new system. Using the framework the developers have performed a rejuvenation process on Mr. X's system without touching

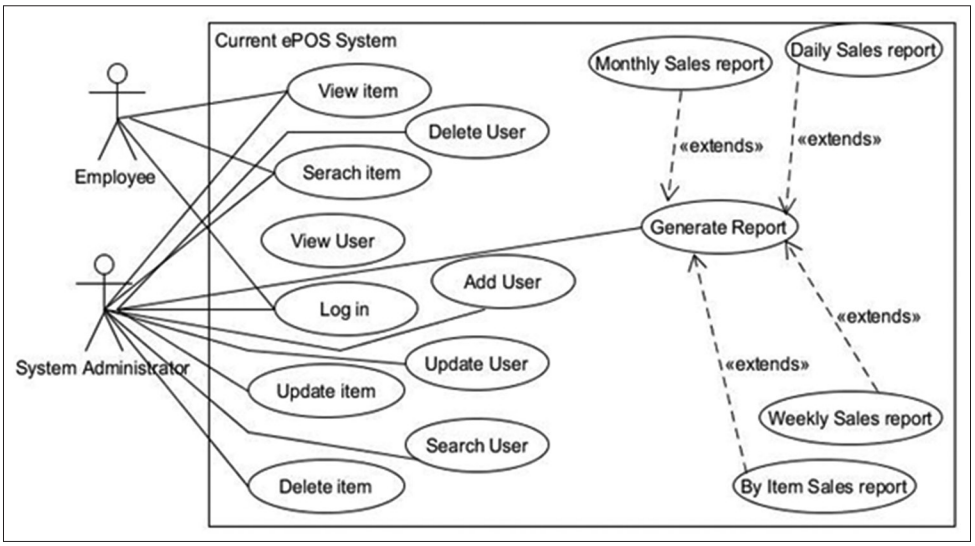

Fig. 4. Current electronic point of sale unified modeling language use case diagram

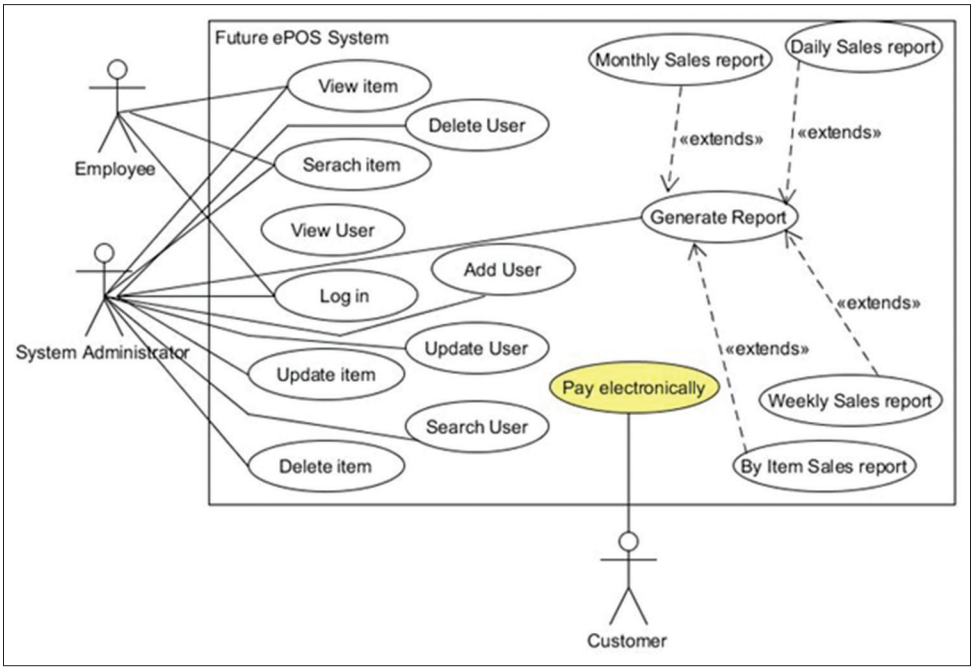

Fig. 5. Future electronic point of sale unified modeling language use case diagram

the current operating ePOS. It is important to point out that the framework tackles software rejuvenation conceptually and on abstract level which means we are bypassing all the technicalities of implementation and testing processes. During our search, we did not come across any related work that considers design for software rejuvenation rather than an actual system, which indicates that our approach is unique. However, it has to be said that although being unique is an advantage, it has made it difficult for us to compare the applicability of our framework with other existing frameworks.

6. DISCUSSION

Researchers in the field have concluded that software aging is inevitable and as software ages it loses its ability to keep up. In this paper, we have proposed a five step recursive software rejuvenation framework based on model driven software development approach. To illustrate the applicability of the framework we have outlined a simple business scenario and explained how the framework rejuvenates the current system in use by the business. The framework will provide the following advantages over existing rejuvenation techniques:

1. The model is used to redevelop the system without taking the old system out of operation which leads to reduction in down time (unavailability) which otherwise lead to lose of customers and profits
2. Using models to maintain and update software gives the development process an edge as models are language independent and can be used to develop systems in the state of the art programming languages which in turn ease software aging as the technology used in the development is current [25]
3. As codes are generated automatically human errors are reduced, which is one of the contributors of software aging
4. Redevelopment costs and times are reduced as developments are automated.

The objective of software rejuvenation is to rollback a system continuously to maintain the normal operation of the system and prevent failures. However, software rejuvenation increases system downtime as the system is taken out of operation while the rejuvenation process is performed. Knowing when to perform rejuvenation process on a system is a crucial factor recognized by researchers to minimize cost and maximize availability [3]. The framework we have proposed addresses this issue by working on the system on design level without terminating the system operation while the rejuvenation solution is finalized. It is important to stress that the framework is conceptual and requires further research as there are a number of limitations that need be addressed to make the framework fully applicable. The limitations can be summarized as follows:

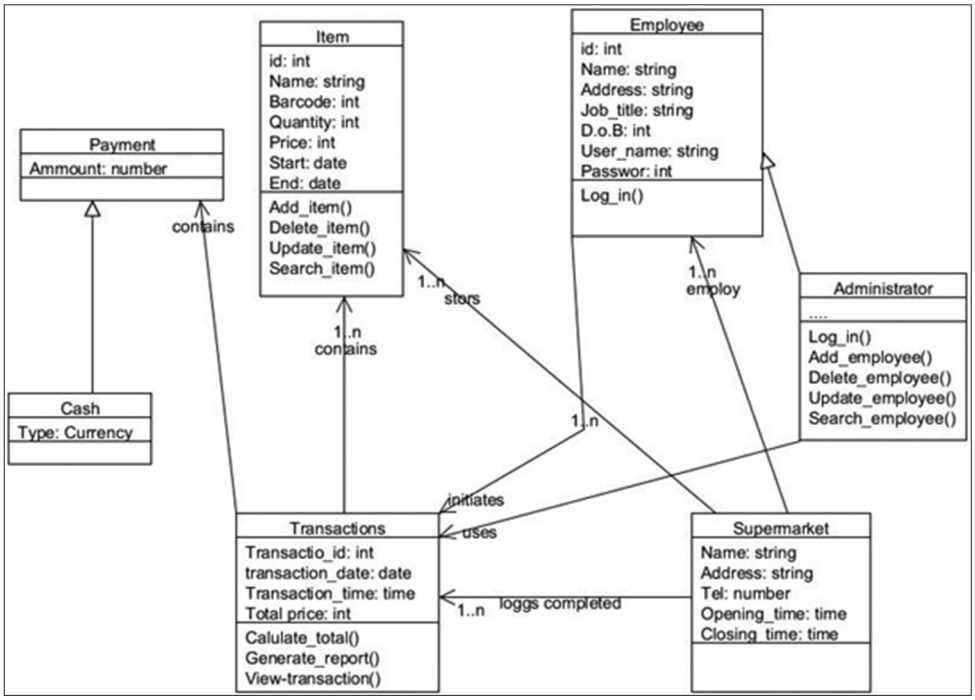

Fig. 6. Current electronic point of sale unified modeling language class diagram

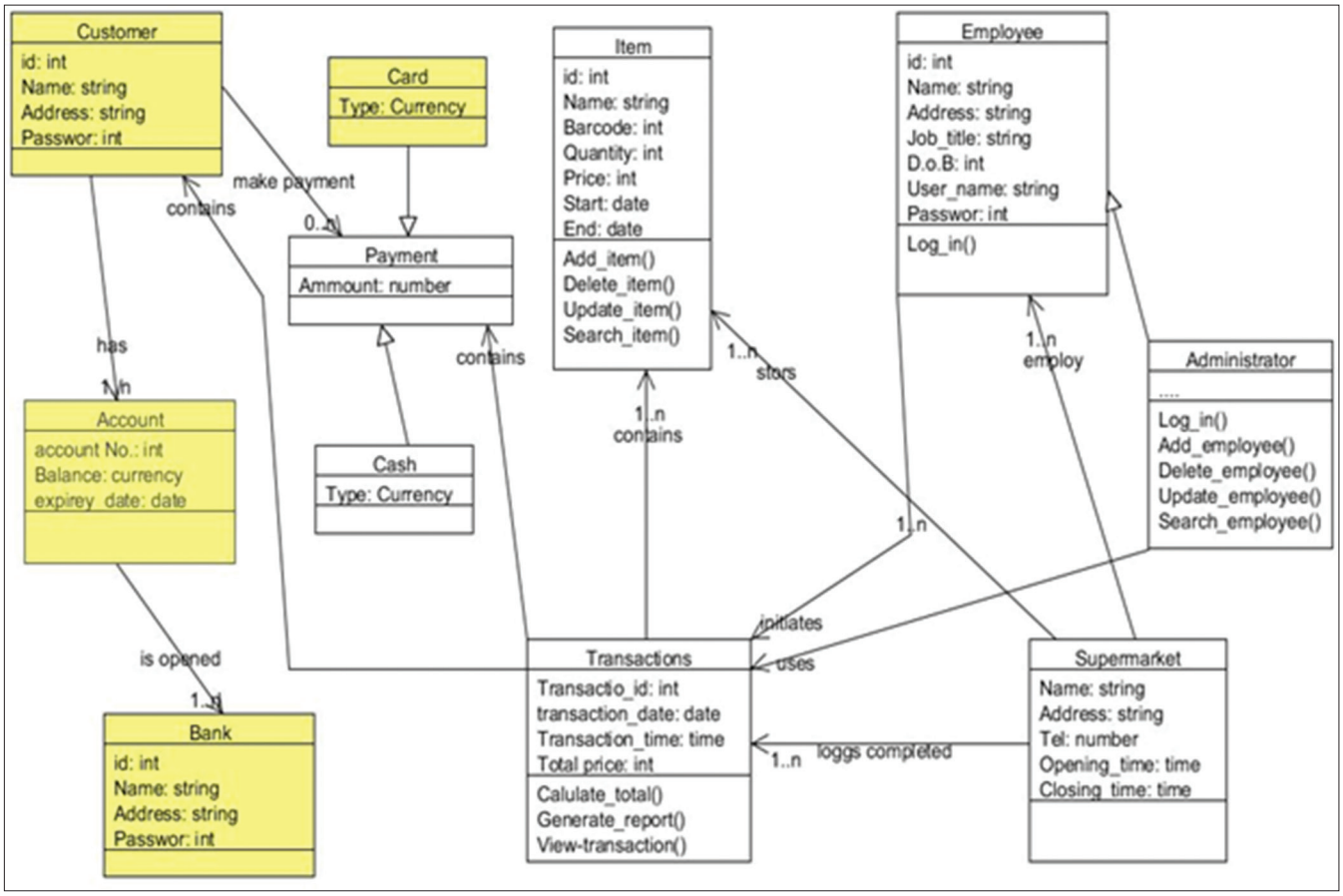

Fig. 7. Future electronic point of sale unified modeling language class diagram

1. Available software modeling tools such as UML 2.0 which is an industry standard currently does not provide the ability to model systems from user-defined viewpoint [21]
2. MDD is not widely used [22] although it has gained momentum with a potential for industry wide adaptation
3. Once the models are developed and finalized there comes the issues of translating it completely into code as a tool to generate 100% codes from a model not yet exist
4. The issue of measuring the quality of models is realized by researchers to tackle this issue France and Rumpe [2] suggests that modeling methods should come with modeling criteria that modelers can use as a guide for system modeling. However, such criterions are yet to be presented by modeling language and tool developers such as developers of UML (www.omg.org)
5. In the course of developing a system, many different models are created at varying abstract levels which creates model tracking, integration, and management issues and the current modeling tools are not sophisticated enough to deal with the issues.

Despite all the limitations, we believe the fundamental concept behind the framework has great potentials to be advanced and implemented in the future.

7. CONCLUSION AND RECOMMENDATIONS

Software aging is inevitable which occurs as a result of changing requirements, ignorant injections, and weight gain. Researchers have proposed a number of different approaches to tackle software aging; however, nearly all approaches are trying to address the aging caused by technical update or software malfunction. In this paper, we have outlined a framework for software rejuvenation that uses MDD approach as base for the rejuvenation process. The framework addresses software aging from a change in business requirement point of view which is different from what current researchers are proposing. It is simple, effective, and applicable as demonstrated by applying it to a simple business case study.

The foundation concept developed in this paper contributes to the field of software aging and paves the way for looking at software aging in a different angle.

Now to delay software aging, we recommend a number of quick mitigating actions as follows:

1. Characterize the changes that are likely to occur over the lifetime of a software product, and the way to achieve this characterization is by applying principles such as object orientation
2. Design and develop the software code in a way that changes can be carried out; to achieve this concise and clear documentation is the key
3. Reviewing and getting a second opinion on the design and documentation of a product helps in prolonging the lifetime of a software product.

When the aging has already occurred there are things we could do to treat it such as:

1. Prevent the aging process to get worse by introducing and creating structures whenever changes are made to the product
2. As changes are introduced to a product a review and update of the documentation is often a very effective step in slowing the aging process
3. Understanding and applying the principle of modularization is a good way to ease the future maintenance of a product
4. Combining different versions of similar functions into one system can increase efficiency of a software product and reduce the size of its code which is one the causes of software aging.

REFERENCES

[1] D. L. Parnas. "Software aging." in Proceedings of the 16th International Conference on Software Engineering, 1994, pp. 279-287.

[2] R. France and B. Rumpe. "Model-driven development of complex software: A research roadmap." in *2007 Future of Software Engineering.* Washington, DC, USA: IEEE Computer Society, 2007, pp. 37-54.

[3] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo. "A survey of software aging and rejuvenation studies." *ACM Journal on Emerging Technologies in Computing Systems (JETC),* vol. 10, no. 1, pp. 8, 2014.

[4] A. Avritzer and E. J. Weyuker. "Monitoring smoothly degrading systems for increased dependability." *Empirical Software Engineering*, vol. 2, no. 1, pp. 59-77, 1997.

[5] M. Grottke, R. Matias, and K. S. Trivedi. "The fundamentals of software aging." in Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on, 2008, pp. 1-6.

[6] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton. "Software rejuvenation: Analysis, module and applications." in Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers, Twenty-Fifth International Symposium on, 1995, pp. 381-390.

[7] C. Jones. "The economics of software maintenance in the twenty first century." Unpublished Manuscript, 2006. Available: http://www.compaid.com/caiinternet/ezine/capersjones-maintenance.pdf. [Last Accessed on 2017 May 15].

[8] R. L. Glass. "On the aging of software." *Information Systems Management*, vol. 28, no. 2, pp. 184-185, 2011.

[9] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo. "Software aging and rejuvenation: Where we are and where we are going." in Software Aging and Rejuvenation (WoSAR), 2011 IEEE Third International Workshop on, 2011, pp. 1-6.

[10] H. Okamura and T. Dohi. "Dynamic software rejuvenation policies in a transaction-based system under Markovian arrival processes." *Performance Evaluation*, vol. 70, no. 3, pp. 197-211, 2013.

[11] A. Pfening, S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. "Optimal software rejuvenation for tolerating soft failures." *Performance Evaluation*, vol. 27, pp. 491-506, 1996.

[12] P. Saravakos, G. Gravvanis, V. Koutras, and A. Platis. "A comprehensive approach to software aging and rejuvenation on a single node software system." in Proceedings of the 9th Hellenic European Research on Computer Mathematics and its Applications Conference (HERCMA 2009), 2009.

[13] T. Dohi, K. Goseva-Popstojanova and K. S. Trivedi. "Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule." in Dependable Computing, 2000. Proceedings. 2000 Pacific Rim International Symposium on, 2000, pp. 77-84.

[14] S. Garg, A. Puliafito, M. Telek and K. S. Trivedi. "Analysis of software rejuvenation using Markov regenerative stochastic Petri net." in Software Reliability Engineering, 1995. Proceedings, Sixth International Symposium on, 1995, pp. 180-187.

[15] F. Salfner and K. Wolter. "Analysis of service availability for time-triggered rejuvenation policies." *Journal of Systems and Software,* vol. 83, no. 9, pp. 1579-1590, 2010.

[16] F. Machida, J. Xiang, K. Tadano and Y. Maeno. "Software life-extension: A new countermeasure to software aging." in Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium on, 2012, pp. 131-140.

[17] K. J. Cassidy, K. C. Gross and A. Malekpour. "Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers." in Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on, 2002, pp. 478-482.

[18] S. Garg, A. van Moorsel, K. Vaidyanathan and K. S. Trivedi. "A methodology for detection and estimation of software aging." in Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on, 1998, pp. 283-292.

[19] S. Beydeda, M. Book, V. Gruhn, G. Booch, A. Brown, S. Iyengar, J. Rumbaugh and B. Selic. *Model-Driven Software Development*, vol. 15. Berlin: Springer, 2005.

[20] J. P. Tolvanen and S. Kelly. "Model-driven development challenges and solutions." *Modelsward*, vol. 2016, p. 711, 2016.

[21] R. B. France, S. Ghosh, T. Dinh-Trong and A. Solberg. "Model-driven development using UML 2.0: Promises and pitfalls." *Computer*, vol. 39, no. 2, pp. 59-66, 2006.

[22] S. J. Mellor, T. Clark and T. Futagami. "Model-driven development: Guest editors' introduction." *IEEE Software,* vol. 20, no. 5, pp. 14-18, 2003.

[23] P. Mayer, A. Schroeder and N. Koch. “MDD4SOA: Model-driven service orchestration.” in Enterprise Distributed Object Computing Conference, 2008. EDOC’08. 12th International IEEE, 2008, pp. 203-212.

[24] D. Harel, B. Rumpe. “Modeling languages: Syntax, semantics and all that stuff (or, what’s the semantics of semantics?).” in Technical Report MCS00-16, Weizmann Institute, Rehovot, Israel, 2004.

[25] N. B. Ruparelia. “Software development lifecycle models.” *SIGSOFT Software Engineering Notes*, vol. 35, no. 3, pp. 8-13, 2010.