Global Journal of Computer Sciences: Theory and Research

Volume 8, Issue 1, (2018) 01-13

www.gjcs.eu

Performance analysis of software maintenance process using stochastic Petri nets

Muhammad Nabeel, Department of Electrical and Computer Engineering, Centre for Advanced Studies in Engineering, 19-Attaturk Avenue, G-5/1, 41000 Islamabad, Pakistan.

Zeeshan Anwar*, Department of Computer Software Engineering, National University of Sciences and Technology, H-12, 41000 Islamabad, Pakistan.

Ali Ahsan, Foundation University, 41000 Islamabad, Pakistan.

Suggested Citation:

Nabeel, M., Anwar, Z. & Ahsan, A. (2018). Performance analysis of software maintenance process using stochastic Petri nets. *Global Journal of Computer Sciences: Theory and Research. 8*(1), 01–13.

Received from September 15, 2017; revised from December 02, 2017; accepted from February, 22, 2018.
Selection and peer review under responsibility of Prof. Dr. Dogan Ibrahim, Near East University, Cyprus.

Abstract

Software maintenance is a time taking activity in the real world. Execution time of software maintenance process may get increased due to interdepartmental communication, thus, increasing the cost and decreasing the performance of the process. We suggested performance evaluation of software maintenance process through the transformation of activity diagram into generalised stochastic Petri nets (GSPN). For this study, execution time and cost of maintenance process are defined as performance measures, and the role-based approach is used to understand the flow of software maintenance activities in a software organisation. Activity diagram is constructed to be transformed into GSPN. We used PIPE2 to analyse the GSPN. PIPE2 calculates average number of tokens on a place in the GSPN, throughput of timed transition and state space analysis. State space involves calculation of the reachability of the GSPN net that shows whether a GSPN holds the property of safeness, boundness and is deadlock free.

Keywords: Generalised stochastic Petri nets (GSPN), GSPN analysis, performance evaluation, performance modelling, software maintenance process, stochastic Petri nets.

* ADDRESS FOR CORRESPONDENCE: **Zeeshan Anwar**, Department of Computer Software Engineering, National University of Sciences and Technology, H-12, 41000 Islamabad, Pakistan. *E-mail address*: zeeshan.phdcse@stuents.mcs.edu.pk / Tel.: +92-333-572-3230

1. Introduction

Software maintenance is a critical process which can lead towards success or prodigious business loss. This paper focuses on the major issues being faced by the industry during the implementation and execution of the software maintenance process. So far, many software practitioners have proposed models to evaluate the performance of software maintenance process like software maintenance process evaluation using discrete event simulation (Marsan, 1995), use of indices systems for evaluation of software maintainability (Kim, Chung & Kim, 2005), RFD and CURE (Staines, 2010) maintenance models. Performance measures that are used by most of the performance evaluation models are complete cycle time (Kumar, 2012) of a particular activity in a process, workload (Bjorling & Hoff, 2002) on individual resource or at team level in a process, throughput of a process (Artikson, 1997) and communication paths (Warmer & Kleppe, 2003). Though all these models have their own significance, a common limitation observed is that they require software maintenance process to be executed first. This means after spending significant amount of time and money, a model is able to identify the performance issues. This research is an attempt to propose a methodology in which performance can be evaluated without executing the software maintenance process. This study will also be helpful in identifying the bottlenecks in software maintenance process by figuring out whether the process implementation will be safe and there will be no deadlocks in the implemented process. Another significance of the study is that it uses automation tool PIPE2, which will help in automating the performance evaluation process for software maintenance process activities. This will reduce the overhead time being used in calculating the performance parameters for software maintenance process.

Rest of the paper is organised as follows: literature review is given in Section 2 and Section 3 discusses the various techniques used to transform software maintenance process into generalised stochastic Petri nets (GSPN). In Section 4, a case study is given by transforming software maintenance process of an organisation into GSPN. Analysis is performed in Section 5. Sections 6 and 7 conclude and give future directions, respectively.

2. Literature Review

2.1. Software maintenance

Changes in software are required for bug fixing and improvements. Improvements are made to include new or modified requirements, upgradation of modules or technology. Maintenance efforts are required to cope with the above-mentioned problems and improvements (Hasan & Chakrborti, 2011).

IEEE (IEEE Std 1219–1998) formally defined software maintenance as ‘The totality of activities required to provide cost-effective support to a software system. Activities are performed during the pre-delivery stage as well as the post-delivery stage’.

Maintenance is further classified as adaptive, perfective, preventive, corrective and emergency maintenance (Benestad, Anda & Arisholm, 2009; Chang & Hsiang, 2011; Chapin, 2000; Hasan & Chakrborti, 2011; Hussian, Asghar, Ahmad & Ahmad, 2009; Schach, Jin, Yu, Heller & Offutt, 2003).

2.2. Petri nets

Petri nets are successful, because they use theoretical characteristics for precise modelling and analysis of system behaviour. Also, state changes can be visualised through graphical representation (Wang, 2007). Petri net is a type of bipartite directed graph (Murata, 1989) and comprises places, directed arcs and transitions. Directed arcs can be used to connect transitions to places or to connect places to transitions (ISO/IEC, 2010). Simplest Petri net which consists of input place P1, output place P2 and one transition T1 is shown in Figure 1.

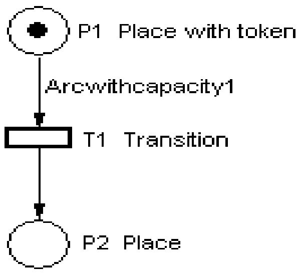
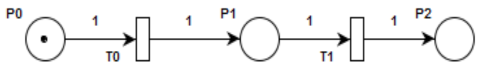
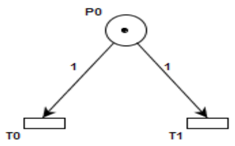
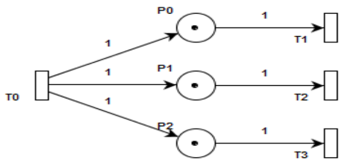
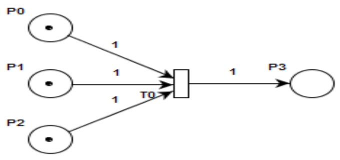
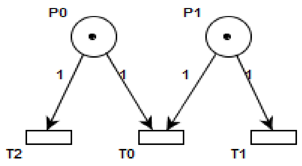
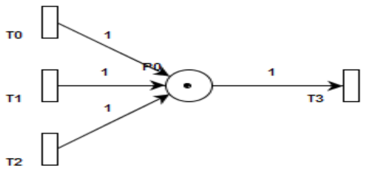


Figure 1. Basic Petri net

Properties of places and transitions can be found in (Mandrioli, Morzenti, Pietro & Silva, 1996) and properties of Petri nets are given in Table 1.

Table 1. Modelling power in Petri nets (Genrich & Lautenbach, 1989; Li & Zhou, 2009; Murata, 1989; http://en.wikipedia.org/wiki/Petri_net)

Petri net property	Corresponding model
Sequential: When T0 fires, then T1 can be fired	
Conflict: Two transitions T0 and T1 enabled due to token at P0. Token is removed and remaining transition gets disabled (T0 or T1) when either one transition fires	
Concurrency: Activities in a process can have concurrent behaviour. Transitions T1, T2 and T3 behave as concurrent activities	
Synchronisation: Petri nets have the power to model synchronisation. P3 will starts only when P0, P1 and P2 get finished	
Confusion: Confusion state will arise when T1 fires and T0 is still enabled. This is due to the enablement of all the transitions (T0, T1 and T2)	
Merging: In merging, parallel processes are merged, so transition firing time could be different	

2.3. Stochastic Petri nets

A stochastic process, also called random process, contains random variables that represent the progression of system having random values over time (Emadi & Shams, 2009). Models developed by using SPN allow proof of correctness, integration of formal description and performance evaluation (Fagundes, Maciel & Rosa, 2007).

2.4. Basic model of stochastic Petri nets

An SPN is six-tuples, SPN = (P, T, I, O, M0, A) where (P, T, I, O, M0) is the marked untimed PN underlying SPN. *A* = (*Λ*1, *Λ*2, …, *Λn*) is an array (possibly marking dependent) which consists of firing rates allied with transitions. In stochastic Petri nets, transition firing holds the condition of atomicity i.e. with one indivisible operation tokens from input places are removed and deposited into output places (Marsan, 1990). Each transition is linked with a firing delay. Firing delay is the amount of time to which transition must hold itself before firing. The firing delay is classified as random variable with negative exponential probability density function (pdf; Miwa, Li, Ge, Matsuno & Miyano, 2011). The parameter of the pdf (according to probability theory, when the probability distribution is defined as a function over general sets of values, then pdf is used) associated with transition t_i is the firing rate associated with t_i and X_i. The average firing delay of transition t_i in marking M_j is $[X_i(M_j)] - 1$.

2.5. Performance evaluation of process

Cost and time are indicators used to measure process performance (Cao & Hoffman, 2011). The performance objectives are to be defined as they play important role in performance evaluation of a process. Performance objectives help in stickiness towards acceptable and focused solution of process execution. Performance objectives guide to what extent efforts should be made in smooth execution of process (Jung & Goldenson, 2009). When process performance evaluation is desired, then considerable substances are number of employees required in order to execute a process, available resource pool (skill set level), collaboration time cycle with associated departments, time required for research and development and change in time cycle due to process variation (Cao & Hoffman, 2011; Kumar, 2012; Stoddard, 2007; Stoddard & Goldenson, 2010; Tan, Shen & Zhao, 2007; http://www.omg.org/technology/documents/formal/uml.htm).

2.6. Software maintenance process evaluation using discrete event simulation

Due to multiple response loops (interdepartmental and client communication) and complex cause–effect bindings, it becomes difficult to analyse the performance of software maintenance process. The above discussed problems of complexity have been coped with simulation power, as simulation can give deeper insight into process activities and impact change on performance of process without actually implementing the process in the real environment.

Modification requests change their states and uniqueness due to occurrence of discrete events in a software maintenance process. State of a modification request can be modelled in order to study the behaviour changes of a modification request. The simulation model represents software maintenance process as decision tree, which will serve as input description, in order to study the system.

3. Techniques for Activities Transformation into SPN

Adel Ouardani, Esteban, Paludetto and Pascal (2006) uses requirements validation process in order to transform UML diagrams into Petri nets (Merseguer, 2003; Rungworawut & Senivongse, 2005). UML activity diagrams (Staines, 2010) are considered important as they are easy and provide powerful visual modelling techniques, which describe number of behaviour found in information and computer systems (Canevet, Gilmore, Hilliston, Kloul & Stevens, 2004; http://www.omg.org/technology/documents/formal/uml.htm). In practical, activity model can be used but not limited to: web service composition, web processing, system integration, business process modelling, task management and software operation tasks modelling (Rungworawut & Senivongse, 2005). Modelling with GSPN is explained in (Marsan, Balbo, Conte, Donatelli & Franceschinis, 1994) and also, uses case diagrams that can have transformed into Petri nets.

Performance models are being considered for performance measurement as discussed in (Merseguer, 2003). Activity diagrams can be used for performance evaluation (Lopez-Grao, Merseguer & Campos, 2004). Merseguer (Motameni, Movaghar & Amiri, 2007) consider non-functional parameters of a software system and uses UML activity diagrams to obtain Petri net model with focus on performance and reliability. In order to analyse the stochastic behaviour of the system, performance parameters are obtained from GSPN model (Motameni, Movaghar & Mozafari, 2005). GSPN model is then used to derive embedded continuous-time Markov chain (Motameni, Montazeri, Siasifar, Movaghar & Zandakbari, 2007).

Three main reasons were involved for using Petri nets for capturing object-oriented behavioural design. The first reason is concurrency, synchronisation and resource sharing behaviour of a system that can be modelled using Petri net. Second, issues related to deadlock and performance analysis can be analysed using numerical results. Finally, automation of behavioural analysis can be achieved through integration of Petri nets and object-oriented design. UML diagrams are powerful tool for system design but they are unable to address non-functional parameters. This means UML diagrams cannot be used for performance evaluation, so in order to solve this problem UML diagrams were translated into GSPN (Marsan, 1995; Merseguer, LopezGrao & Campos, 2004). Overall process for transformation of activity diagrams into GSPN for performance evaluation is given in Figure 2.

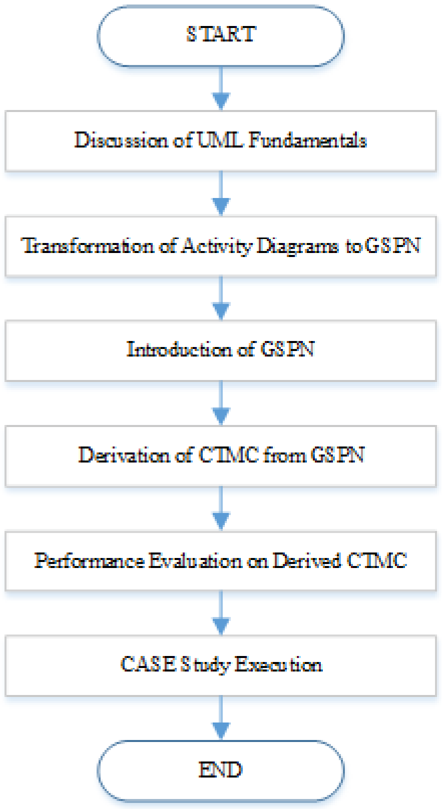

START
Discussion of UML Fundamentals
Transformation of Activity Diagrams to GSPN
Introduction of GSPN
Derivation of CTMC from GSPN
Performance Evaluation on Derived CTMC
CASE Study Execution
END

Figure 2. Transformation process from activity diagrams into GSPN

4. Case Study

4.1. Organisation introduction

An organisation which is working on client communication management, team collaboration and content management solutions and having clients in more than 30 countries was chosen. Due to diverse culture, changing business needs and rapid change in technology, organisation faces immense challenges in maintenance of the products. Maintenance of such mission critical software required that there should be a software maintenance process which is cost effective and less time consuming. Much of the time is wasted due to lack of timely communication between different departments within the organisation and also communication to client for clarification of requirements.

Roles and responsibilities involved in software maintenance process are presented in form of activity diagram given in Appendix A. Same activity diagram is used for case study execution. Following section presents mapping of software maintenance process into GSPN and their corresponding analysis using PIPE2.

4.2. Mapping of MR/PR analysis into GSPN

Figure 3 presents the mapping of MR/PR analysis into GSPN.

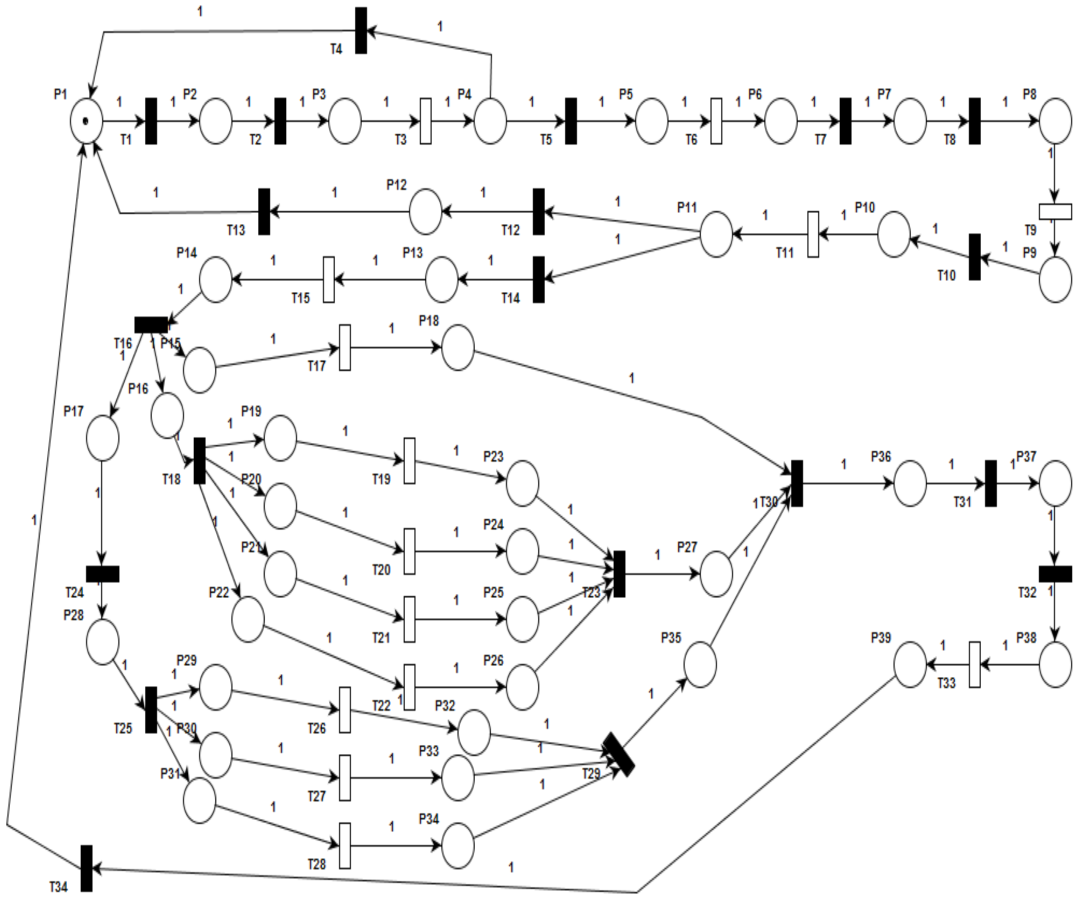


Figure 3. Mapping of MR/PR analysis into GSPN

4.3. GSPN of modification implementation

Figure 4 presents the mapping of modification implementation into GSPN.

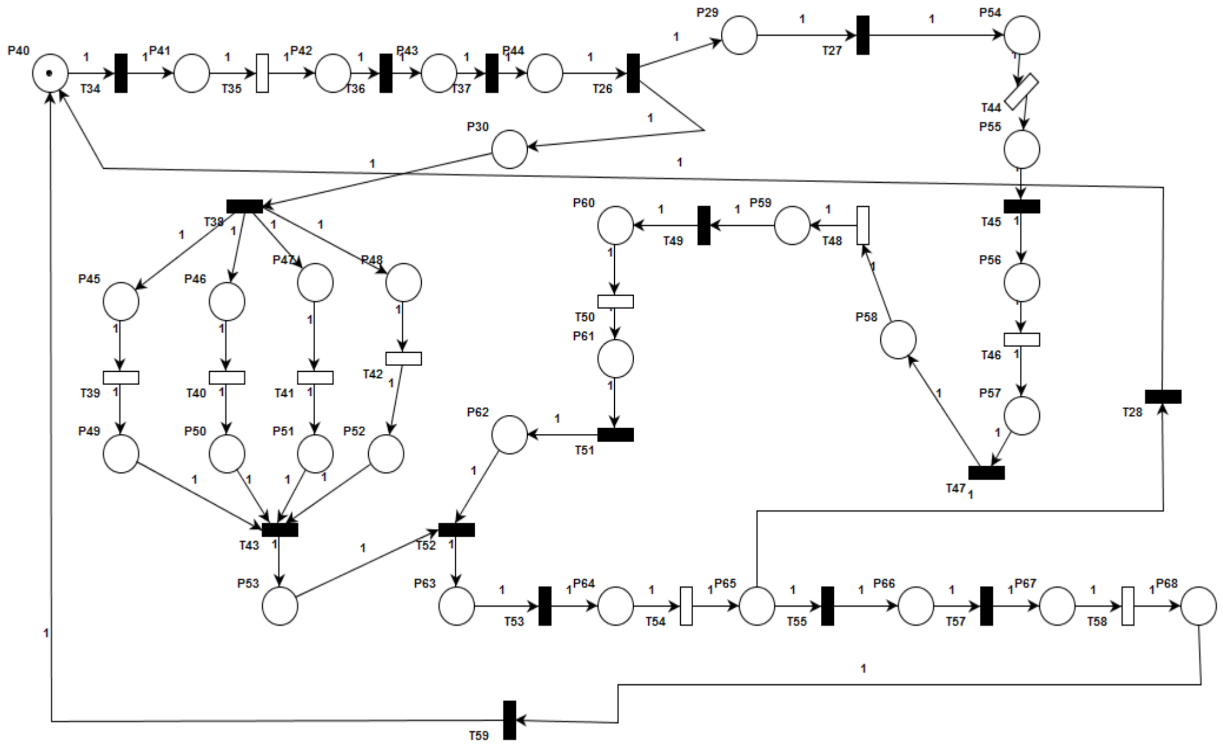


Figure 4. Mapping of modification implementation into GSPN

4.4. GSPN of maintenance review and acceptance

Figure 5 presents the mapping of maintenance review and acceptance into GSPN.

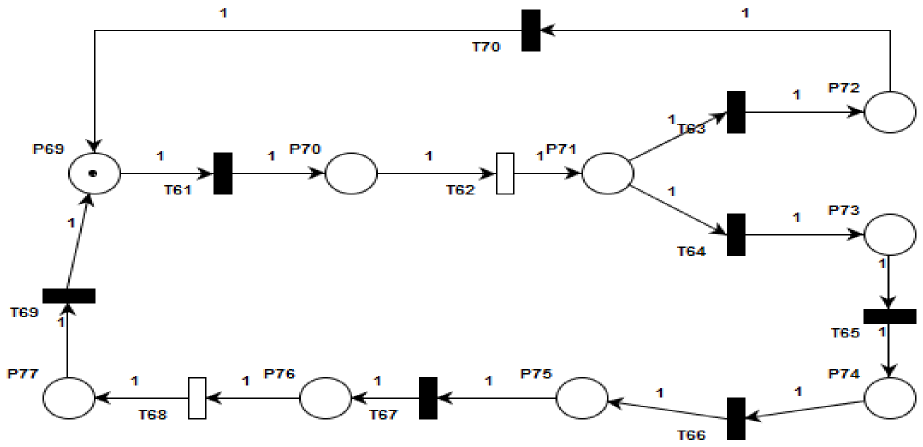
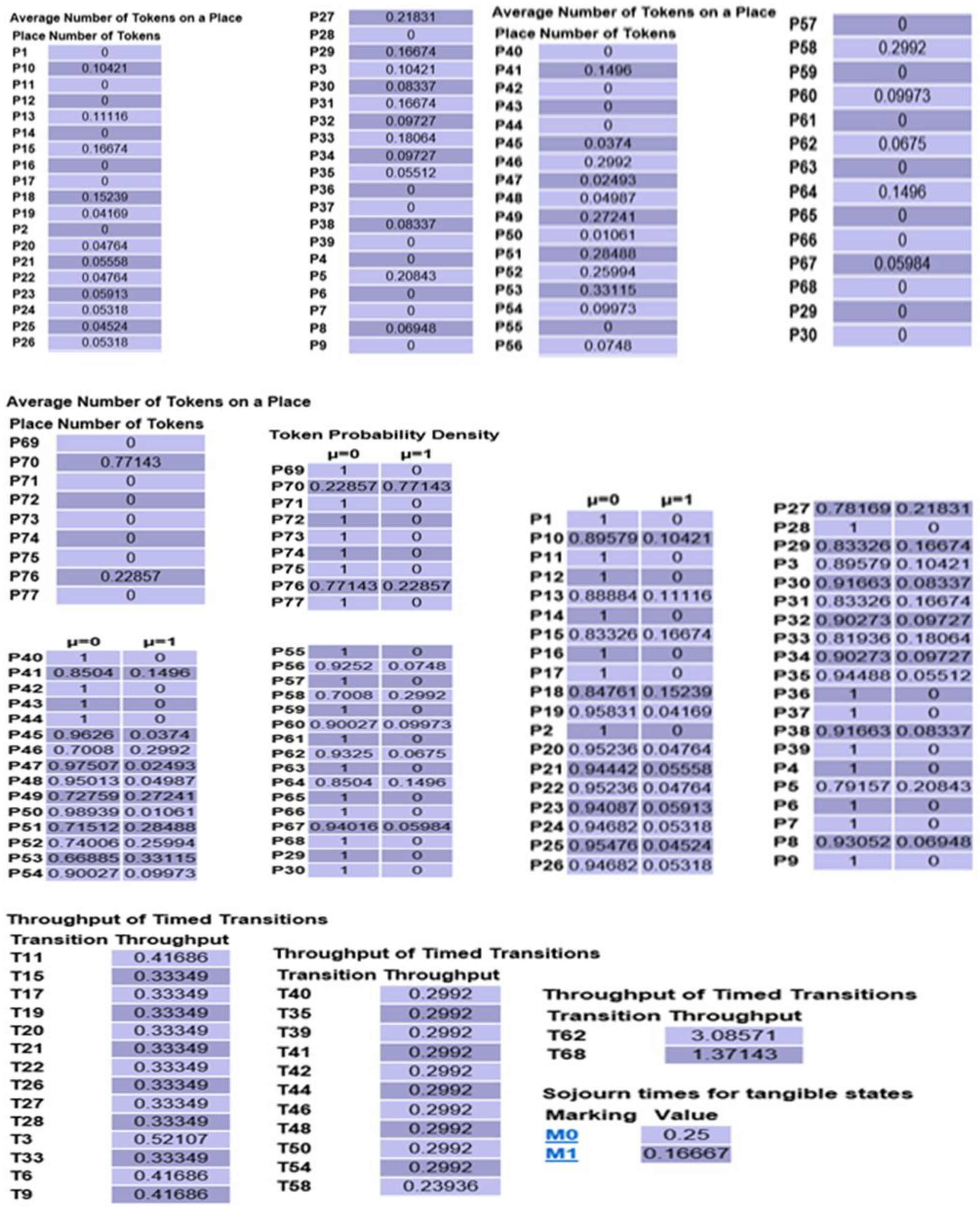


Figure 5. Mapping of maintenance review and acceptance into GSPN

5. Analysis of Results

GSPN analysis is presented in Figure 6 on the results taken from experiments performed using PIPE2.



Average Number of Tokens on a Place

Place	Number of Tokens
P1	0
P10	0.10421
P11	0
P12	0
P13	0.11116
P14	0
P15	0.16674
P16	0
P17	0
P18	0.15239
P19	0.04169
P2	0
P20	0.04764
P21	0.05558
P22	0.04764
P23	0.05913
P24	0.05318
P25	0.04524
P26	0.05318
P27	0.21831
P28	0
P29	0.16674
P3	0.10421
P30	0.08337
P31	0.16674
P32	0.09727
P33	0.18064
P34	0.09727
P35	0.05512
P36	0
P37	0
P38	0.08337
P39	0
P4	0
P5	0.20843
P6	0
P7	0
P8	0.06948
P9	0

Average Number of Tokens on a Place

Place	Number of Tokens
P40	0
P41	0.1496
P42	0
P43	0
P44	0
P45	0.0374
P46	0.2992
P47	0.02493
P48	0.04987
P49	0.27241
P50	0.01061
P51	0.28488
P52	0.25994
P53	0.33115
P54	0.09973
P55	0
P56	0.0748
P57	0
P58	0.2992
P59	0
P60	0.09973
P61	0
P62	0.0675
P63	0
P64	0.1496
P65	0
P66	0
P67	0.05984
P68	0
P29	0
P30	0

Average Number of Tokens on a Place

Place	Number of Tokens
P69	0
P70	0.77143
P71	0
P72	0
P73	0
P74	0
P75	0
P76	0.22857
P77	0

Token Probability Density

	μ=0	μ=1
P69	1	0
P70	0.22857	0.77143
P71	1	0
P72	1	0
P73	1	0
P74	1	0
P75	1	0
P76	0.77143	0.22857
P77	1	0

	μ=0	μ=1
P1	1	0
P10	0.89579	0.10421
P11	1	0
P12	1	0
P13	0.88884	0.11116
P14	1	0
P15	0.83326	0.16674
P16	1	0
P17	1	0
P18	0.84761	0.15239
P19	0.95831	0.04169
P2	1	0
P20	0.95236	0.04764
P21	0.94442	0.05558
P22	0.95236	0.04764
P23	0.94087	0.05913
P24	0.94682	0.05318
P25	0.95476	0.04524
P26	0.94682	0.05318
P27	0.78169	0.21831
P28	1	0
P29	0.83326	0.16674
P3	0.89579	0.10421
P30	0.91663	0.08337
P31	0.83326	0.16674
P32	0.90273	0.09727
P33	0.81936	0.18064
P34	0.90273	0.09727
P35	0.94488	0.05512
P36	1	0
P37	1	0
P38	0.91663	0.08337
P39	1	0
P4	1	0
P5	0.79157	0.20843
P6	1	0
P7	1	0
P8	0.93052	0.06948
P9	1	0

	μ=0	μ=1
P40	1	0
P41	0.8504	0.1496
P42	1	0
P43	1	0
P44	1	0
P45	0.9626	0.0374
P46	0.7008	0.2992
P47	0.97507	0.02493
P48	0.95013	0.04987
P49	0.72759	0.27241
P50	0.98939	0.01061
P51	0.71512	0.28488
P52	0.74006	0.25994
P53	0.66885	0.33115
P54	0.90027	0.09973
P55	1	0
P56	0.9252	0.0748
P57	1	0
P58	0.7008	0.2992
P59	1	0
P60	0.90027	0.09973
P61	1	0
P62	0.9325	0.0675
P63	1	0
P64	0.8504	0.1496
P65	1	0
P66	1	0
P67	0.94016	0.05984
P68	1	0
P29	1	0
P30	1	0

Throughput of Timed Transitions

Transition	Throughput
T11	0.41686
T15	0.33349
T17	0.33349
T19	0.33349
T20	0.33349
T21	0.33349
T22	0.33349
T26	0.33349
T27	0.33349
T28	0.33349
T3	0.52107
T33	0.33349
T6	0.41686
T9	0.41686

Throughput of Timed Transitions

Transition	Throughput
T40	0.2992
T35	0.2992
T39	0.2992
T41	0.2992
T42	0.2992
T44	0.2992
T46	0.2992
T48	0.2992
T50	0.2992
T54	0.2992
T58	0.23936

Throughput of Timed Transitions

Transition	Throughput
T62	3.08571
T68	1.37143

Sojourn times for tangible states

Marking	Value
M0	0.25
M1	0.16667

Figure 6. GSPN analysis of MR/PR analysis, modification implementation and MR/PR review and acceptance

For every GSPN presented in the Figures 3–5, PIPE2 calculates the performance measures like average number of tokens on place, probability density of token, timed transitions with throughput, minimal siphons and minimal traps, steady space analysis and Petri net simulation results. Tokens are used to study the dynamic actions of a system modelled using Petri net in view of system's states and states changes. A place can hold none or positive numeral of tokens. Condition allied with a place can either be false or true and can be determined by presence or absence of a token in that place. These distributions of tokens are used in understanding the behaviour of the process and contribute towards structural analysis of Petri net, which will show whether a Petri net is safe, bounded or deadlock free. Steady space analysis for GSPN of MR/PR analysis, modification implementation and MR/PR review and acceptance show that nets are bounded, safe and deadlock Free.

6. Research Findings

It is possible to perform performance evaluation of software maintenance process using GSPN. Time and cost can be used as measures for performance evaluation. In this study, only time is used as measure for performance evaluation, time determines the delay in activities. By this approach, bottlenecks in the process can be identified and changes are proposed to improve maintenance process. Software maintenance process was mapped into GSPN and performance evaluation was simulated in an automated tool PIPE2.

Reachability Graph is calculated in order to satisfy the structural properties of created GSPNs for maintenance process. Reachability graph is used in order to answer whether the constructed GSPN is holding safety property which ensured that net is feasible for performance evaluation. Through the calculation of reachability graph, it shown that constructed nets are deadlock free which means software maintenance process has no dead ends and customer request is not stuck at any stage.

Time measures are mentioned in Table 2 in the form of analysis time for MR/PR analysis, modification implementation and MR/PR review and acceptance.

Measures mentioned in Figure 6 in form of timed transitions can be combined with time measures of analysis time in order to calculate the cost spent in software maintenance process. This calculated cost can be compared to other cost calculating models in order to validate the effectiveness of current stated methodology of performance evaluation for the under consideration process.

Table 2. State space exploration

GSPN	State space exploration time (seconds)	Steady state distribution time (seconds)	Total time (seconds)
MR/PR analysis	2.91	2.91	2.91
Modification implementation	0.415	0.415	0.415
MR/PR review and acceptance	0.075	0.075	0.075

7. Conclusion

This research proposed a new technique which is powerful and can be represented graphically for the performance evaluation of software maintenance process, i.e. GSPN. Through PIPE2 token, probability density, place holding average number of tokens and throughput of timed transitions have been calculated for GSPN of MR/PR analysis, modification implementation and MR/PR review and acceptance. State space analysis for each particular GSPN has been performed and results show that each GSPN has satisfied the properties of: boundness, safeness and deadlock free. Time taken in state space exploration, time taken in solving the steady state distribution and total time taken by activities have also been calculated through PIPE2.

8. Future Work

In our future work, we are planning to use other techniques like use case, collaboration and workflow diagrams for performance evaluation of maintenance process. Furthermore, performance evaluation of software maintenance process using Performance Query Editor module available in PIPE2 can also be utilised.

References

Artikson, C. (1997). Meta-modeling for distributed object environments. In *IEEE enterprise distributed object computing workshop* (pp. 90–101).

Benestad, H. C., Anda, B. & Arisholm, E. (2009). Understanding software maintenance and evolution by analyzing individual changes: A literature review. *Journal of Software Maintenance and Evolution: Research and Practice, 21*, 349–378.

Bjorling, E. & Hoff, A. (2002). *An evaluation of a maintenance model: A comparison with theory and results from case studies* (Unpublished Master's thesis). Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Karlskrona, Sweden.

Canevet, C., Gilmore, S., Hilliston, J., Kloul, L. & Stevens, P. (2004). *Analysing UML 2.0 activity diagrams in the software engineering performance process*. Redwood, CA: ACM.

Cao, Q. & Hoffman, J. J. (2011). A case study approach for developing a project performance evaluation system. *International Journal of Project Management, 29*(2), 155–164.

Chang, C. K. & Hsiang, C. L. (2011). Using generalized stochastic Petri nets for preventive maintenance optimization in automated manufacturing systems. *Journal of Quality, 18*(2), 117–135.

Chapin, N. (2000). Software maintenance types—a fresh view. In *Proceedings of the international conference on software maintenance (ICSM'2000)* (pp. 247–252).

Emadi, S. & Shams, F. (2009). Transformation of use case and sequence diagrams into Petri nets. *Computing, Communication, Control, and Management, 4*, 399–403.

Fagundes, R. A. A., Maciel, P. R. M. & Rosa, S. (2007). Performance evaluation of CORBA concurrency control service using stochastic Petri nets. *RITA, 14*(2), 109–132.

Genrich, J. H. & Lautenbach, K. (1989). System modeling with high-level Petri nets. *Theoretical Computer Science, 3*(1), 109–136.

Hasan, R. & Chakrborti, S. (2011). Investigating software maintenance challenges in small organizations. *Americas Conference on Information Systems, 17*, 1–9.

Hussian, S., Asghar, M. Z., Ahmad, B. & Ahmad, S. (2009). A step towards software corrective maintenance: Using RCM model. *International Journal of Computer Science and Information Security, 4*(1 & 2), 25-36.

IEEE Std 1219–1998. *IEEE standard for software maintenance*. Revision of IEEE Std. 1219–1992, Approved 25 June 1998, ISBN 0-7381-0336-5.

ISO/IEC. (2010). *ISO/IEC 15909-1:2004/Amd.1:2010(en) systems and software engineering—High-level Petri nets—Part 1: Concepts, definitions and graphical notation AMENDMENT 1: Symmetric nets*. Retrieved from https://www.iso.org

Jung, H. W. & Goldenson, D. R. (2009). Evaluating the relationship between process improvement and schedule deviation in software maintenance. *Information and Software Technology, 51*(2), 351–336.

Kim, G., Chung, W. & Kim, M. (2005). A selection framework of multiple navigation primitives using generalized stochastic Petri nets. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (pp. 3790–3795).

Kumar, B. (2012). A survey of key factors affecting software maintainability. In *International Conference on Computing Sciences* (pp. 261–266).

Li, Z. & Zhou, M. (2009). *Deadlock resolution in automated manufacturing systems—A novel Petri net approach*. London, UK: Springer-Verlag.

Lopez-Grao, J. P., Merseguer, J. & Campos, J. (2004). From UML activity diagrams to stochastic Petri nets: Application to software performance engineering. *Proceedings of the 4th International Workshop on Software and Performance, 29*(1), 25–36.

Mandrioli, D., Morzenti, A., Pietro, S. & Silva, S. (1996). *A Petri net and logic approach to the specification and verification of real time systems*. Hoboken, NJ: John Wiley & Sons. R

Marsan, M. A. (1990). Stochastic Petri nets: An elementary introduction. *Lecture Notes in Computer Science, 424*, 1–29.

Marsan, M. A. (1995). *Modelling with generalized stochastic Petri nets. John Wiley Series in Parallel Computing Chichester*. Chichester, UK: John Wiley & Sons.

Marsan, M. A., Balbo, G., Conte, G., Donatelli, S. & Franceschinis, G. (1994). *Modelling with generalized stochastic Petri nets* (1st ed.). New York, NY: John Wiley & Sons.

Merseguer, J. (2003). *Software performance engineering based on UML and Petri nets* (Unpublished Ph.D. thesis). Departamento de Informatica e Ingenieria de Sistemas, Universidad de Zaragoza, Zaragoza, Spain.

Merseguer, J., LopezGrao, J. P. & Campos, J. (2004). From UML activity diagrams to stochastic Petri nets: Application to Software Performance Engineering. *ACM WOSP 04, 29*(1), 25–36.

Miwa, Y., Li, C., Ge, Q. W., Matsuno, H. & Miyano, S. (2011). On determining firing delay time of transitions for Petri net based signalling pathways by introducing stochastic decision rules. *Studies in Health Technology and Informatics, 162*, 204–221.

Motameni, H., Movaghar, A. & Amiri, M. F. (2007). Mapping activity diagram to Petri net: Application of Markov theory foranalyzing non-functional parameters. *International Journal of Engineering Transactions, 20*(1), 65–76.

Motameni, H., Movaghar, A. & Mozafari, M. (2005). Evaluating UML state diagrams using colored Petri net. In *Proceeding of SYNASC'05, Romania* (pp. 87–108).

Motameni, H., Montazeri, H., Siasifar, M., Movaghar, A. & Zandakbari, M. (2007). Mapping state diagram to Petri net: An approach to use Markov theory for analyzing non-functional parameters from state diagram. In K. Elleithy (Ed.), *Advances and innovations in systems, computing sciences and software engineering* (pp. 185–190). Dordrecht, The Netherlands: Springer.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE, 77*(4), 541–580.

Ouardani, A., Esteban, P., Paludetto, M. & Pascal, J. C. (2006). A meta-modeling approach for sequence diagrams to Petri nets transformation within the requirements validation process. In *Proceedings of the 20th European simulation and modeling conference* (pp. 1–10).

Rungworawut, W. & Senivongse, T. (2005). A guideline to mapping business processes to UML class diagrams. *WSEAS Transactions on Computers, 4*(11), 1526–1533.

Schach, S. R., Jin, B., Yu, L., Heller, G. Z. & Offutt, J. (2003). Determining the distribution of maintenance categories: Survey versus measurement. *Empirical Software Engineering, 8*(4), 351–365.

Staines, A. S. (2010). A triple graph grammar mapping of UML 2 activities into Petri nets. In *SEPADS'10 proceedings of the 9th WSEAS international conference on Software engineering, parallel and distributed systems* (pp. 90–95). Retrieved from http://dl.acm.org/

Stoddard, R. W. (2007). *CMMI process performance models and reliability*. Retrieved from http://paris.utdallas.edu/IEEE-RS-ATR/document/2007/2007-14.pdf

Stoddard, R. W. & Goldenson, D. R. (2010). Approaches to process performance modelling. In *A summary from the SEI series of workshops on CMMI high maturity measurement and analysis technical report, CMU/SEI-2009-TR-021, ESC-TR-2009-021* (pp. 1–106). Retrieved from http://repository.cmu.edu/

Tan, W. A., Shen, W. & Zhao, J. (2007). A methodology for dynamic enterprise process performance evaluation. *Computers in Industry, 58*(5), 474–485.

Wang, J. (2007). Petri nets for dynamic event-driven system modeling. In P. Fishwick (Ed.), *Handbook of dynamic system modelling*. Boca Raton, FL: Chapman & Hall.

Warmer, J. B. & Kleppe, A. J. (2003). *The object constraint language second edition: Getting your models for MDA*. Boston, MA: Addison-Wesley.

Appendix A

OMG UML 2 superstructure specification. V2.2, OMG. Retrieved from http://www.omg.org/technology/documents/formal/uml.htm

Activity diagram for Software Maintenance Process of Alpha Technologies

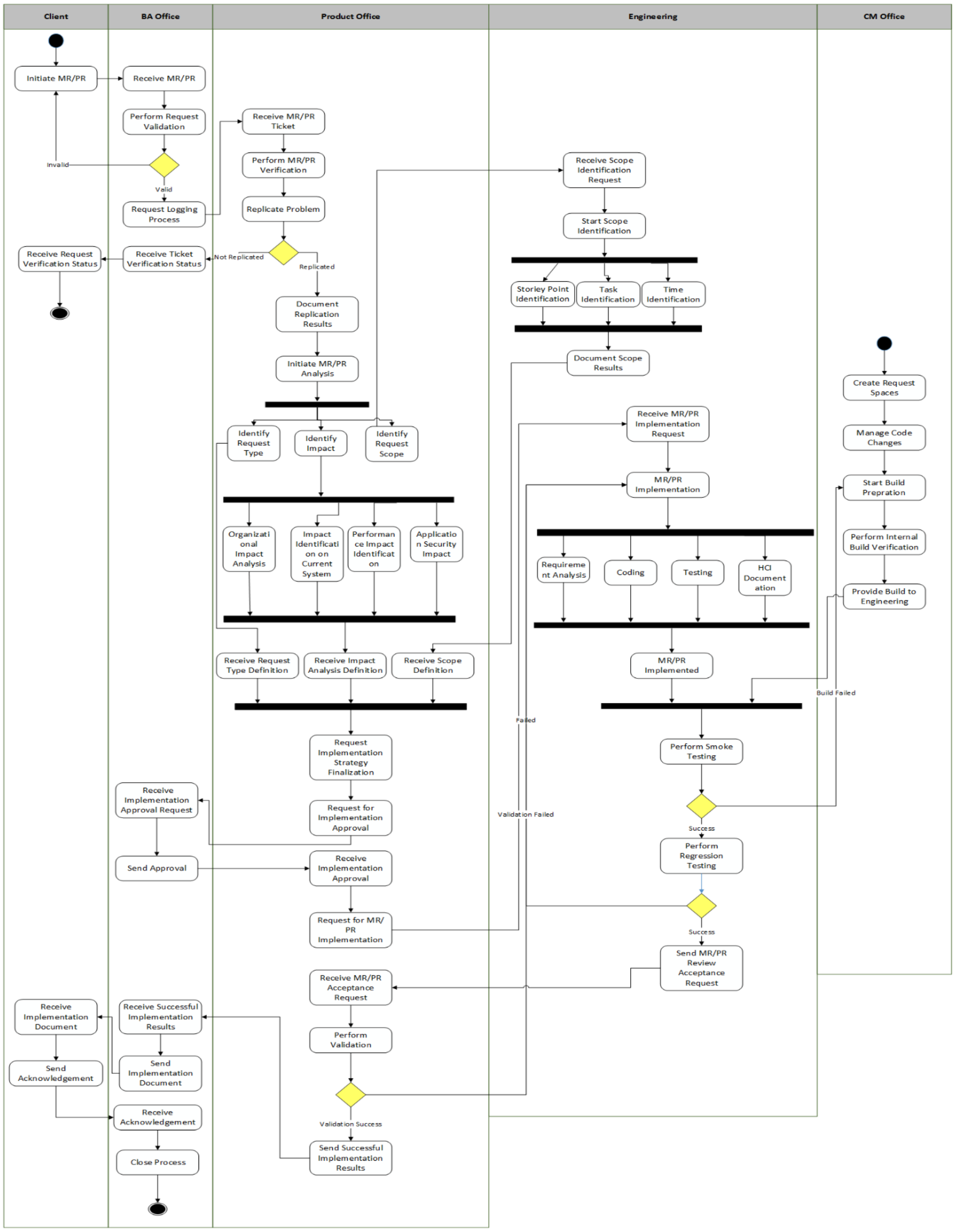

Client
BA Office
Product Office
Engineering
CM Office
Initiate MR/PR
Receive MR/PR
Perform Request Validation
Invalid
Valid
Request Logging Process
Receive MR/PR Ticket
Perform MR/PR Verification
Replicate Problem
Not Replicated
Replicated
Receive Request Verification Status
Receive Ticket Verification Status
Document Replication Results
Initiate MR/PR Analysis
Identify Request Type
Identify Impact
Identify Request Scope
Organizational Impact Analysis
Impact Identification on Current System
Performance Impact Identification
Application Security Impact
Receive Request Type Definition
Receive Impact Analysis Definition
Receive Scope Definition
Request Implementation Strategy Finalization
Request for Implementation Approval
Receive Implementation Approval Request
Send Approval
Receive Implementation Approval
Request for MR/PR Implementation
Receive MR/PR Acceptance Request
Perform Validation
Validation Success
Send Successful Implementation Results
Receive Successful Implementation Results
Send Implementation Document
Receive Implementation Document
Send Acknowledgement
Receive Acknowledgement
Close Process
Receive Scope Identification Request
Start Scope Identification
Storey Point Identification
Task Identification
Time Identification
Document Scope Results
Receive MR/PR Implementation Request
MR/PR Implementation
Requirement Analysis
Coding
Testing
HCI Documentation
MR/PR Implemented
Failed
Validation Failed
Build Failed
Perform Smoke Testing
Success
Perform Regression Testing
Success
Send MR/PR Review Acceptance Request
Create Request Spaces
Manage Code Changes
Start Build Prepration
Perform Internal Build Verification
Provide Build to Engineering

Figure 7. Activity diagram for software maintenance process