A TEST CASE GENERATION FRAMEWORK BASED ON UML STATECHART DIAGRAM

YASIR DAWOOD SALMAN

DOCTOR OF PHILOSOPHY
UNIVERSITI UTARA MALAYSIA
2018

Awang Had Salleh
Graduate School
of Arts And Sciences

Universiti Utara Malaysia

PERAKUAN KERJA TESIS / DISERTASI
(Certification of thesis / dissertation)

Kami, yang bertandatangan, memperakukan bahawa
(We, the undersigned, certify that)

YASIR DAWOOD SALMAN

calon untuk Ijazah **PhD**
(candidate for the degree of)

telah mengemukakan tesis / disertasi yang bertajuk:
(has presented his/her thesis / dissertation of the following title):

"A TEST CASE GENERATION FRAMEWORK BASED ON UML STATECHART DIAGRAM"

seperti yang tercatat di muka surat tajuk dan kulit tesis / disertasi.
(as it appears on the title page and front cover of the thesis / dissertation).

Bahawa tesis/disertasi tersebut boleh diterima dari segi bentuk serta kandungan dan meliputi bidang ilmu dengan memuaskan, sebagaimana yang ditunjukkan oleh calon dalam ujian lisan yang diadakan pada : ***10 October 2017.***
That the said thesis/dissertation is acceptable in form and content and displays a satisfactory knowledge of the field of study as demonstrated by the candidate through an oral examination held on: ***October 10, 2017.***

Pengerusi Viva: *(Chairman for VIVA)*	**Assoc. Prof. Dr. Yuhanis Yusof**	Tandatangan *(Signature)*
Pemeriksa Luar: *(External Examiner)*	**Prof. Dr. Abu Bakar Md Sultan**	Tandatangan *(Signature)*
Pemeriksa Dalam: *(Internal Examiner)*	**Dr. Rohaida Romli**	Tandatangan *(Signature)*
Nama Penyelia/Penyelia-penyelia: *(Name of Supervisor/Supervisors)*	**Dr. Nor Laily Hashim**	Tandatangan *(Signature)*

Tarikh:
(Date) ***October 10, 2017***

Permission to Use

In presenting this thesis in fulfilment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the Universiti Library may make it freely available for inspection. I further agree that permission for the copying of this thesis in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying, publication, or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my thesis.

Requests for permission to copy or to make other use of materials in this thesis, in completely or in part should be addressed to:

Dean of Awang Had Salleh Graduate School of Arts and Sciences
UUM College of Arts and Sciences
Universiti Utara Malaysia
06010 UUM Sintok

Abstrak

Pengesanan awal kesalahan perisian menawarkan lebih fleksibiliti untuk membetulkan kesalahan tersebut pada peringkat awal pembangunan sistem. Malangnya, kajian sedia ada masih belum cukup menyeluruh dalam menerangkan proses utama penjanaan kes ujian secara automatik. Malahan algoritma yang digunakan dalam penjanaan ujian kes tidak disediakan atau diterangkan dengan jelas. Kajian semasa juga hampir tidak menangani isu gelung dan laluan selari, malahan kriteria liputan yang dicapai adalah rendah. Oleh itu, kajian ini mencadangkan satu kerangka penjanaan kes ujian yang menjana kes ujian yang diminimumkan dan diprioritasikan daripada gambarajah UML keadaan dengan kriteria liputan yang lebih tinggi. Kajian literatur telah dilaksanakan untuk mengenal pasti isu dan jurang yang berkaitan penjanaan kes ujian, pengujian berasaskan model, dan kriteria liputan. Kerangka yang dicadangkan ini direka bentuk hasil daripada maklumat yang dikumpul dan telah mengenalpasti lapan komponen yang mewakili proses dalam penjanaan kes ujian. Komponen tersebut adalah jadual hubungan, graf hubungan, pemeriksaan konsistensi, meminimumkan laluan ujian, memprioritasikan laluan ujian, pemangkasan laluan, penjanaan laluan ujian dan penjanaan kes ujian. Sebagai tambahan, satu prototaip untuk melaksanakan kerangka turut dibangunkan. Penilaian kerangka yang dibangunkan melibatkan tiga fasa: prototaip, perbandingan dengan kajian terdahulu dan ulasan pakar. Dapatan kajian menunjukkan kriteria liputan yang paling sesuai bagi gambarajah UML keadaan adalah liputan semua keadaan, liputan semua peralihan, liputan semua pasangan peralihan, dan liputan semua laluan gelung bebas. Selain itu, kajian ini mencapai kriteria liputan yang lebih tinggi dalam semua kriteria liputan yang dinyatakan di atas, kecuali liputan semua keadaan apabila dibandingkan dengan kajian sebelumnya. Hasil ulasan pakar menunjukkan bahawa pakar domain bersetuju bahawa kerangka yang dicadangkan ini adalah praktikal, mudah untuk dilaksanakan kerana kesesuaiannya dalam menjanakan kes ujian. Algoritma yang dicadangkan menghasilkan keputusan yang betul, dan prototaip berupaya menjana kes ujian dengan berkesan. Secara umumnya, sistem yang dicadangkan diterima baik oleh pakar berdasarkan aspek kebergunaan, kebolehgunaan, dan ketepatannya. Kajian ini menyumbang secara teori dan praktikal dengan menyediakan kerangka penjanaan kes ujian alternatif awal yang mencapai liputan yang tinggi dan dapat dilaksanakan dengan efektif menggunakan gambarajah UML keadaan. Kajian ini turut menambahkan pengetahuan baru dalam bidang pengujian perisian khususnya kepada proses pengujian dalam teknik berasaskan model, aktiviti pengujian, dan alat sokongan pengujian.

Kata kunci: Kerangka penjanaan kes ujian, liputan gelung, laluan selari, kes ujian yang diminimumkan, kes ujian yang diprioritasikan.

Abstract

Early software fault detection offers more flexibility to correct errors in the early development stages. Unfortunately, existing studies in this domain are not sufficiently comprehensive in describing the major processes of the automated test case generation. Furthermore, the algorithms used for test case generation are not provided or well described. Current studies also hardly address loops and parallel paths issues, and achieved low coverage criteria. Therefore, this study proposes a test case generation framework that generates minimized and prioritized test cases from UML statechart diagram with higher coverage criteria. This study, conducted a review of the previous research to identify the issues and gaps related to test case generation, model-based testing, and coverage criteria. The proposed framework was designed from the gathered information based on the reviews and consists of eight components that represent a comprehensive test case generation processes. They are relation table, relation graph, consistency checking, test path minimization, test path prioritization, path pruning, test path generation, and test case generation. In addition, a prototype to implement the framework was developed. The evaluation of the framework was conducted in three phases: prototyping, comparison with previous studies, and expert review. The results reveal that the most suitable coverage criteria for UML statechart diagram are all-states coverage, all-transitions coverage, all-transition-pairs coverage, and all-loop-free-paths coverage. Furthermore, this study achieves higher coverage criteria in all coverage criteria, except for all-state coverage, when compared with the previous studies. The results of the experts' review show that the framework is practical, easy to implement due to it is suitability to generate the test cases. The proposed algorithms provide correct results, and the prototype is able to generate test case effectively. Generally, the proposed system is well accepted by experts owing to its usefulness, usability, and accuracy. This study contributes to both theory and practice by providing an early alternative test case generation framework that achieves high coverage and can effectively generate test cases from UML statechart diagrams. This research adds new knowledge to the software testing field, especially for testing processes in the model-based techniques, testing activity, and testing tool support.

Keywords: Test case generation framework, loop coverage, parallel path, minimized test cases, prioritized test cases

Acknowledgement

All praises and thanks to the Almighty, Allah (SWT), for giving me the strength, the patience, and the opportunity to complete this thesis. Besides, completing this thesis would not have been possible without a number of people who offered their unfailing support throughout the period of the study.

I would like to express my sincerest thanks and gratitude to my supervisor Dr. Nor Laily Binti Hashim for the continuous support of my PhD study and related research, for her patience, motivation, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my PhD study.

To my family, thank you for encouraging me in all of my pursuits and inspiring me to follow my dreams. I am especially grateful to my parents, who have been a constant source of inspiration to me. It is their love, patience and encouragement during the PhD period that helped me through hardships. I always knew that you believed in me and wanted the best for me. My special gratitude to my brother and sister for supporting me spiritually throughout writing this thesis and my life in general.

I would also like to thank my colleagues at Information Technology, Universiti Utara Malaysia for their encouragement and support throughout this journey. I would also like also to extend my thanks and appreciation to all of my friends who have contributed in one way or another to help me complete this thesis successfully.

I could not have completed my thesis without the support of all these wonderful people!

Table of Contents

List of Tables

List of Figures

List of Appendices

List of Publications

Published

Hashim, N. L., & **Salman, Y. D.** (2011). An improved algorithm in test case generation from UML activity diagram using activity path. *Proceedings of the 3rd International Conference on Computing and Informatics, ICOCI*.

Salman, Y. D., & Hashim, N. L. (2014). An improved method of obtaining basic path testing for test case based on UML state chart. *Science International, 26*(4).

Salman, Y. D., & Hashim, N. L. (2016). Automatic Test Case Generation from UML State Chart Diagram: A Survey *Advanced Computer and Communication Engineering Technology* (pp. 123-134): Springer.

Salman, Y. D., Hashim, N. L., Rejab, M. M., Romli, R., & Mohd, H. (2017a). *Coverage criteria for test case generation using UML state chart diagram.* Paper presented at the AIP Conference Proceedings.

Salman, Y. D., Hashim, N. L., Rejab, M. M., Romli, R., & Mohd, H. (2017b). Coverage Criteria for UML State Chart Diagram in Model-based Testing. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 9*(2-11), 85-89.

Salman, Y. D., & Hashim, N. L. (2017). Test Case Generation Model for UML Diagrams. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC),* 9(2-2), 171-175.

Accepted

Salman, Y. D., Hashim, N. L., Rejab, M. M., Romli, R., & Mohd, H. (2017). Generating test cases for model-based testing and detecting deadlocks using Tarjan's algorithm.

Salman, Y. D. & Hashim, N. L. (2017). A Test Cases Minimization And Prioritized Method Using Firefly Algorithm Based On UML State Chart Diagram.

List of Abbreviations

BFS	Breath First Search
DFS	Depth First Search
EFSM	Extended Finite State Machine
FSM	Finite State Machine
GeMiTefSc	Generation and Minimization of Test Cases from State Chart
GTSC	Automated Generated Test Case Based on Statechart
IOCO	Input/Output Conformance
MBT	Model-based testing
OCL	Object Constraint Language
OMDAG	Object Method Acyclic Graph
OMG	Object Management Group
POS	Practical Swarm Optimization
PSO	Particle Swarm Optimization
RAD	Rapid Application Development
SAD	State Activity Diagram
SCCF	Statechart Coverage Criteria Family
SCOTEM	State COllaboration TEst Model
SRT	State Relationship Table
SRG	State Relationship Graph
SUT	System Under Test
TCG	Test Cases Generation
TCGP	Test Case Generation Paths
TeGeMiOOSc	Test Generation and Minimization for O-O Software with State Charts
TFG	Test Flow Graph
TGV	Test Generation with Verification
UML	Unified Modelling Language
VDT	Vertex Description Table
XML	Extensible Markup Language

CHAPTER ONE
INTRODUCTION

1.1 Introduction

This introductory chapter deliberates on the motivational aspects of software testing in general and automatic test case generation in practice, and focuses on using Unified Modelling Language (UML) diagrams as inputs to generate test cases.

This chapter presents the introduction to this study, beginning with the background of the study, which includes the background of software testing and automatic test case generation and the related literature. The next sections present the research problems, research questions, and research objectives. Subsequently, the scope of the research and the research framework will be discussed. Finally, the significance of the study and the terminologies will be presented. This chapter is concluded with an outline of the remaining chapters of this thesis.

1.2 Background of the Study

Computers and software are some of the major innovations in the history of mankind (Srivastav & Gupta, 2016). The use of computers plays a key role in the daily lives of people. The significant roles of computers in society and the increasing demand for complex computer applications makes software development difficult for software developers (Chavez, Shen, France, Mechling, & Li, 2016). Thus, the effort exerted and the cost of software development testing ultimately increases (Chen & Li, 2010).

The contents of the thesis is for internal user only

REFERENCES

Abdurazik, A., & Offutt, J. (1999). *Generating test cases from UML specifications.* George Mason University.

Abdurazik, A., & Offutt, J. (2000). *Using UML collaboration diagrams for static checking and test generation.* Paper presented at the «UML» 2000 -The Unified Modeling Language.

Abdurazik, A., Offutt, J., & Baldini, A. (2004). A controlled experimental evaluation of test cases generated from UML diagrams: Technical Report, ISE-TR-04-03. George Mason University.

Aggarwal, M., & Sabharwal, S. (2012). *Test case generation from UML state machine diagram: A survey.* Paper presented at the Computer and Communication Technology (ICCCT), 2012 Third International Conference on.

Ahamed, S. (2010). Studying the feasibility and importance of software testing: An analysis. *Internatinal Journal of Engineering Science and Technology, 1*(3), 119-128.

Ahmad, J., & Baharom, S. (2017). A Systematic Literature Review of the Test Case Prioritization Technique for Sequence of Events. *International Journal of Applied Engineering Research, 12*(7), 1389-1395.

Ahmed, B. S. (2016). Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing. *Engineering Science and Technology, an International Journal, 19*(2), 737-753.

Aichernig, B. K. (2001). *Systematic black-box testing of computer-based systems through formal abstraction techniques.* (PhD Dissertation), Graz University of Technology, Graz, Austria.

Al-kahlout, A., B. salha, B., & El-haddad, N. (2017). *E-Account APP*. University of Palestine, Gaza Strip, Palestine.

Al-Tarawneh, F. H. (2014). *A framework for cots software evaluation and selection for COTS mismatches handling and non-functional requirements.* Universiti Utara Malaysia.

Al Dallal, J., & Sorenson, P. (2006). Generating class based test cases for interface classes of object-oriented black box frameworks. *Transactions on Engineering, Computing and Technology, 16*, 90-95. doi: 10.1.1.193.4045

Alhroob, A. (2014). *Best Test Cases Selection Approach*. Paper presented at the Scientific Cooperations International Workshops on Electrical and Computer Engineering Subfields.

Alhroob, A. M. (2012). *Software test case generation from system models and specification. Use of the UML diagrams and High Level Petri Nets models for developing software test cases.* University of Bradford.

Ali, M. A., Shaik, K., & Kumar, S. (2014). Test case generation using UML state diagram and OCL expression. *International Journal of Computer Applications, 95*(12), 7 -11. doi: 10.5120/ijais2016451599

Ali, S., Briand, L. C., Hemmati, H., & Panesar-Walawege, R. K. (2010). A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Transactions on software engineering, 36*(6), 742-762. doi: 10.1109/TSE.2009.52

Ali, S., Briand, L. C., Rehman, M. J.-u., Asghar, H., Iqbal, M. Z. Z., & Nadeem, A. (2007). A state-based approach to integration testing based on UML models. *Information and Software Technology, 49*(11), 1087–1106. doi: 10.1016/j.infsof.2006.11.002

Ammann, P., & Offutt, J. (2008). *Introduction to software testing*. Cambridge, United Kingdom: Cambridge University Press.

Anand, S., Burke, E. K., Chen, T. Y., Clark, J., Cohen, M. B., Grieskamp, W., . . . McMinn, P. (2013). An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software, 86*(8), 1978–2001. doi: 10.1016/j.jss.2013.02.061

Avancena, A. T., & Nishihara, A. (2015). Usability and pedagogical assessment of an algorithm learning tool: a case study for an introductory programming course for high school. *Issues in Informing Science & Information Technology, 12*, 21-44.

Bahrin, Z. S. (2011). *Mobile game-based learning (mGBL) engineering model.* Universiti Utara Malaysia.

Baig, M. M. (2009). *New software testing strategy.* NED University of Engineering & Technology, Karachi.

Baudry, B., Fleurey, F., Jézéquel, J.-M., & Le Traon, Y. (2005). Automatic test case optimization: A bacteriologic algorithm. *IEEE software, 22*(2), 76-82.

Bell, D. (2003). UML basics Part III: The class diagram. *The Rational Edge Nov*.

Belli, F., & Hollmann, A. (2008). *Test generation and minimization with basic statecharts.* Paper presented at the Proceedings of the 2008 ACM symposium on Applied computing.

Belli, F., Hollmann, A., & Kleinselbeck, M. (2009). *A graph-model-based testing method compared with the classification tree method for test case generation.* Paper presented at the Secure Software Integration and Reliability Improvement, 2009. SSIRI 2009. Third IEEE International Conference on.

Bentley, J. E. (2005). *Software testing fundamentals-concepts, roles, and terminology.* Paper presented at the Proceedings of SAS Conference.

Berardi, D., Calvanese, D., & De Giacomo, G. (2005). Reasoning on UML class diagrams. *Artificial Intelligence, 168*(1), 70-118. doi: 10.1016/j.artint.2005.05.003

Bernstein, P. A. (1996). Middleware: a model for distributed system services. *Communications of the ACM, 39*(2), 86-98.

Bertolino, A. (2003). *Software testing research and practice.* Paper presented at the Abstract State Machines 2003.

Bertolino, A. (2007). *Software testing research: Achievements, challenges, dreams.* Paper presented at the Future of Software Engineering.

Beynon-Davies, P., Carne, C., Mackay, H., & Tudhope, D. (1999). Rapid application development (RAD): an empirical review. *European Journal of Information Systems, 8*(3), 211-223. doi: 10.1057/palgrave.ejis.3000325

Bhat, S., & Prashanth, C. (2014). A study on automatic test case generation. *International Journal of Engineering Sciences & Research Technology, 3*(4), 4073-4079. doi: 10.1.1.682.1527

Binder, R. V. (2000). *Testing object-oriented systems: models, patterns, and tools.* Massachusetts, United States: Addison-Wesley Longman Publishing Co., Inc.

Biswal, B. N. (2010). *Test case generation and optimization of object-oriented software using UML behavioral models.*

Biswal, B. N., Nanda, P., & Mohapatra, D. P. (2008). *A novel approach for scenario-based test case generation.* Paper presented at the Information Technology, 2008. ICIT'08. International Conference on.

Blanco, R., Fanjul, J., & Tuya, J. (2010). Test case generation for transition-pair coverage using Scatter Search. *International Journal of Software Engineering and Its Applications, 4*(4), 37-56. doi: 10.1.1.233.765

Boehm, B., & Basili, V. R. (2005). Software defect reduction top 10 list. *Foundations of empirical software engineering: the legacy of Victor R. Basili, 426*(426-431). doi: 10.1109/2.962984

Boghdady, P., Badr, N., Hashem, M., & Tolba, M. (2012). *An enhanced technique for generating hybrid coverage test cases using activity diagrams.* Paper presented at the Informatics and Systems (INFOS), 2012 8th International Conference on.

Boghdady, P. N., Badr, N. L., Hashem, M., & Tolba, M. F. (2011a). A proposed test case generation technique based on activity diagrams. *International Journal of Engineering & Technology IJET-IJENS, 11*(03), 37-57. doi: 10.1.1.296.9294

Boghdady, P. N., Badr, N. L., Hashim, M. A., & Tolba, M. F. (2011b). *An enhanced test case generation technique based on activity diagrams.* Paper presented at the Computer Engineering & Systems (ICCES), 2011 International Conference on.

Booch, G. (2005). *The unified modeling language user guide*. London, United Kingdom: Pearson Education.

Bozeman, C., Ellsworth, A., Hogben, L., Lin, J. C.-H., Maurer, G., Nowak, K., . . . Strickland, J. (2015). Minimum rank of graphs with loops. *Electronic Journal of Linear Algebra, 27*(1), 1071. doi: 10.13001/1081-3810.2007

Bozkurt, M., Harman, M., & Hassoun, Y. (2013). Testing and verification in service-oriented architecture: a survey. *Software Testing, Verification and Reliability, 23*(4), 261-313.

Briand, L. C., Labiche, Y., & Cui, J. (2005). Automated support for deriving test requirements from UML statecharts. *Software & Systems Modeling, 4*(4), 399–423. doi: 10.1007/s10270-005-0090-5

Budnik, C. J., Subramanyan, R., & Vieira, M. (2008). Peer-to-Peer Comparison of Model-Based Test Tools. *GI Jahrestagung (1), 133*, 223-226.

Cain, A., Chen, T. Y., Grant, D., Poon, P.-L., Tang, S.-F., & Tse, T. (2003). An automatic test data generation system based on the integrated classification-tree methodology *Software Engineering Research and Applications* (pp. 225-238): Springer.

Calisir, F., & Calisir, F. (2004). The relation of interface usability characteristics, perceived usefulness, and perceived ease of use to end-user satisfaction with enterprise resource planning (ERP) systems. *Computers in Human Behavior, 20*(4), 505-515.

Carmel, E., & Becker, S. (1995). A process model for packaged software development. *Engineering Management, IEEE Transactions on, 42*(1), 50-61.

Cartaxo, E. G., Neto, F. G. O., & Machado, P. D. (2007). *Test case generation by means of UML sequence diagrams and labeled transition Systems.* Paper presented at the Systems, Man and Cybernetics.

Cavarra, A., Crichton, C., Davies, J., Hartman, A., & Mounier, L. (2002). *Using UML for automatic test generation.* Paper presented at the international symposium on software testing and analysis ISSTA.

Chan, E. P., & Lim, H. (2007). Optimization and evaluation of shortest path queries. *The VLDB Journal—The International Journal on Very Large Data Bases, 16*(3), 343-369.

Chavez, H. M., Shen, W., France, R. B., Mechling, B. A., & Li, G. (2016). An approach to checking consistency between UML class model and its Java implementation. *IEEE Transactions on software engineering, 42*(4), 322-344.

Chen, L., & Li, Q. (2010). *Automated test case generation from use case: A model based approach.* Paper presented at the Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on.

Chen, M., Mishra, P., & Kalita, D. (2008). *Coverage-driven automatic test generation for UML activity diagrams.* Paper presented at the Proceedings of the 18th ACM Great Lakes symposium on VLSI, Orlando, Florida, USA.

Chen, M., Qiu, X., Xu, W., Wang, L., Zhao, J., & Li, X. (2009). UML activity diagram-based automatic test case generation for Java programs. *Computer Journal, 52(5)*, 545-556.

Chen, T., Poon, P.-L., & Tse, T. (1999). *A new restructuring algorithm for the classification-tree method.* Paper presented at the Software Technology and Engineering Practice, 1999. STEP'99. Proceedings.

Chimisliu, V., & Wotawa, F. (2012). *Model based test case generation for distributed embedded systems*. Paper presented at the Industrial Technology (ICIT), 2012 IEEE International Conference on.

Chimisliu, V., & Wotawa, F. (2013a). *Improving test case generation from UML statecharts by using control, data and communication dependencies.* Paper presented at the Quality Software (QSIC), 2013 13th International Conference on.

Chimisliu, V., & Wotawa, F. (2013b). *Using dependency relations to improve test case generation from UML statecharts.* Paper presented at the Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual.

Choudhary, K., Gigras, Y., & Rani, P. (2016). *Cuckoo Search in Test Case Generation and Conforming Optimality using Firefly Algorithm.* Paper presented at the Proceedings of the Second International Conference on Computer and Communication Technologies.

Chrissis, M. B., Konrad, M., & Shrum, S. (2011). *CMMI for development: guidelines for process integration and product improvement*. London, United Kingdom: Pearson Education.

Claude, J., & Thierry, J. (2002). TGV: theory, principles and algorithms: A Tool for the Automatic Synthesis of Conformance Test Cases for Non-Deterministic Reactive Systems. *Software Tools for Technology Transfer, 7*(4), 297-315.

Costagliola, G., Ferrucci, F., & Francese, R. (2002). Web engineering: Models and methodologies for the design of hypermedia applications. *Handbook of Software Engineering & Knowledge Engineering, 2*, 181-199.

Cruz-Lemus, J. A., Maes, A., Genero, M., Poels, G., & Piattini, M. (2010). The impact of structural complexity on the understandability of UML statechart diagrams. *Information Sciences, 180*(11), 2209-2220.

D'Souza, S., Rao, A., Sharma, A., & Singh, S. (2012). Modeling and verification of a multi-agent argumentation system using NuSMV. *arXiv preprint arXiv:1209.4330*.

Dahiya, S. S., Chhabra, J. K., & Kumar, S. (2010). *Application of artificial bee colony algorithm to software testing.* Paper presented at the Software Engineering Conference (ASWEC), 2010 21st Australian.

Daneva, M., & Ahituv, N. (2011). What practitioners think of inter-organizational ERP requirements engineering practices: focus group results. *International Journal of Information System Modeling and Design, 2*(3), 49-74.

Das, J. (2014). *Bengali digit recognition using adjacency matrix.* Jadavpur University Kolkata.

Dawson, M., Burrell, D. N., Rahim, E., & Brewster, S. (2010). Integrating software assurance into the Software Development Life Cycle (SDLC). *Journal of Information Systems Technology and Planning, 3*(6), 49-53.

Devroey, X., Perrouin, G., Legay, A., Cordy, M., Schobbens, P.-Y., & Heymans, P. (2014). *Coverage criteria for behavioural testing of software product lines.* Paper presented at the International Symposium On Leveraging Applications of Formal Methods, Verification and Validation.

Diestel, R. (2012). *Graph theory*. Berlin, Germany: Springer-Verlag Berlin Heidelberg.

Dinh-Trong, T. T., Ghosh, S., & France, R. B. (2006). *A systematic approach to generate inputs to test UML design models.* Paper presented at the Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on.

Dix, A. (2009). *Human-computer interaction*. United States: Springer

Doungsa-ard, C. (2012). *Generation of software test data from the design specification using heuristic techniques. Exploring the UML state machine diagrams and GA based heuristic techniques in the automated generation of software test data and test code.* University of Bradford.

Doungsa-ard, C., Dahal, K., Hossain, A., & Suwannasart, T. (2008). GA-based automatic test data generation for UML state diagrams with parallel paths *Advanced Design and Manufacture to Gain a Competitive Edge* (pp. 147-156): Springer.

Drusinsky, D. (2011). *Modeling and verification using UML statecharts: a working guide to reactive system design, Runtime Monitoring and Execution-based Model Checking*: Elsevier.

Dssouli, R., Saleh, K., Aboulhamid, E., En-Nouaary, A., & Bourhfir, C. (1999). Test development for communication protocols: towards automation. *Computer Networks, 31*(17), 1835-1872.

Dubey, Y., Singh, D., & Singh, A. (2016). A parallel early binding recursive Ant Colony optimization (PEB-RAC) approach for generating optimized auto test cases from programming inputs. *International Journal of Computer Applications, 136*(3), 11-17.

Dustin, E., Garrett, T., & Gauf, B. (2009). *Implementing automated software testing: How to save time and lower costs while raising quality*. London, United Kingdom: Pearson Education.

Easterbrook, S., Singer, J., Storey, M.-A., & Damian, D. (2008). Selecting empirical methods for software engineering research *Guide to advanced empirical software engineering* (pp. 285-311): Springer.

Edvardsson, J. (1999). *A survey on automatic test data generation.* Paper presented at the Proceedings of the 2nd Conference on Computer Science and Engineering.

Eghbali, S., & Tahvildari, L. (2016). Test case prioritization using lexicographical ordering. *IEEE Transactions on software engineering, 42*(12), 1178-1195.

Elallaoui, M., Nafil, K., Touahni, R., & Messoussi, R. (2016). Automated model driven testing using AndroMDA and UML2 testing profile in scrum process. *Procedia Computer Science, 83*, 221-228.

Eshuis, R. (2006). Symbolic model checking of UML activity diagrams. *ACM Transactions on Software Engineering and Methodology (TOSEM), 15*(1), 1-38.

Eusuff, M., Lansey, K., & Pasha, F. (2006). Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. *Engineering Optimization, 38*(2), 129-154.

Fan, X., Shu, J., Liu, L., & Liang, Q. J. (2009). *Test case generation from uml subactivity and activity diagram.* Paper presented at the Electronic Commerce and Security, 2009. ISECS'09. Second International Symposium on.

Farooq, S. U., & Quadri, S. (2011). Evaluating effectiveness of software testing techniques with emphasis on enhancing software reliability. *Journal of emerging trends in Computing and Information Sciences, 2*(12), 740-745.

Felderer, M., & Herrmann, A. (2015). Manual test case derivation from UML activity diagrams and state machines: A controlled experiment. *Information and Software Technology, 61*, 1-15.

Felicie, A. L. (2012). *UML state machine*. Rhode Island, United States: Salve Regina University

Frantzen, L., Tretmans, J., & Willemse, T. A. (2006). A symbolic framework for model-based testing *Formal approaches to software testing and runtime verification* (pp. 40-54): Springer.

Fraser, G., & Wotawa, F. (2007). *Test-case generation and coverage analysis for nondeterministic systems using model-checkers.* Paper presented at the Software Engineering Advances, 2007. ICSEA 2007. International Conference on.

Garousi, V. (2010). Applying peer reviews in software engineering education: an experiment and lessons learned. *IEEE Transactions on Education, 53*(2), 182-193. doi: 10.1109/TE.2008.2010994

Ghai, S., & Kaur, S. (2017). Hill-Climbing Approach for Test Case Prioritization. *International Journal of Software Engineering and Its Applications, 11*(3), 13-20.

Gnesi, S., Latella, D., & Massink, M. (2004). *Formal test case generation for UML statecharts.* Paper presented at the Engineering Complex Computer Systems, 2004. Proceedings. Ninth IEEE International Conference on.

Gogolla, M., Hamann, L., Hilken, F., Sedlmeier, M., & Nguyen, Q. D. (2014). *Behavior modeling with interaction diagrams in a UML and OCL tool.* Paper presented at the Proceedings of the 2014 Workshop on Behaviour Modelling-Foundations and Applications.

Goodubaigari, A. (2013). A software test data generation tool for unit testing of C++ programs using control flow graph. *IJECS*, 2388-2392.

Gotlieb, A. (2012). TCAS software verification using constraint programming. *The Knowledge Engineering Review, 27*(03), 343-360.

Grant, E. S., & Datta, T. (2016). Modeling RTCA DO-178C Specification to Facilitate Avionic Software System Design, Verification, and Validation. *International Journal of Future Computer and Communication, 5*(2), 120.

Gries, D., & Schneider, F. B. (2005). *An integrated approach to software engineering*. Kanpur, India: Pankaj Jalote. Indian Institute of Technology

Gross, H.-G. (2005). *Component-based software testing with UML*: Springer.

Gulia, P., & Chillar, R. S. (2012). A new approach to generate and optimize test cases for UML state diagram using genetic algorithm: http://doi.acm.org/10.1145/180921.2180933. *SIGSOFT Softw. Eng. Notes, 37*(3), 1-5. doi: 10.1145/180921.2180933

Gulia, P., & Chugh, J. (2015). Comparative analysis of traditional and object-oriented software testing. *ACM SIGSOFT Software Engineering Notes, 40*(2), 1-4.

Gupta, J. (2014). *An investigation of test cases generation from activity diagram.* Thapar University Patiala.

Håkansson, J., & Mokrushin, L. (2004). *An analysis tool for UML models with SPT annotations.* Paper presented at the Nordic Workshop on Programming Theory.

Hallowell, M. R., & Gambatese, J. A. (2009). Qualitative research: Application of the Delphi method to CEM research. *Journal of construction engineering and management, 136*(1), 99-107.

Han, S.-H., & Kwon, Y.-R. (2008). An empirical evaluation of test data generation techniques. *Journal of Computing Science and Engineering, 2*(3), 275-300.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of computer programming, 8*(3), 231-274.

Hartmann, J., Imoberdorf, C., & Meisinger, M. (2000). *UML-based integration testing.* Paper presented at the ACM SIGSOFT Software Engineering Notes.

Hashim, N. L., & Salman, Y. D. (2011). *An improved algorithm in test case generation from UML activity diagram using activity path.* Paper presented at the Proceedings of the 3rd International Conference on Computing and Informatics, ICOCI, Bandung, Indonesia

Hashmi, A., Goel, N., Goel, S., & Gupta, D. (2013). Firefly algorithm for unconstrained optimization. *IOSR J Comput Eng, 11*(1), 75-78.

Hedjazi, S. M., & Marjani, S. S. (2010). *Pruned genetic algorithm.* Paper presented at the International Conference on Artificial Intelligence and Computational Intelligence.

Hessel, A. (2006). *Model-based test case selection and generation for real-time systems.* (PhD Dissertation), Uppsala University.

Heumann, J. (2001). Generating test cases from use cases. *The rational edge, 6*(1).

Holbrook, A. L., Krosnick, J. A., Moore, D., & Tourangeau, R. (2007). Response order effects in dichotomous categorical questions presented orally: The impact of question and respondent attributes. *Public Opinion Quarterly, 71*(3), 325-348.

Hong, H. S., & Ural, H. (2004). *Using model checking for reducing the cost of test generation.* Paper presented at the International Workshop on Formal Approaches to Software Testing.

Hooda, I., & Chhillar, R. (2014). A review: study of test case generation techniques. *International Journal of Computer Applications, 107*(16), 33- 37.

Hopper, G. M. (1981). The first bug. *Annals of the History of Computing, 3*(3), 285-286.

Ibrar, M. (2013). UML diagrams: an aid to database design specification: a review. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*(3), 598 -602.

Inamdar, Y. (2015). Airport check-in of passenger. Retrieved Documents, 2016, from http://docslide.us/documents/airport-check-in-of-passenger.html

Ingle, S., & Mahamune, M. (2015). An UML based software automatic test case generation: survey. *International Research Journal of Engineering and Technology, 2*(2), 971-973.

Jain, E. S., & Sheikh, E. M. (2014). A novel test case generation method through metamorphic priority for 2-way testing method UMBCA implementation criteria. *International Journal of Engineering and Management Research, 4*(3), 157 -163.

Jain, R. (1990). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. New Jersey, United States: John Wiley & Sons.

Javed, A. Z., Strooper, P. A., & Watson, G. (2007). *Automated generation of test cases using model-driven architecture.* Paper presented at the Automation of Software Test, 2007. AST'07. Second International Workshop on Automation of Software Test.

Javed, M., Ahmad, B., Abbas, Z., Nawaz, A., Abid, M. A., & Ullah, I. (2012). Decreasing defect rate of test cases by designing and analysis for recursive modules of a program structure: Improvement in test cases. *International Journal of Computer Science and Information Security, 10*(8).

Jena, A. K., Swain, S. K., & Mohapatra, D. P. (2014). *A novel approach for test case generation from UML activity diagram.* Paper presented at the Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on.

Jena, A. K., Swain, S. K., & Mohapatra, D. P. (2015). Test case creation from UML sequence diagram: a soft computing approach *Intelligent Computing, Communication and Devices* (pp. 117-126): Springer.

Jia, X., & Liu, H. (2002). *Rigorous and automatic testing of web applications.* Paper presented at the Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA 2002).

Jia, X., Liu, H., & Qin, L. (2003). *Formal structured specification for web application testing.* Paper presented at the Midwest Software Engineering Conference.

Joo, S., Lin, S., & Lu, K. (2011). A usability evaluation model for academic library websites: efficiency, effectiveness and learnability. *Journal of Library and Information studies, 9*(2), 11-26.

Jorgensen, P. C. (2013). *Software testing: a craftsman's approach*. Boca Raton, Florida, United States: CRC press.

Jürjens, J. (2005). *Secure systems development with UML*. Berlin, Germany: Springer Science & Business Media.

Kaner, C., Falk, J., & Nguyen, H. Q. (1999). *Testing computer software*. India: Dreamtech Press.

Kaner, C., & Fiedler, R. L. (2013). *Foundations of Software Testing*. Massachusetts, United States: Context-Driven Press.

Kang, H., Lee, S., Lee, C., Yoon, C., & Shin, S. (2015). *SPIRIT: A framework for profiling SDN*. Paper presented at the Network Protocols (ICNP), 2015 IEEE 23rd International Conference on.

Kangas, K.-M. (2008). *Test automation of digital mammography device.* Helsinki Polytechnic Stadia.

Kansomkeat, S., Offutt, J., Abdurazik, A., & Baldini, A. (2008). *A comparative evaluation of tests generated from different UML diagrams.* Paper presented at the Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD'08. Ninth ACIS International Conference on.

Kansomkeat, S., & Rivepiboon, W. (2003). *Automated generating test case using UML statechart diagrams.* Paper presented at the Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology.

Kansomkeat, S., Thiket, P., & Offutt, J. (2010). *Generating test cases from UML activity diagrams using the condition-classification tree method.* Paper presented at the Software Technology and Engineering (ICSTE), 2010 2nd International Conference on.

Karambir, & Kaur, K. (2013). Performance analysis of Test Generation Techniques. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*(7), 490-498.

Karambir, & Kuldeep, K. (2013). Survey of software test case generation techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 937-942.

Kaur, A., & Harwinder, S. S. (2013). Automatic test case generation with SilK testing. *International Journal of Computer Applications, 79*(15), 32-34.

Kaur, G., & Singh, P. (2015). Test Case Generation Using UML Diagram. *International Journal of Emerging Technologies in Engineering Research, 1*(2), 23- 25.

Kaur, P., & Gupta, G. (2013). Automated model-based test path generation from UML diagrams via graph coverage techniques. *International Journal of Computer Science and Mobile Computing, 2*(7), 302-311.

Kavita, C., Shilpa, Yogita, G., Payal, R., & Akshath, G. (2015). A Survey Paper on Test Case Generation and Optimization: Cuckoo Search and Firefly Algorithm. *IJEDR, 3*(2), 584-589.

Keen, P. G. (1980). Decision support systems: a research perspective. *Decision Support Systems: Issues and Challenges*, 23-44.

Kelly, D. (1999). Software test automation and the product life cycle. *Mactech Magazine, 13*(10).

Kerlinger, F. N. (1986). *Foundations of behavioral research*. Orlando, Florida, United States: Holt, Rinehart and Winston.

Kernschmidt, K., & Vogel-Heuser, B. (2013). *An interdisciplinary SysML based modeling approach for analyzing change influences in production plants to support the engineering.* Paper presented at the Automation Science and Engineering (CASE), 2013 IEEE International Conference on.

Khandai, M., Acharya, A. A., & Mohapatra, D. P. (2011). A survey on test case generation from UML model. *International Journal of Computer Science and Information Technologies, 2*(3), 1164-1171.

Khurana, N., & Chillar, R. (2015). Test case generation and optimization using UML models and genetic algorithm. *Procedia Computer Science, 57*, 996-1004.

Kim, H., Kang, S., Baik, J., & Ko, I. (2007). *Test cases generation from UML activity diagrams.* Paper presented at the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007).

Kim, J. M., Porter, A., & Rothermel, G. (2005). An empirical study of regression test application frequency. *Software Testing, Verification and Reliability, 15*(4), 257-279.

Kim, S., Lively, W. M., & Simmons, D. B. (2006). *An Effort Estimation by UML Points in Early Stage of Software Development.* Paper presented at the Software Engineering Research and Practice.

Kim, W. Y., Son, H. S., & Kim, R. Y. C. (2011). A study on test case generation based on state diagram in modeling and simulation environment *Advanced Communication and Networking* (pp. 298-305): Springer.

Kim, Y. G., Hong, H. S., Bae, D.-H., & Cha, S.-D. (1999). Test cases generation from UML state diagrams *IEE Proceedings-Software, 146*(4), 187-192. doi: 10.1049/ip-sen:19990602

Knaak, N., & Page, B. (2005). *UML ¾ as a Modelling Language in Discrete Event Simulation.* Paper presented at the 9th european conference on modelling and simulation.

Komuro, M., & Komoda, N. (2008). *An explanation model for quality improvement effect of peer reviews.* Paper presented at the International Conference on Computational Intelligence for Modelling Control & Automation.

Konstantinou, P. (2013). Rapid application development. *Retrieved April, 1*.

Koong, C.-S., Shih, C., Hsiung, P.-A., Lai, H.-J., Chang, C.-H., Chu, W. C., . . . Yang, C.-T. (2012). Automatic testing environment for multi-core embedded software - ATEMES. *Journal of Systems and Software, 85*(1), 43-60.

Korel, B. (1990). Automated software test data generation. *IEEE Transactions on software engineering, 16*(8), 870-879. doi: 10.1109/32.57624

Kosindrdecha, N., & Daengdej, J. (2010). A test generation method based on state diagram. *Journal of Theoretical and Applied Information Technology*, 28-44.

Kot, M. (2003). The state explosion problem. *Retrieved May, 18*, 2015.

Krishnachandra, M. (2016). A customized approach for automated test case generation and optimization for system based software testing. *international Journal of Emerging Trends & Technology in Computer Science, 5*(2), 36-39.

Kulkarni, N. J., Naveen, K. V., Singh, P., & Srivastava, P. R. (2011). Test case optimization using artificial bee colony algorithm. *Advances in Computing and Communications*, 570-579.

Kull, A. (2009). *Model-based testing of reactive systems.* (PhD Dissertation), Tallinn University of Technology, Tallinn, Estonia.

Kumar, V. K., & Mathew, S. (2014). Compiler based test case generation. *International Journal on Recent Trends in Engineering & Technology, 11*(1), 558.

Kumaran, U. S., Kumar, S. A., & Kumar, K. V. (2011). An approach to automatic generation of test cases based on use cases in the requirements phase *International Journal on Computer Science and Engineering, 3*(1), 102-113.

Kundu, D., & Samanta, D. (2009). A novel approach to generate test cases from UML activity diagrams. *Journal of Object Technology, 8*(3), 65-83.

Kusumoto, S., Matukawa, F., Inoue, K., Hanabusa, S., & Maegawa, Y. (2005). Effort estimation tool based on use case points method. *Osaka University*.

Kwiecień, J., & Filipowicz, B. (2012). Firefly algorithm in optimization of queueing systems. *Bulletin of the Polish Academy of Sciences: Technical Sciences, 60*(2), 363-368.

Lam, S. S. B., Raju, M. H. P., Ch, S., & Srivastav, P. R. (2012). Automated generation of independent paths and test suite optimization using artificial bee colony. *Procedia Engineering, 30*, 191-200.

Lammich, P., & Neumann, R. (2015). *A framework for verifying depth-first search algorithms.* Paper presented at the Proceedings of the 2015 Workshop on Certified Programs and Proofs.

Lange, C. F., Chaudron, M. R., & Muskens, J. (2006). In practice: UML software architecture and design description. *IEEE software, 23*(2), 40-46. doi: 10.1109/MS.2006.50

Lauder, A., & Kent, S. (2001). Statecharts for Business Process Modeling *Enterprise Information Systems II* (pp. 121-125): Springer.

Lavagno, L., Markov, I. L., Martin, G., & Scheffer, L. K. (2016). *Electronic Design Automation for Ic System Design, Verification, and Testing*. United States: CRC Press.

Leitner, A., Oriol, M., Zeller, A., Ciupa, I., & Meyer, B. (2007). *Efficient unit test case minimization.* Paper presented at the Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering.

Li, B.-L., Li, Z.-s., Qing, L., & Chen, Y.-H. (2007). *Test case automate generation from UML sequence diagram and OCL expression.* Paper presented at the Computational Intelligence and Security, 2007 International Conference.

Li, H., & Lam, C. P. (2005). *An ant colony optimization approach to test sequence generation for state-based software testing.* Paper presented at the Quality Software, 2005.(QSIC 2005). Fifth International Conference

Li, L., He, T., & Wu, J. (2012). Automatic test Generation from UML statechart diagram based on euler circuit. *International Journal of Digital Content Technology & its Applications, 6*(19), 129-136.

Li, L., Li, X., He, T., & Xiong, J. (2013a). Extenics-based test case generation for UML activity diagram. *Procedia Computer Science, 17*, 1186-1193.

Li, L., Li, X., Tan, S., & Xiong, J. (2013b). Generating test cases from UML statechart diagram based on extended context-free grammars. *International Journal of Digital Content Technology and its Applications, 7*(5), 1206.

Lilly, R., & Uma, G. V. (2010). Reliable Mining of Automatically Generated Test Cases from Software Requirements Specification. *IJCSI international journal of computer science issues, 7*(1), 87-91.

Lima, B., & Faria, J. P. (2016). *A Survey on Testing Distributed and Heterogeneous Systems: The State of the Practice.* Paper presented at the International Conference on Software Technologies.

Linzhang, W., Jiesong, Y., Xiaofeng, Y., Jun, H., Xuandong, L., & Guoliang, Z. (2004). *Generating test cases from UML activity diagram based on Gray-box method*. Paper presented at the Software Engineering Conference.

Lithner, J. (2008). A research framework for creative and imitative reasoning. *Educational Studies in Mathematics, 67*(3), 255-276.

Liu, J., & Zhang, L. (2014). Using Formal Methods and Aspect Oriented Techniques to Model Cyber Physical Systems. *International Information Institute (Tokyo). Information, 17*(5), 1729.

Lu, M.-S., & Tseng, L.-K. (2010). An integrated object-oriented approach for design and analysis of an agile manufacturing control system. *The International Journal of Advanced Manufacturing Technology, 48*(9), 1107-1122.

Makker, V., & Singh, V. (2011). An Approach for Test Case Generation Using UML State chart Diagram. *International Journal of Advanced Research in Computer Science, 2*(5), 567 - 571.

Mala, D. J., Kamalapriya, M., Shobana, R., & Mohan, V. (2009). *A non-pheromone based intelligent swarm optimization technique in software test suite optimization.* Paper presented at the Intelligent Agent & Multi-Agent Systems, 2009. IAMA 2009. International Conference on.

Mala, D. J., & Mohan, V. (2009). ABC tester-artificial bee colony based software test suite optimization approach. *International Journal of Software Engineering, 2*(2), 15-43.

Mala, D. J., & Mohan, V. (2010). Quality improvement and optimization of test cases: a hybrid genetic algorithm based approach. *ACM SIGSOFT Software Engineering Notes, 35*(3), 1-14.

Mala, D. J., Ruby, E., & Mohan, V. (2012). A hybrid test optimization framework-coupling genetic algorithm with local search technique. *Computing and Informatics, 29*(1), 133-164.

Mall, R. (2009). *Fundamentals of software engineering*. New Delhi, India: PHI Learning Pvt. Ltd.

Mani, P., & Prasanna, M. (2016). Test Case Generation for Embedded System Software Using UML Interaction Diagram. *Journal of Engineering Science and Technology, 12*(4), 860 - 874.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision support systems, 15*(4), 251-266.

Marijuán, P. C., & Westley, J. (1992). Enzymes as molecular automata: a reflection on some numerical and philosophical aspects of the hypothesis. *BioSystems, 27*(2), 97-113.

Martin, J. (1991). *Rapid application development*. Basingstoke, United Kingdom: Macmillan Publishing Company.

Mathur, A. P. (2008). *Foundations of Software Testing, 2/e*. London, United Kingdom: Pearson Education.

McCaffrey, J. D. (2009). *Generation of pairwise test sets using a genetic algorithm.* Paper presented at the Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International.

McMinn, P. (2004). Search-based software test data generation: a survey. *Software Testing, Verification and Reliability, 14*(2), 105-156.

McQuillan, J. A., & Power, J. F. (2005). A survey of UML-based coverage criteria for software testing. *Department of Computer Science. NUI Maynooth, Co. Kildare, Ireland*, 1 -17.

Miller, T., Padgham, L., & Thangarajah, J. (2010). *Test coverage criteria for agent interaction testing.* Paper presented at the International Workshop on Agent-Oriented Software Engineering.

Mingsong, C., Qiu, X., Xu, W., Wang, L., Zhao, J., & Li, X. (2009). UML activity diagram-based automatic test case generation for Java programs. *THE COMPUTER JOURNAL, 52*(5), 545-556.

Mingsong, C., Xiaokang, Q., & Xuandong, L. (2006). *Automatic test case generation for UML activity diagrams.* Paper presented at the Proceedings of the 2006 international workshop on Automation of software test.

Mohamed, S. F. P. (2015). *A process based approach software certification model for agile and secure environment.* Universiti Utara Malaysia.

Mohi-Aldeen, S. M., Mohamad, R., & Deris, S. (2014). *Automatic test case generation for structural testing using negative selection algorithm.* Paper presented at the 1st International Conference of Recent Trends in Information and Communication Technologies.

Morell, L. J. (1984). *A Theory of Error-based Testing.* (PhD Dissertation), University of Maryland at College Park.

Morell, L. J. (1990). A theory of fault-based testing. *Software Engineering, IEEE Transactions on, 16*(8), 844–857.

Moret, B. M., & Shapiro, H. D. (2001). Algorithms and experiments: The new (and old) methodology. *Journal of Universal Computer Science, 7*(5), 434-446.

Muniz, L. L., Netto, U. S., & Maia, P. H. M. (2015). *TCG-a model-based testing tool for functional and statistical testing.* Paper presented at the ICEIS.

Murthy, P., Anitha, P., Mahesh, M., & Subramanyan, R. (2006). *Test ready UML statechart models.* Paper presented at the Proceedings of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools.

Na, H.-S., Choi, O.-H., & Lim, J.-E. (2006). *A metamodel-based approach for extracting ontological semantics from UML models*. Paper presented at the International Conference on Web Information Systems Engineering.

Naik, K., & Tripathy, P. (2011). *Software testing and quality assurance: theory and practice*. New Jersey, United States: John Wiley & Sons.

Nayak, A., & Samanta, D. (2010). Automatic test data synthesis using UML sequence diagrams. *Journal of Object Technology, 9*(2), 75-104.

Ngah, A. (2012). *Regression test selection by exclusion.* (PhD thesis), Durham University.

Nidhra, S., & Dondeti, J. (2012). Blackbox and whitebox testing techniques-a literature review. *International Journal of Embedded Systems and Applications (IJESA), 2*(2), 29-50.

Norshuhada, S., & Shahizan, H. (2010). Design research in software development: Constructing and linking research questions, objectives, methods and outcomes: Sintok: Penerbit Universiti Utara Malaysia.

O'Neil, D. (2001). *Peer reviews, encyclopedia of software engineering*. New York, United States: Wiley.

Offermann, P., Levina, O., Schönherr, M., & Bub, U. (2009). *Outline of a design science research process.* Paper presented at the Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology.

Offutt, A. J. (1988). *Automatic test data generation.* (PhD), Georgia Institute of Technology, Atlanta, GA, USA.

Offutt, J., & Abdurazik, A. (1999). *Generating tests from UML specifications*: Springer.

Offutt, J., Liu, S., Abdurazik, A., & Ammann, P. (2003). Generating test data from state-based specifications. *Software Testing, Verification and Reliability, 13*(1), 25-53.

Oladejo, B. F., & Ogunbiyi, D. T. (2014). An empirical study on the effectiveness of automated test case generation techniques. *American Journal of Software Engineering and Applications, 3*(6), 95-101.

Olson, K. (2010). An examination of questionnaire evaluation by expert reviewers. *Field Methods, 22*(4), 295-318.

Oluwagbemi, O., & Asmuni, H. (2014). Development of a robust parser for extracting artifacts during model-based testing from UML diagrams. *International Journal of Software Engineering and Technology, 1*(2), 43-50.

Oluwagbemi, O., & Asmuni, H. (2015). Automatic generation of test cases from activity diagrams for UML based testing (UBT). *Jurnal Teknologi, 77*(13).

Omotunde, H., Ibrahim, R., Ahmed, M., Olanrewaju, R., Ibrahim, N., & Shah, H. (2016). A framework to reduce redundancy in android test suite using

refactoring. *Indian Journal of Science and Technology, 9*(46). doi: 10.17485/ijst/2016/v9i46/107107

Ooi, W., Shahrizal, I., Noordin, A., Nurulain, M., & Norhan, M. (2014). *Development of rural emergency medical system (REMS) with geospatial technology in Malaysia.* Paper presented at the IOP Conference Series: Earth and Environmental Science.

Ostrand, T. J., & Balcer, M. J. (1988). The category-partition method for specifying and generating fuctional tests. *Communications of the ACM, 31*(6), 676-686.

Pahwa, N., & Solanki, K. (2014). UML based test case generation methods: a review. *International Journal of Computer Applications, 95*(20), 1-6.

Pandey, B., & Jain, R. (2014). Importance of unified modelling language for test case generation in software testing. *Rituraj Jain et al, Int.J.Computer Technology & Applications, 5*(2), 345-350.

Panthi, V., & Mohapatra, D. (2015). Generating prioritized test sequences using Firefly optimization technique *Computational Intelligence in Data Mining-Volume 2* (pp. 627-635): Springer.

Panthi, V., & Mohapatra, D. P. (2012). *Automatic test case generation using sequence diagram.* Paper presented at the Proceedings of International Conference on Advances in Computing.

Parnami, S. (2013). Testing target path by automatic generation of test data using genetic algorithm. *International Journal of Information and Computation Technology, 3*(8), 825-832.

Patnaik, D., Acharya, A. A., & Mohapatra, D. P. (2011). Generating testcases for concurrent systems using UML state chart diagram *Information Technology and Mobile Communication* (pp. 100-105): Springer.

Patwa, S., & Malviya, A. K. (2014). Impact of coding phase on object oriented software testing. *Covenant Journal of Informatics and Communication Technology (CJICT), 2*(1), 57-67.

Paul, A., & Jeff, O. (2008). *Introduction to Software Testing*. New York, United States: Cambridge University Press.

Perry, W. E. (2007). *Effective Methods for Software Testing: Includes Complete Guidelines, Checklists, and Templates*. New Jersey, United States: John Wiley & Sons.

Pilskalns, O., Andrews, A., Ghosh, S., & France, R. (2003). Rigorous testing by merging structural and behavioral UML representations *The Unified Modeling Language. Modeling Languages and Applications* (pp. 234-248): Springer.

Pimenta, A. (2006). *Automated specification based testing of graphical user interfaces.* (PhD Thesis), Porto University, Porto, Portugal.

Pinheiro, A. C., Simão, A., & Ambrosio, A. M. (2014). FSM-based test case generation methods applied to test the communication software on board the ITASAT University Satellite: A Case Study. *Journal of Aerospace Technology and Management, 6*(4), 447-461.

Popp, R., Falb, J., Arnautovic, E., Kaindl, H., Kavaldjian, S., Ertl, D., . . . Bogdan, C. (2009). *Automatic generation of the behavior of a user interface from a high-level discourse model.* Paper presented at the System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on.

Prasanna, M., & Chandran, K. (2011). Automated Test Case Generation for Object Oriented Systems Using UML Object Diagrams *High Performance Architecture and Grid Computing* (pp. 417-423): Springer.

Prasanna, M., Chandran, K., & Suberi, D. B. (2011). Automatic test case generation for UML class diagram using data flow approach. *Academia Education*, 1-7.

Prasanna, M., Sivanandam, S., Venkatesan, R., & Sundarrajan, R. (2005). A survey on automatic test case generation. *Academic Open Internet Journal, 15*(6).

Presser, S., & Blair, J. (1994). Survey pretesting: Do different methods produce different results? *Sociological methodology, 24*, 73-104.

Priya, S. S., & Sheba, P. (2013). *Test case generation from UML models-a survey.* Paper presented at the Proc. International Conference on Information Systems and Computing (ICISC-2013), INDIA.

Rafi, D. M., Moses, K. R. K., Petersen, K., & Mäntylä, M. V. (2012). *Benefits and limitations of automated software testing: Systematic literature review and practitioner survey.* Paper presented at the Proceedings of the 7th International Workshop on Automation of Software Test.

Rapos, E. (2012). *Understanding the effects of model evolution through incremental test case generation for UML-RT models.* Queen's University.

Rhmann, W., & Saxena, V. (2016). Optimized and prioritized test paths generation from UML activity diagram using firefly algorithm. *International Journal of Computer Applications, 145*(6), 16-22.

Rice, R. W. (2010). STBC: the economics of testing. http://www.riceconsulting.com/public_pdf/STBC-WM.pdf.

Robinson, H. (1999). *Graph theory techniques in model-based testing.* Paper presented at the International Conference on Testing Computer Software.

Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. *IEEE Transactions on software engineering, 27*(10), 929-948.

Rumbaugh, J., Jacobson, I., & Booch, G. (2004). *Unified Modeling Language Reference Manual, The*: Pearson Higher Education.

Rungi, K., & Matulevičius, R. (2013). Empirical analysis of the test maturity model integration (TMMi) *Information and Software Technologies* (pp. 376-391): Springer.

Ruohonen, K. (2013). *Graph theory*. Tampere, Finland: Tampere University of Technology.

Sahoo, R. K., Mohapatra, D. P., & Patra, M. R. (2016a). A firefly algorithm based approach for automated generation and optimization of test cases. *International Journal on Computer Science and Engineering, 4*(8), 54-58.

Sahoo, R. K., Ojha, D., Mohapatra, D. P., & Patra, M. R. (2016b). Automated test case generation and optimization: a comparative review. *International Journal of Computer Science & Information Technology, 8*(5), 19-32.

Saifan, A. A., & Mustafa, W. B. (2015). Using formal methods for test case generation according to transition-based coverage criteria. *Jordanian Journal of Computers and Information Technology, 1*(1), 15-30.

Saini, E. S., & Srivastava, E. V. (2015). Case Generation from the Combination of UML Class and Activity Diagrams. *International Journal Of Modern Engineering Research, 5*(7), 10-13.

Salah, D., Paige, R., & Cairns, P. (2014). *An evaluation template for expert review of maturity models.* Paper presented at the International Conference on Product-Focused Software Process Improvement.

Salman, Y. D., & Hashim, N. L. (2014). An improved method of obtaining basic path testing for test case based on UML state chart. *Science International, 26*(4), 1607 - 1610.

Salman, Y. D., & Hashim, N. L. (2016). Automatic Test Case Generation from UML State Chart Diagram: A Survey *Advanced Computer and Communication Engineering Technology* (pp. 123-134): Springer.

Salman, Y. D., & Hashim, N. L. (2017). Test Case Generation Model for UML Diagrams. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 9*(2-2), 171-175.

Salman, Y. D., Hashim, N. L., Rejab, M. M., Romli, R., & Mohd, H. (2017). Coverage Criteria for UML State Chart Diagram in Model-based Testing. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 9*(2-11), 85-89.

Samuel, P., Mall, R., & Bothra, A. K. (2008). Automatic test case generation using unified modeling language (UML) state diagrams. *IET software, 2*(2), 79-93.

Santiago, V., do Amaral, A. S. M., Vijaykumar, N., Mattiello-Francisco, M. F., Martins, E., & Lopes, O. C. (2006). *A practical approach for automated test case generation using statecharts.* Paper presented at the Computer Software

and Applications Conference, 2006. COMPSAC'06. 30th Annual International.

Santiago, V., Vijaykumar, N. L., Guimarães, D., Amaral, A. S., & Ferreira, É. (2008). *An environment for automated test case generation from statechart-based and finite state machine-based behavioral models.* Paper presented at the Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on.

Santosh, M., & Singh, R. (2013). Test Case Minimization By Generating Requirement Based Mathematical Equations. *International Journal of Engineering Research & Technology (IJERT), 2*(6), 1180 - 1188.

Sapna, P., & Mohanty, H. (2008). *Automated Scenario Generation Based on UML Activity Diagrams*. Paper presented at the International Conference on Information Technology.

Schligloff, H., & Roggenbach, M. (2002). Path testing. *Advanced Topics in Computer Science*.

Schroeder, P. J., Kim, E., Arshem, J., & Bolaki, P. (2003). *Combining behavior and data modeling in automated test case generation.* Paper presented at the Quality Software, 2003. Proceedings. Third International Conference on.

Schwarzl, C., & Peischl, B. (2010a). Static-and dynamic consistency analysis of UML state chart models *Model Driven Engineering Languages and Systems* (pp. 151-165): Springer.

Schwarzl, C., & Peischl, B. (2010b). *Test sequence generation from communicating UML state charts: An industrial application of symbolic transition systems.* Paper presented at the Quality Software (QSIC), 2010 10th International Conference.

Schweighofer, T., & Heričko, M. (2014). *Approaches for test case generation from UML diagrams.* Paper presented at the Third Workshop on Software Quality Analysis, Monitoring, Improvement and Applications.

Shah, S. A. A., Shahzad, R. K., Bukhari, S. S. A., Minhas, N. M., & Humayun, M. (2016). A Review of Class Based Test Case Generation Techniques. *Journal of Software, 11*(5), 464-480.

Shahzad, A., Raza, S., Azam, M. N., Bilal, K., & Shamail, S. (2009). *Automated optimum test case generation using web navigation graphs.* Paper presented at the Emerging Technologies, 2009. ICET 2009. International Conference on.

Shamshiri, S., Just, R., Rojas, J. M., Fraser, G., McMinn, P., & Arcuri, A. (2015). *Do automatically generated unit tests find real faults? an empirical study of effectiveness and challenges (t).* Paper presented at the Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on.

Shamsoddin-Motlagh, E. (2012). A review of automatic test cases generation. *International Journal of Computer Applications, 57*(13), 25 - 29.

Shanthi, A., & Kumar, G. M. (2012). Automated test cases generation from UML sequence diagram. *International Proceedings of Computer Science & Information Technology, 41*, 83 -89.

Sharma, P. (2014). Automated software testing using metahurestic technique based on improved ant algorithms for software testing. *International Journal on Recent and Innovation Trends in Computing and Communication, 2*(11).

Sharma, R., & PrakashSonwani, S. (2015). Programmed test case generation from simulink/stateflow model. *Indian Journal of Computer Science and Engineering (IJCSE), 6*(2), 45 - 51.

Sharp, H., Rogers, Y., & Preece, J. (2007). Interaction design: beyond human-computer interaction. *netWorker: The Craft of Network Computing, 11*(4), 34.

Shen, J., & Abraham, J. A. (2000). An RTL abstraction technique for processor microarchitecture validation and test generation. *Journal of Electronic Testing, 16*(1), 67-81.

Sherwood, C., & Rout, T. (1998). *A structured methodology for multimedia product and systems development.* Paper presented at the ASCILITE.

Shinde, V. (2013). *Software testing career package - a software tester's journey from getting a job to becoming a test leader!* : Software Testing Help.

Shiratuddin, N., Hassan, S., Hashim, N. L., Sarif, S. M., Bakar, A., & Shahbani, M. (2013). Focus group evaluation on IPTComKitTM commercialization model. *Recent Advances in Electrical and Computer Engineering*, 90-95.

Shirole, M., & Kumar, R. (2010). A hybrid genetic algorithm based test case generation using sequence diagrams *Contemporary Computing* (pp. 53-63): Springer.

Shirole, M., & Kumar, R. (2013). UML Behavioral Model Based Test Case Generation: A Survey. *ACM SIGSOFT Software Engineering Notes, 38*(4), 1-13.

Shirole, M., Suthar, A., & Kumar, R. (2011). *Generation of improved test cases from UML state diagram using genetic algorithm.* Paper presented at the Proceedings of the 4th India Software Engineering Conference.

Shneiderman, S. B., & Plaisant, C. (2005). *Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th Edition).* United States: Pearson Addison Wesley.

Shull, F., Rus, I., & Basili, V. (2000). How perspective-based reading can improve requirements inspections. *Computer, 33*(7), 73-79.

Singh, R. (2014). *Test case generation for object-oriented systems: A review.* Paper presented at the Communication Systems and Network Technologies (CSNT), 2014 Fourth International Conference on.

Singh, S., & Shree, R. (2016). A combined approach to optimize the test suite size in regression testing. *CSI transactions on ICT, 4*(2-4), 73-78.

Society, I. C. (2014). *Guide to the software engineering body of knowledge (SWEBOK Version 3)*: IEEE.

Sommerville, I. (2011). *Software engineering* (9th ed.). Massachusetts, United States: Addison-Wesley.

Sood, B., & Rattan, D. (2016). An efficient method to generate automation scripts using selenium tool. *An International Journal of Engineering Sciences, 17*, 1 - 7.

Specification, O. A. (2007). OMG unified modeling language (OMG UML), Superstructure, V2. 1.2. *Object Management Group, 2*(12).

Sprague Jr, R. H., & Carlson, E. D. (1982). *Building effective decision support systems*. New Jersey, United States: Prentice Hall Professional Technical Reference.

Srikant, Y., & Shankar, P. (2007). *The compiler design handbook: optimizations and machine code generation*. Cambridge, United Kingdom: CRC Press.

Srivastav, S., & Gupta, S. (2016). Software design pattern static validation using cyclomatic complexity and UML approach. *International Journal, 4*(7), 89 - 97.

Srivastava, P. R., Baby, K., & Raghurama, G. (2009). *An approach of optimal path generation using ant colony optimization.* Paper presented at the TENCON 2009-2009 IEEE Region 10 Conference.

Srivastava, P. R., & Kim, T.-h. (2009). Application of genetic algorithm in software testing. *International Journal of Software Engineering and Its Applications, 3*(4), 87-96.

Srivatsava, P. R., Mallikarjun, B., & Yang, X.-S. (2013). Optimal test sequence generation using firefly algorithm. *Swarm and Evolutionary Computation, 8*, 44-53.

Srividhya, J., & Alagarsamy, K. (2014). A synthesized overview of test case optimization techniques. *Journal of Recent Research in Engineering and Technology, 1*(2).

Stecklein, J., Dabney, J., Dick, B., Haskins, B., Lovell, R., & Moroney, G. (2004). Error cost escalation through the project life cycle. *National Aeronautics and Space Administration*.

Stewart, D. W., & Shamdasani, P. N. (2014). *Focus groups: Theory and practice* (Vol. 20). California, United States: Sage Publications.

Sumalatha, V. M., & Raju, G. (2014). Model based test case optimization of UML activity diagrams using evolutionary algorithms. *Model Based Test Case Optimization of UML Activity Diagrams using Evolutionary Algorithms, 12*(11), 131-142.

Sung, P. W.-B., & Paynter, J. (2006). *Software testing practices in New Zealand.* Paper presented at the Proceedings of the 19th Annual Conference of the National Advisory Committee on Computing Qualifications.

Suri, B., Mangal, I., & Srivastava, V. (2011). Regression test suite reduction using an hybrid technique based on BCO and genetic algorithm. *Special Issue of International Journal of Computer Science & Informatics (IJCSI), 2*(2), 2231-5292.

Swain, R. K., Behera, P. K., & Mohapatra, D. P. (2012a). Generation and optimization of test cases for object-oriented software using state chart diagram. *International Journal*, 407- 424.

Swain, R. K., Behera, P. K., & Mohapatra, D. P. (2012b). Minimal testcase generation for object-oriented software with state charts. *International Journal of Software Engineering & Applications (IJSEA), 3*(4).

Swain, R. K., Panthi, V., Behera, P., & Mohapatra, D. (2012c). Automatic test case generation from UML state chart diagram. *International Journal of Computer Applications, 42*(7), 26-36.

Swain, S. K., Mohapatra, D. P., & Mall, R. (2010a). Test case generation based on state and activity models. *Journal of Object Technology, 9*(5), 1-27.

Swain, S. K., Mohapatra, D. P., & Mall, R. (2010b). Test case generation based on use case and sequence diagram. *International Journal of Software Engineering, 3*(2), 21-52.

Tan, R. P. (2003). *Programming language support for automated testing.* (PhD Dissertation), Virginia Tech.

Tewari, A., & Misra, A. K. (2015). An approach to Model Based Test case generation for Student Admission Process. *International Journal of Innovative Science, Engineering & Technology, 2*(10), 818 -825.

Theis, B., Frood, J., Nishri, D., & Marrett, L. D. (2002). Evaluation of a risk factor survey with three assessment methods. *Chronic Diseases and Injuries in Canada, 23*(1), 1 - 47.

Tomar, A., & Singh, P. (2016). Software testing with different optimization techniques. *International Journal of Emerging Technology and Advanced Engineering, 6*(4), 169-171.

Tripathy, A., & Mitra, A. (2012). *Test case generation using activity diagram and sequence diagram.* Paper presented at the International Conference on Advances in Computing.

Tsumaki, T., & Morisawa, Y. (2000). *A framework of requirements tracing using UML*. Paper presented at the Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific.

UML, O. M. G. (2004). UML 2.0 Infrastructure Specification. *OMG, Needham*.

Utting, M., & Legeard, B. (2007). *Practical model-based testing: a tools approach*. San Francisco, United States: Morgan Kaufmann.

Utting, M., & Legeard, B. (2010). *Practical model-based testing: a tools approach*. San Francisco, United States: Morgan Kaufmann.

Utting, M., Pretschner, A., & Legeard, B. (2006). A Taxonomy of Model-based Testing *Technical report*. Hamilton, New Zealand: The University of Waikato.

Vaziri, R., & Mohsenzadeh, M. (2012). A questionnaire-based data quality methodology. *International Journal of Database Management Systems, 4*(2), 55.

Verma, A., & Dutta, M. (2014). Automated Test case generation using UML diagrams based on behavior. *International Journal of Innovations in Engineering and Technology (IJIET), 4*(1), 31 - 39.

Vernotte, A., Dadeau, F., Lebeau, F., Legeard, B., Peureux, F., & Piat, F. (2014). *Efficient Detection of Multi-step Cross-Site Scripting Vulnerabilities*. Paper presented at the 10th International Conference on Information Systems Security Hyderabad, India.

Voloshin, V. I. (2009). *Introduction to graph theory*. New York, United States: Nova Science Publishers.

Waller, M. P., Dresselhaus, T., & Yang, J. (2013). JACOB: an enterprise framework for computational chemistry. *Journal of computational chemistry, 34*(16), 1420-1428.

Wei, Z., & Xiaoxue, W. (2010). *Graph theory model based automatic test platform design.* Paper presented at the Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on.

Weilkiens, T. (2011). *Systems engineering with SysML/UML: modeling, analysis, design*. Massachusetts, United States: Morgan Kaufmann.

Weißleder, S. (2010). *Test models and coverage criteria for automatic model-based test generation with UML state machines.* Humboldt University of Berlin.

Weißleder, S., & Sokenou, D. (2010). ParTeG-a Model-Based Testing tool. *Softwaretechnik-Trends, 30*(2), 1 -2

Werner, E., & Grabowski, J. (2012). Mining test cases: optimization possibilities. *International Journal On Advances in Software, 5*(3 and 4), 200-211.

Wiegers, K., & Beatty, J. (2013). *Software requirements*. London, United Kingdom: Pearson Education.

Wiegers, K. E. (2002a). *Peer reviews in software: A practical guide*. Boston, United State: Addison-Wesley Boston.

Wiegers, K. E. (2002b). Seven truths about peer reviews. *Cutter IT Journal, 15*(7), 31-37.

Wu, Y.-C., & Fan, C.-F. (2014). Automatic test case generation for structural testing of function block diagrams. *Information and Software Technology, 56*(10), 1360-1376.

Xiong, J. (2011). *New software engineering paradigm based on complexity science: an introduction to NSE*. Berlin, Germany: Springer Science & Business Media.

Xu, S., Chen, L., Wang, C., & Rud, O. (2016). *A comparative study on black-box testing with open source applications.* Paper presented at the Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2016 17th IEEE/ACIS International Conference on.

Xu, Z., Kim, Y., Kim, M., Rothermel, G., & Cohen, M. B. (2010). *Directed test suite augmentation: techniques and tradeoffs.* Paper presented at the Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering.

Yadav, K., Patel, S., Arora, T., Uptu, U. P., & Jnu, J. (2016). Challenges in automatic test case generation. *International Journal of Communications, 1*, 99-102.

Yan, X.-T., Jiang, C., & Eynard, B. (2008). *Advanced design and manufacture to gain a competitive edge*: Springer.

Yang, X.-S. (2010). *Nature-inspired metaheuristic algorithms*: Luniver press.

Yang, X.-S., & He, X. (2013). Firefly algorithm: recent advances and applications. *International Journal of Swarm Intelligence, 1*(1), 36-50.

Yemul, M. S., Vhatkar, K., & Bag, V. (2014). Testing approach for automatic test case generation and Optimization using GA. *international Journal of Emerging Trends & Technology in Computer Science, 3*(5), 69 - 71.

Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability, 22*(2), 67-120.

Yu, Z., Martinez, M., Danglot, B., Durieux, T., & Monperrus, M. (2017). Test case generation for program repair: a study of feasibility and effectiveness. *ArXiv e-prints*, 1 -12.

Zaibon, S. B., & Shiratuddin, N. (2010). *Mobile game-based learning (mGBL) engineering model as a systematic development approach.* Paper presented at the Global Learn.

Zelkowitz, M. V., & Wallace, D. R. (1998). Experimental models for validating technology. *Computer, IEEE, 31*(5), 23-31. doi: 10.1109/2.675630

Zhang, C., Duan, Z., Yu, B., Tian, C., & Ding, M. (2016). A Test Case Generation Approach Based on Sequence Diagram and Automata Models. *Chinese Journal of Electronics, 25*(2), 234-240.

Zhang, W., & Liu, S. (2013). Supporting tool for automatic specification-based test case generation *Structured Object-Oriented Formal Language and Method* (pp. 12-25): Springer.

Zhu, H., Hall, P. A., & May, J. H. (1997). Software unit test coverage and adequacy. *ACM Computing Surveys (CSUR), 29*(4), 366-427.

Appendix A

EXPERT EVALUATION FORM

EXPERT EVALUATION FORM

Dear Respected Respondent,

My name is Yasir Dawood Salman and I am currently pursuing my Ph.D. in Information Technology (IT). I am specializing in Software Testing at the School of Computing, College of Arts and Sciences, Universiti Utara Malaysia (UUM). My Ph.D. research entitled An Automated Test Case Generation Model for UML Statechart Diagram aims to develop a model and algorithms that can automatically generate test case from the UML statechart diagram.

Expert review is the verification method selected to evaluate this study. This study seeks your expertise in evaluating the proposed work. The information supplied will be treated as confidential and will be used for research purposes only and may be reported anonymously in academic publications. I humbly solicit for your kind assistance to participate in this research.

The main purpose of this verification is to verify the proposed model and its components, as well as other entities within the model, possesses a satisfactory range of accuracy, completeness, and consistency.

Kindly attach a copy of your CV after completion of this verification form for the proper documentation of this research.

If you have any questions regarding this research, please feel free to contact me by e-mail at yasir.dawod@gmail.com, phone number (+60169790922), or through my supervisor Dr. Nor Laily Hashim at laily@uum.edu.my.

Thank you for your time and assistance.

Instructions:

Please read the system review documents provided to you and go through the model, algorithms, and prototype carefully. Once this is done, please tick (✓) the most appropriate answer. You are advised to answer the questions based on your knowledge and experience and verify the items in Section B. This section on software quality dimensions is used to measure the originality and validity of the proposed system implementation for automatic test case generation of the UML statechart diagram. Section A is expert profile. This questionnaire is NOT intended to assess people, their work, or knowledge. Completing the questionnaire will take around 30–45 minutes. I will deeply appreciate if you could answer the questions carefully as the information you provide will influence the accuracy and success of this research.

Section A: Expert Profile

Name (First and Last) ……...

Employer/ Facility ……...

Position [ ] Professor [ ] Associate Professor [ ] Senior Lecturer [ ] Lecturer [ ] Others (Please specify)……...

……...

Fields of ……...

Specialization ……...

……...

Years of Experience in:

Algorithms	Software Development	Software Engineering	Software Testing
……	……	……	……

Research Interests ……...

……...

……...

E-mail ……...

Office Phone …… Mobile Phone ……...

Section B: Items for Review

Please validate and give comments on the below mentioned dimensions on the proposed system (framework, algorithms and prototype) implementation for an automatic test case generation:

DIMENSIONS	DESCRIPTIONS	COMMENTS/SUGGESTIONS
Practicality	The proposed framework of automatic test case generation from UML diagrams can practically be implemented in the real world.	☐ Agree ☐ Disagree Comments/ Suggestions: ---------- ---------- ----------
Clarity	As a whole, the framework is workable and the steps in the framework are easily followed.	☐ Agree ☐ Disagree Comments/ Suggestions: ---------- ---------- ----------
Completeness	The essential items of the proposed framework are complete, satisfactory, and suitable to generate test cases.	☐ Agree ☐ Disagree Comments/ Suggestions: ---------- ---------- ---------- ----------

Correctness	The algorithms: State Relationships Table (SRT), Test Cases Paths Generation (TCGP), minimization, prioritization, and Test Cases Generation (TCG), provide correct results and achieve its objectives.	☐ Agree ☐ Disagree Comments/ Suggestions: --- --- ---
Effectiveness	The prototype automatically generates the test cases from UML statechart diagram, for which it is intended.	☐ Agree ☐ Disagree Comments/ Suggestions: --- --- --- ---
Accuracy	The system provides correct test cases result to the inputted UML statechart diagram.	☐ Agree ☐ Disagree Comments/ Suggestions: --- --- --- ---
Perceived Usefulness	The proposed system is useful for the software tester in improving the coverage criteria quality of test case generation.	☐ Agree ☐ Disagree Comments/ Suggestions: --- --- ---

Usability	Using the proposed system would make generating the test cases easy for the software tester.	☐ Agree ☐ Disagree Comments/ Suggestions: ---------- ---------- ---------- ----------
Understand-ability	All documentations are clearly and simply written such that procedures, rules, and algorithms are readable and can be easily understood.	☐ Agree ☐ Disagree Comments/ Suggestions: ---------- ---------- ---------- ----------

Additional comments (if any):

..……..

……

……

……

Thank you.

…… Date……

(Signature & Official Stamp)

Appendix B

CONSENT FOR PARTICIPATION IN EXPERT VERIFICATION

I volunteer to participate in a research project conducted by Yasir Dawood Salman, Ph.D. student, in Information Technology (IT), School of Computing, College of Arts and Sciences, Universiti Utara Malaysia (UUM).

I understand that the expert verification form is designed to evaluate the proposed framework, algorithms, and prototype. I will be one of approximately eight people being interviewed for this research.

1. My participation in this project is voluntary. I may withdraw and discontinue participation at any time. If I decline to participate or withdraw from the study, no one on my campus will be told.
2. The interview will last approximately 30-45 minutes. Notes will be written during the interview. An audio tape of the interview and subsequent dialogue will be make. If I do not want to be taped, I will need to inform in advance.
3. I understand that the researcher will not identify me by name in any reports using information obtained from this interview, and that my confidentiality as a participant in this study will remain secure. Subsequent uses of records and data will be subject to standard data use policies, which protect the anonymity of individuals and institutions.
4. I have read and understand the explanation provided to me. I have had all my questions answered to my satisfaction, and I voluntarily agree to participate in this study.
5. I have been given a copy of this consent form.

Name of Participant	Date	Signature
Name of Researcher	Date	Signature

Appendix C

DETAILED MINIMIZATION AND PRIORITIZATION FOR SELECTED EXAMPLES

Section A: UML Statechart Diagram of an Online Shop

The process of minimize and prioritize the UML statechart diagram of an online shop example (see Section 5.2.2.1.2) is shown below.

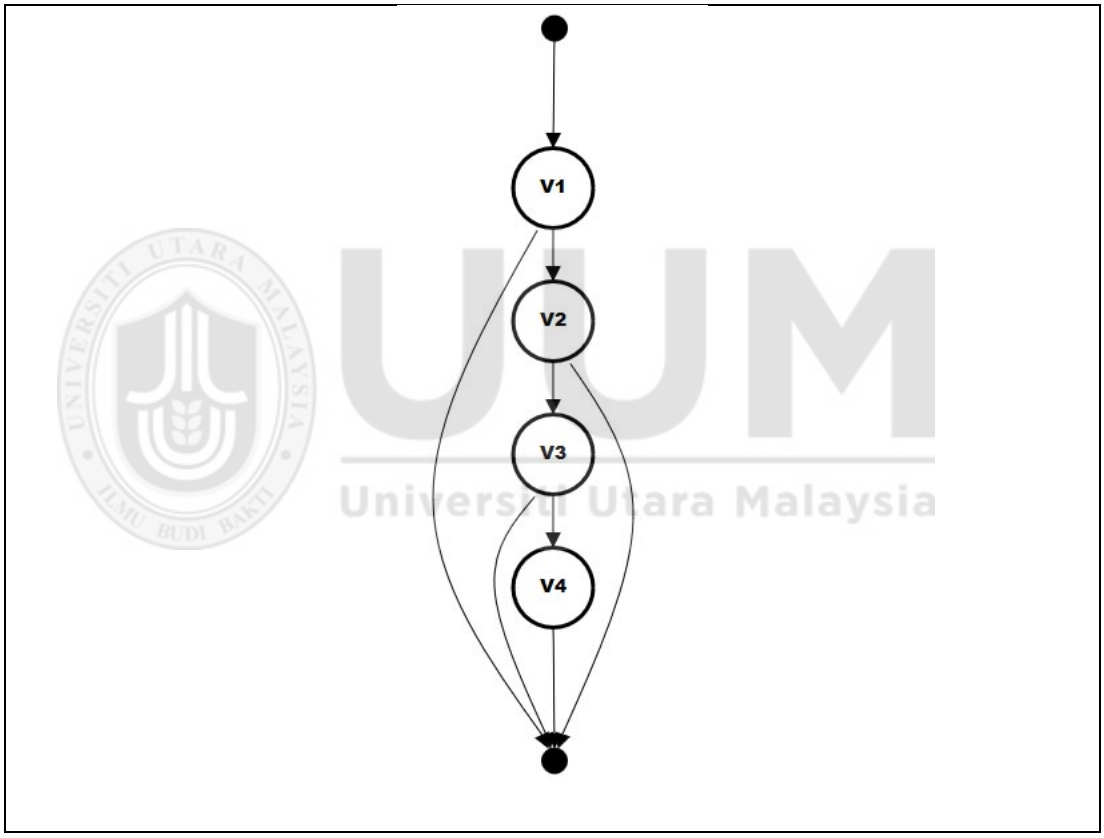

Figure B.1. Chart Relationship Graph for the UML Statechart Diagram of an Online Shop

The intermediate graph (Figure B.1) was converted to test paths using TCGP algorithm, and all the possible generated test paths from the intermediate graph is shown in Figure B.2.

TP 1: [S → 1 → 2 → 3 → 4 → E]
TP 2: [S → 1 → E]
TP 3: [S → 1 → 2 → E]
TP 4: [S → 1 → 2 → 3 → E]

Figure B.2 All Possible Test Paths for the UML Statechart Diagram of an Online Shop

Path weight was calculated for each tests path using Equation 4.7, as shown in Table B.1, to determine each path weight of transactions in the system

Table B.1

Path Weight for Each Path for the UML Statechart Diagram of an Online Shop

TC	S→1	1→2	1→E	2→3	2→E	3→4	3→E	4→E	E	W_v
1	1	1	0	1	0	1	0	1	5	0.83
2	1	0	1	0	0	0	0	0	2	0.66
3	1	1	0	0	1	0	0	0	3	0.75
4	1	1	0	1	0	0	1	0	4	0.8

After generate the path weight, next step start by calculate path coverage for each single path as shown in Table B.2.

Table B.2

Coverage Criteria for Each Path for the UML Statechart Diagram of an Online Shop

TP No	All-State	All-Transition	All-Transition-pairs	All-One-loop-paths
1	100%	62%	50%	-
2	50%	25%	16%	-
3	66%	37%	33%	-
4	83%	50%	50%	-

After generate the path weight and coverage criteria for each path the intermediate graph is converted to adjacency matrix, as showing in Table B.3. Then, this matrix is used to generate the guidance matrix for the graph.

Table B.3

Adjacency Matrix for the UML Statechart Diagram of an Online Shop

States	**0**	**1**	**2**	**3**	**4**	**5**
0	0	1	0	0	0	0
1	0	0	1	0	0	1
2	0	0	0	1	0	1
3	0	0	0	0	1	1
4	0	0	0	0	0	1
5	0	0	0	0	0	0

After creating adjacency matrix, it is then used to generate guidance matrix. In the example in Figure 1, the number of vertices is 6, and the number of edges is 8, therefore the Cyclomatic Complexity equal to 4. However, the Cyclomatic Complexity for each vertex need to be calculated using Equation 4.9 to be used to calculate the guidance value using Equation 4.8. The results are shown in Table 4.

Table B.4

Guidance Value for the UML Statechart Diagram of an Online Shop

States	**Cyclomatic Complexity CC**	**guidance value GF**
0	4	196
1	4	156
2	3	87
3	2	38
4	1	9
5	1,000 [END vertex infinity]	1,000 [finial state]

Guidance matrix (Table B.5) is just as a look-up/decision table of adjacency matrix with each guidance value corresponding to every edge.

Table B.5

Guidance Matrix for the UML Statechart Diagram of an Online Shop

States	0	1	2	3	4	5
0	0	156	0	0	0	0
1	0	0	87	0	0	1000
2	0	0	0	38	0	1000
3	0	0	0	0	9	1000
4	0	0	0	0	0	1000
5	0	0	0	0	0	0

Then the algorithm will generate the path sequences as:

Path 1= [0, 1, 2, 3, 4, 5],
Path 2= [1, 5],
Path 3= [2, 5],
Path 3= [3, 5].

To optimize the test cases, the algorithm will match each optimal path with paths in Figure B.2, and chose the lowest path weight W_v between the selected paths match paths. The minimized test paths are shown in Figure B.3

TP 1: [S → 1 → 2 → 3 → 4 → E]
TP 2: [S → 1 → E]
TP 3: [S → 1 → 2 → E]
TP 4: [S → 1 → 2 → 3 → E]

Figure B.3. Optimized Test Paths for the UML Statechart Diagram of an Online Shop

The combination use of these three paths lead to achieving: all-state coverage, all-transition coverage and, all-transition-pairs coverage as shown in Table B.6.

Table B.6

Coverage Criteria Percentage for the Minimized Paths for the UML Statechart Diagram of an Online Shop

TP No	All-State	All-Transition	All-Transition-pairs	All-One-loop-paths
1,2,3,4	100%	100%	100%	-

The ten generated fireflies for each state are showing in Table B.7.

Table B.7

Calculation of Brightness Values of 10 Fireflies

V	1		2		3		4		5		6		7		8		9		10	
	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i
0	6.9	3.5	6.8	3.55	6.7	3.6	6.6	3.65	6.5	3.7	6.4	3.76	6.3	3.82	6.2	3.88	6.1	3.94	6	4
1	5.9	2.07	5.8	2.11	5.7	2.15	5.6	2.18	5.5	2.2	5.4	2.26	5.3	2.3	5.2	2.35	5.1	2.39	5	2.44
2	4.9	2.83	4.8	2.89	4.7	2.95	4.6	3.01	4.5	3.08	4.4	3.14	4.3	3.22	4.2	3.29	4.1	3.37	4	3.45
3	3.9	4.1	3.8	4.2	3.7	4.31	3.6	4.42	3.5	4.55	3.4	4.67	3.3	4.81	3.2	4.95	3.1	5.1	3	5.26
4	2.9	14.71	2.8	15.15	2.7	15.63	2.6	16.13	2.5	16.67	2.4	17.24	2.3	17.86	2.2	18.52	2.1	19.23	2	20

Table B.8 shows the separate calculation for cyclomatic complexity and information flow for each vertex, then show the Firefly brightness for that specific vertex after including the random factor.

Table B.8

Objective Function

Vertex	Cyclomatic Complexity CC	Information Flow IF_i	Firefly brightness A_i
0	4	0	3.94
1	4	4	2.26
2	3	4	3.29
3	2	4	4.31
4	1	1	14.71

By calculating the mean of brightness at every path using Equation 4.15, the results are shown in Table 9.

Table B.9

Test Path Prioritization for the Minimized Paths for the UML Statechart Diagram of an Online Shop

Test ID	Test path	Brightness value
TP 1	$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$	6.1010304355335
TP 4	$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 5$	4.6998174561816
TP 2	$0 \rightarrow 1 \rightarrow 5$	4.5997256564649
TP 3	$0 \rightarrow 1 \rightarrow 2 \rightarrow 5$	4.4963083323801

In Table B.9 test paths mean of the brightness value is calculated for each generated optimized test path. From the table it is observed that optimized test path one has the highest brightness value and hence having high priority. Then the fourth path, the second path, and finely the third one.

Section B: UML Statechart Diagram of an Airline Check-in

The process of minimize and prioritize the UML statechart diagram of an airline check-in example (see Section 5.2.2.1.3) is shown below.

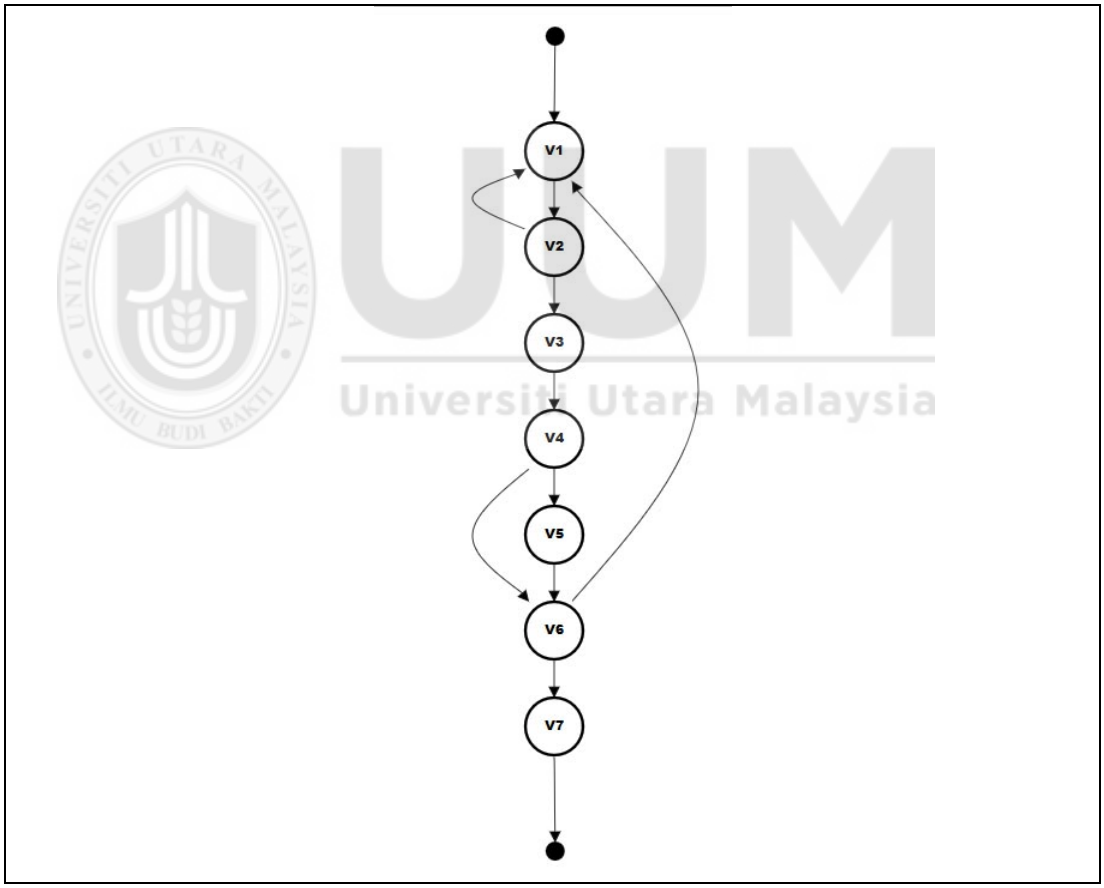

Figure B.4. Chart Relationship Graph of a UML Statechart Diagram of an Airline Check-in

The intermediate graph (Figure B.4) was converted to test paths using TCGP algorithm, and all the possible generated test paths from the intermediate graph is shown in Figure B.5.

TP 1: [S→1→2→3→4→5→6→7→E]
TP 2: [S→1→2→3→4→6→7→E]
TP 3: [S→1→2→3→4→5→6→1→2→3→4→5→6→7→E]
TP 4: [S→1→2→3→4→5→6→1→2→3→4→6→7→E]
TP 5: [S→1→2→3→4→6→1→2→3→4→5→6→7→E]
TP 6: [S→1→2→3→4→6→1→2→3→4→6→7→E]
TP 7: [S→1→2→1→2→3→4→5→6→7→E]
TP 8: [S→1→2→1→2→3→4→6→7→E]

Figure B.5. All Possible Test Paths of a UML Statechart Diagram of an Airline Check-in

Path weight was calculated for each tests path using Equation 4.7, as shown in Table B.10, to determine each path weight of transactions in the system

Table B.10

Path Weight for Each Path of a UML Statechart Diagram of an Airline Check-in

TC	S→1	1→2	2→3	2→1	3→4	4→5	4→6	5→6	6→7	6→1	7→E	E	W_v
1	1	1	1	0	1	1	0	1	1	0	1	8	0.88
2	1	1	1	0	1	0	1	0	1	0	1	7	0.87
3	1	1	1	0	1	1	0	1	1	1	1	9	0.6
4	1	1	1	0	1	1	1	1	1	1	1	10	0.71
5	1	1	1	0	1	1	1	1	1	1	1	10	0.71
6	1	1	1	0	1	0	1	0	1	1	1	8	0.61
7	1	1	1	1	1	1	0	1	1	0	1	9	0.81
8	1	1	1	1	1	0	0	0	1	0	1	7	0.7

After generate the path weight, next step start by calculate path coverage for each single path as shown in Table B.11.

Table B.11

Coverage Criteria for Each Path of a UML Statechart Diagram of an Airline Check-in

TP No	All-State	All-Transition	All-Transition-pairs	All-One-loop-paths
1	100%	72%	25%	0%
2	88%	63%	25%	0%
3	100%	81%	50%	50%
4	100%	90%	75%	50%
5	100%	90%	50%	50%
6	88%	72%	50%	50%
7	100%	81%	50%	50%
8	88%	63%	25%	50%

After generate the path weight and coverage criteria for each path the intermediate graph is converted to adjacency matrix, as showing in Table B.12. Then, this matrix is used to generate the guidance matrix for the graph.

Table B.12

Adjacency Matrix of a UML Statechart Diagram of an Airline Check-in

States	0	1	2	3	4	5	6	7	8
0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
2	0	1	0	1	0	0	0	0	0
3	0	0	0	0	1	0	0	0	0
4	0	0	0	0	0	1	1	0	0
5	0	0	0	0	0	0	1	0	0
6	0	1	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	0

After creating adjacency matrix, it is then used to generate guidance matrix. In the example in Figure B.4, the number of vertices is 9, and the number of edges is 11, therefore the Cyclomatic Complexity equal to 4. However, the Cyclomatic Complexity for each vertex need to be calculated using Equation 4.9 to be used to calculate the guidance value using Equation 4.8. The results are shown in Table B.13.

Table B.13

Guidance Value of a UML Statechart Diagram of an Airline Check-in

States	Cyclomatic Complexity CC	guidance value GF
0	4	316
1	4	276
2	4	236
3	3	147
4	3	117
5	2	58
6	2	38
7	1	9
8	1,000 [END vertex infinity]	1,000 [finial state]

Guidance matrix (Table B.14) is just as a look-up/decision table of adjacency matrix with each guidance value corresponding to every edge.

Table B.14

Guidance Matrix of a UML Statechart Diagram of an Airline Check-in

States	0	1	2	3	4	5	6	7	8
0	0	276	0	0	0	0	0	0	0
1	0	0	236	0	0	0	0	0	0
2	0	276	0	147	0	0	0	0	0
3	0	0	0	0	117	0	0	0	0
4	0	0	0	0	0	58	38	0	0
5	0	0	0	0	0	0	38	0	0
6	0	276	0	0	0	0	0	9	0
7	0	0	0	0	0	0	0	0	1000
8	0	0	0	0	0	0	0	0	0

Then the algorithm will generate the path sequences as:

Path 1= [0, 1, 2, 3, 4, 6, 7, 8],
Path 2= [2, 1],
Path 3= [4, 5, 6, 1].

To optimize the test cases, the algorithm will match each optimal path with paths in Figure B.5, and chose the lowest path weight W_v between the selected paths match paths. The minimized test paths are shown in Figure B.6

TP 2: [S→1→2→3→4→6→7→E]
TP 4: [S→1→2→3→4→5→6→1→2→3→4→6→7→E]
TP 8: [S→1→2→1→2→3→4→6→7→E]

Figure B.6. Minimized Test Paths of a UML Statechart Diagram of an Airline Check-in

The combination use of these three paths lead to achieving: all-state coverage, all-transition coverage, all-transition-pairs coverage, and all-one-loop coverage as shown in Table B15.

Table B.15

Coverage Criteria Percentage for the Minimized Paths

TP No	**All-State**	**All-Transition**	**All-Transition-pairs**	**All-One-loop-paths**
2, 4, 8	100%	100%	100%	100%

The ten generated fireflies for each state are showing in Table B16.

Table B.16

Calculation of Brightness Values of the Ten Fireflies

V	**1**		**2**		**3**		**4**		**5**		**6**		**7**		**8**		**9**		**10**	
	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i
0	8.9	2.73	8.8	2.76	8.7	2.79	8.6	2.82	8.5	2.86	8.4	2.89	8.3	2.92	8.2	2.96	8.1	2.99	8	3.03
1	7.9	0.96	7.8	0.98	7.7	0.99	7.6	1	7.5	1.02	7.4	1.03	7.3	1.04	7.2	1.06	7.1	1.07	7	1.09
2	6.9	1.78	6.8	1.81	6.7	1.83	6.6	1.86	6.5	1.89	6.4	1.92	6.3	1.95	6.2	1.98	6.1	2.01	6	2.04
3	5.9	4.07	5.8	4.13	5.7	4.2	5.6	4.27	5.5	4.35	5.4	4.42	5.3	4.5	5.2	4.59	5.1	4.67	5	4.76
4	4.9	2.83	4.8	2.89	4.7	2.95	4.6	3.01	4.5	3.08	4.4	3.14	4.3	3.22	4.2	3.29	4.1	3.37	4	3.45
5	3.9	7.87	3.8	8.06	3.7	8.26	3.6	8.47	3.5	8.7	3.4	8.93	3.3	9.17	3.2	9.43	3.1	9.71	3	10
6	2.9	5.43	2.8	5.62	2.7	5.81	2.6	6.02	2.5	6.25	2.4	6.49	2.3	6.76	2.2	7.04	2.1	7.35	2	7.69
7	1.9	20.83	1.8	21.74	1.7	22.73	1.6	23.81	1.5	25	1.4	26.32	1.3	27.78	1.2	29.41	1.1	31.25	1	33.33

Table B.17 shows the separate calculation for cyclomatic complexity and information flow for each vertex, then show the Firefly brightness for that specific vertex after including the random factor.

Table B.17

Objective Function

Vertex	Cyclomatic Complexity CC	Information Flow IF_i	Firefly brightness A_i
0	4	0	2.82
1	4	9	1.04
2	4	4	2.04
3	3	1	4.59
4	3	4	2.89
5	2	1	8.47
6	2	16	6.76
7	1	1	31.25

By calculating the mean of brightness at every path using Equation 4.15, the results are shown in Table B.18.

Table B.18

Test Path Prioritization of a UML Statechart Diagram of an Airline Check-in

Test ID	Test path	Brightness value
TP 2	0→1→2→3→4→6→7→0	8.3417877259468
TP 8	0→1→2→1→2→3→4→6→7→8	6.8306759195254
TP 4	0→1→2→3→4→5→6→1→2→3→4→5→7→8	6.7065188982599

In Table B.19 test paths mean of the brightness value is calculated for each generated optimized test path. From the table it is observed that optimized test second path has the highest brightness value and hence having high priority. Then the eighth path, and finely the fourth one.

Section C: UML Statechart Diagram for a Retail Point of Sale

The process of minimize and prioritize the UML statechart diagram for a retail point of sale example (see Section 5.2.2.1.4) is shown below.

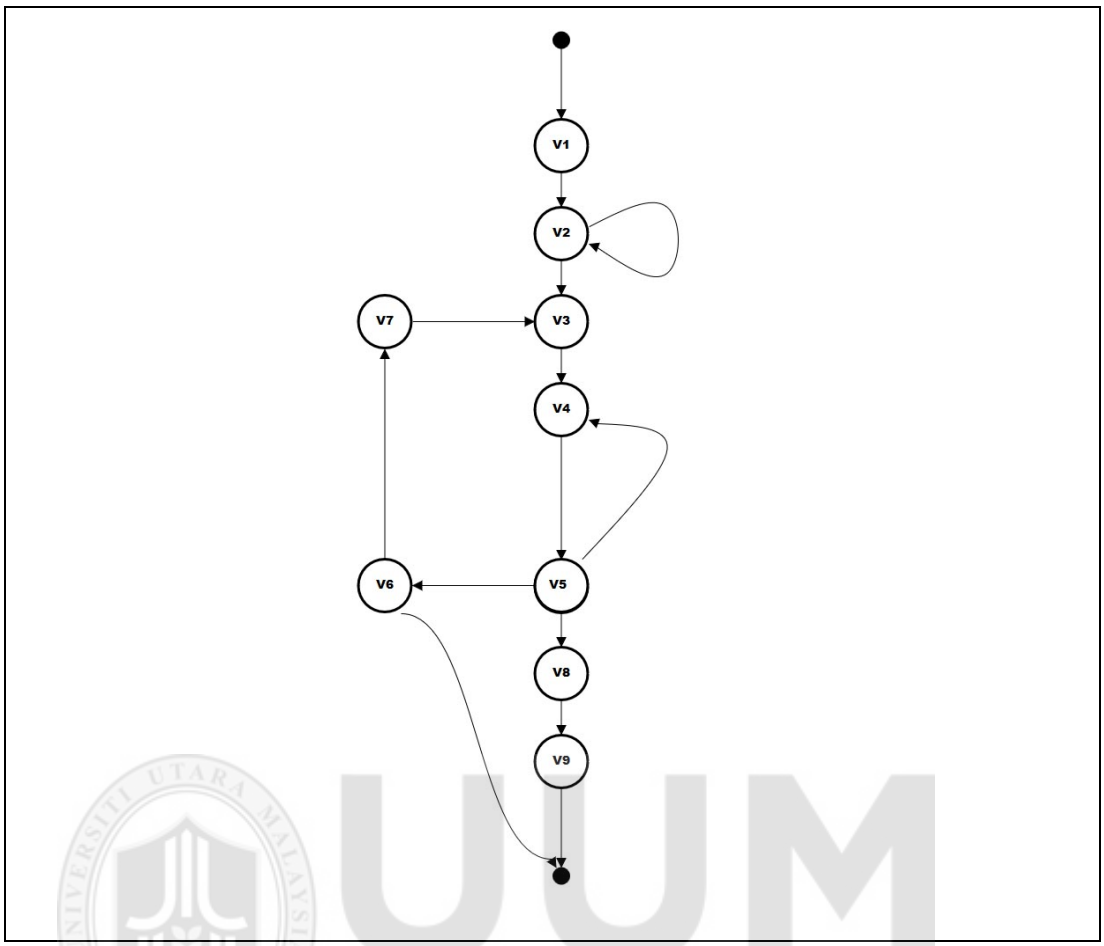

Figure B.7. Chart Relationship Graph for UML Statechart Diagram for a Retail Point of Sale

The intermediate graph (Figure B.7) was converted to test paths using TCGP algorithm, and all the possible generated test paths from the intermediate graph is shown in Figure B.8.

TP 1: [S→1→2→3→4→5→8→9→E]
TP 2: [S→1→2→3→4→5→6→E]
TP 3: [S→1→2→3→4→5→6→7→3→4→5→8→9→E]
TP 4: [S→1→2→3→4→5→6→7→3→4→5→6→E]
TP 5: [S→1→2→3→4→5→4→5→8→9→E]
TP 6: [S→1→2→3→4→5→4→5→6→E]
TP 7: [S→1→2→2→3→4→5→8→9→E]
TP 8: [S→1→2→2→3→4→5→6→E]

Figure B.8. All Possible Test Paths for UML Statechart Diagram for a Retail Point of Sale

Path weight was calculated for each tests path using Equation 4.7, as shown in Table B.19, to determine each path weight of transactions in the system

Table B.19

Path Weight for Each Path for UML Statechart Diagram for a Retail Point of Sale

TC	S → 1	1 → 2	2 → 3	2 → 2	3 → 4	4 → 5	5 → 6	5 → 4	5 → 8	6 → 7	6 → E	7 → 3	8 → 9	9 → E	E	W_v
1	1	1	1	0	1	1	0	0	1	0	0	0	1	1	8	0.88
2	1	1	1	0	1	1	1	0	0	0	1	0	0	0	7	0.87
3	1	1	1	0	1	1	1	1	1	1	0	1	1	1	12	0.85
4	1	1	1	0	1	1	1	1	0	1	1	1	0	0	10	0.76
5	1	1	1	0	1	1	0	1	1	0	0	0	1	1	9	0.81
6	1	1	1	0	1	1	1	1	0	0	1	0	0	0	8	0.8
7	1	1	1	1	1	1	0	0	1	0	0	0	1	1	8	0.9
8	1	1	1	1	1	1	1	0	0	0	1	0	0	0	8	0.88

After generate the path weight, next step start by calculate path coverage for each single path as shown in Table B.20.

Table B.20

Coverage Criteria for Each Path for UML Statechart Diagram for a Retail Point of Sale

TP No	All-State	All-Transition	All-Transition-pairs	All-One-loop-paths
1	81%	57%	28%	0%
2	72%	50%	42%	0%
3	100%	85%	42%	50%
4	81%	71%	57%	50%
5	81%	64%	42%	50%
6	72%	57%	57%	50%
7	81%	57%	42%	50%
8	72%	57%	57%	50%

After generate the path weight and coverage criteria for each path the intermediate graph is converted to adjacency matrix, as showing in table B.21. Then, this matrix is used to generate the guidance matrix for the graph.

Table B.21

Adjacency Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0	0	0
3	0	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	1	0	1	0	1	0	0
6	0	0	0	0	0	0	0	1	0	0	0
7	0	0	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0	0

After creating adjacency matrix, it is then used to generate guidance matrix. In the example in Figure B.7, the number of vertices is 11, and the number of edges is 14, therefore the Cyclomatic Complexity equal to 5. However, the Cyclomatic Complexity for each vertex need to be calculated using Equation 4.9 to be used to calculate the guidance value using Equation 4.8. The results are shown in Table B.22.

Table B.22

Guidance Value

States	Cyclomatic Complexity CC	guidance value GF
0	5	495
1	5	*445*
2	5	395
3	4	276
4	4	236
5	4	196
6	3	117
7	3	87
8	1	19
9	1	9
10	1,000 [END vertex infinity]	1,000 [finial state]

Guidance matrix (Table B.23) is just as a look-up/decision table of adjacency matrix with each guidance value corresponding to every edge.

Table B.23

Guidance matrix

States	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	445	0	0	0	0	0	0	0	0	0
1	0	0	395	0	0	0	0	0	0	0	0
2	0	0	395	276	0	0	0	0	0	0	0
3	0	0	0	0	236	0	0	0	0	0	0
4	0	0	0	0	0	195	0	0	0	0	0
5	0	0	0	0	236	0	117	0	19	0	0
6	0	0	0	0	0	0	0	87	0	0	1000
7	0	0	0	276	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	9	0
9	0	0	0	0	0	0	0	0	0	0	1000
10	0	0	0	0	0	0	0	0	0	0	0

Then the algorithm will generate the path sequences as:

Path 1= [0, 1, 2, 3, 4, 5, 8, 9, 10],
Path 2= [2, 2],
Path 3= [5, 6, 7, 3],
Path 4= [5, 4],
Path 5= [6, 10].

To optimize the test cases, the algorithm will match each optimal path with paths in Figure B.8, and chose the lowest path weight W_v between the selected paths match paths. The minimized test paths are shown in Figure B.9.

TP 1: [S→1→2→3→4→5→8→9→E]
TP 8: [S→1→2→2→3→4→5→6→E]
TP 4: [S→1→2→3→4→5→6→7→3→4→5→6→E]
TP 6: [S→1→2→3→4→5→4→5→6→E]

Figure B.9. Minimized Test Paths for UML Statechart Diagram for a Retail Point of Sale

The combination use of these three paths lead to achieving: all-state coverage, all-transition coverage, all-transition-pairs coverage, and all-one-loop coverage as shown in table B.24.

Table B.24

Coverage Criteria Percentage for the Minimized Paths

TP No	All-State	All-Transition	All-Transition-pairs	All-One-loop-paths
1, 8, 4, 6	100%	100%	100%	100%

The ten generated fireflies for each state are showing in Table B.25.

Table B.25

Calculation of Brightness Values of the Ten Fireflies

V	1		2		3		4		5		6		7		8		9		10	
	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i	d_i	A_i
0	11.9	1.65	11.8	1.67	11.7	1.68	11.6	1.69	11.5	1.71	11.4	1.72	11.3	1.74	11.2	1.75	11.1	1.77	1	1.79
1	10.9	1.51	10.8	1.52	10.7	1.53	10.6	1.55	10.5	1.56	10.4	1.58	10.3	1.59	10.2	1.61	10.1	1.62	1	1.64
2	9.9	0.48	9.8	0.48	9.7	0.49	9.6	0.49	9.5	0.50	9.4	0.50	9.3	0.51	9.2	0.51	9.1	0.52	9	0.52
3	8.9	1.39	8.8	1.41	8.7	1.42	8.6	1.43	8.5	1.45	8.4	1.46	8.3	1.47	8.2	1.49	8.1	1.50	8	1.52
4	7.9	1.56	7.8	1.58	7.7	1.59	7.6	1.61	7.5	1.62	7.4	1.64	7.3	1.65	7.2	1.67	7.1	1.68	7	1.70
5	6.9	1.18	6.8	1.19	6.7	1.21	6.6	1.22	6.5	1.24	6.4	1.25	6.3	1.27	6.2	1.28	6.1	1.30	6	1.31
6	5.9	2.36	5.8	2.38	5.7	2.40	5.6	2.42	5.5	2.44	5.4	2.46	5.3	2.48	5.2	2.50	5.1	2.52	5	2.54
7	4.9	4.85	4.8	4.89	4.7	4.93	4.6	4.97	4.5	5.01	4.4	5.05	4.3	5.09	4.2	5.13	4.1	5.17	4	5.21
8	3.9	11.36	3.8	11.63	3.7	11.91	3.6	12.20	3.5	12.50	3.4	12.82	3.3	13.16	3.2	13.51	3.1	13.89	3	14.29
9	2.9	14.71	2.8	15.15	2.7	15.63	2.6	16.13	2.5	16.67	2.4	17.24	2.3	17.86	2.2	18.52	2.1	19.23	2	20.00

Table B.26 shows the separate calculation for cyclomatic complexity and information flow for each vertex, then show the Firefly brightness for that specific vertex after including the random factor.

Table B.26

Objective Function

Vertex	Cyclomatic Complexity CC	Information Flow IF_i	Firefly brightness A_i
0	5	0	1.74
1	5	1	1.61
2	5	16	0.5
3	4	4	1.52
4	4	4	1.58
5	4	9	1.1
6	3	4	2.4
7	3	1	5.75
8	1	1	11.36
9	1	1	14.71

By calculating the mean of brightness at every path using Equation 4.15, the results are shown in Table B.27.

Table B.27

Test Path Prioritization for UML Statechart Diagram for a Retail Point of Sale

Test ID	Test path	Brightness value
TP 1	S→1→2→3→4→5→8→9→E	4.9599705586331
TP 4	S→1→2→3→4→5→6→7→3→4→5→6→E	2.6504048295212
TP 8	S→1→2→2→3→4→5→6→E	2.3697922355156
TP 6	S→1→2→3→4→5→4→5→6→E	2.3482365027468

In Table B.27 test paths mean of the brightness value is calculated for each generated optimized test path. From the table it is observed that optimized test path 1 has the highest brightness value and hence having high priority. Then the fourth path, the eighth path, and finely the sixth one.