Comment améliorer votre flux de travail à l'aide de tests basés sur des modèles

Moez Krichen

Comment améliorer votre flux de travail à l'aide de tests basés sur des modèles

Moez Krichen

Laboratoire ReDCAD, Université de Sfax, Tunisie
moez.krichen@redcad.org

Résumé. Les tests de logiciels sont une phase importante dans la construction d'un système logiciel évolutif qui a généralement des fonctions critiques, des flux commerciaux/logiques et des entités externes connectées. Cette nature distribuée des systèmes logiciels induit un certain niveau de complexité lors de l'écriture des tests pour chaque unité, fonction ou flux. Il existe différents types d'approches de test que vous pouvez utiliser. La meilleure approche que vous pouvez utiliser de manière transparente est le test basé sur un modèle. En termes simples, cela signifie créer un modèle de votre système et générer un test par rapport au modèle.

1 Introduction

Les tests de logiciels jouent un rôle essentiel dans le développement et le déploiement de systèmes logiciels (9; 18). Il s'agit d'un processus essentiel qui garantit que le logiciel répond aux normes de qualité souhaitées et fonctionne comme prévu . Les tests aident à identifier les défauts, les erreurs et les vulnérabilités du logiciel, qui peuvent être résolus avant que le logiciel ne soit mis à la disposition des utilisateurs finaux (31; 10). Des tests appropriés contribuent également à améliorer la fiabilité et les performances du logiciel, à réduire les coûts de maintenance et à accroître la satisfaction des clients (15; 41). Dans le monde numérique en évolution rapide d'aujourd'hui, où les systèmes logiciels deviennent de plus en plus complexes, les tests de logiciels sont devenus plus importants que jamais (17; 14). Dans ce contexte, il est crucial pour les organisations de comprendre et de mettre en œuvre des stratégies de test efficaces pour assurer le succès de leurs projets logiciels (47).

2 Qu'est-ce que les tests basés sur des modèles ?

Dans le développement de logiciels, il est crucial de savoir comment le système est censé fonctionner avant de le tester en profondeur. C'est là que les modèles sont utiles. Ils servent de représentation du comportement attendu du système et aident à définir ses processus en fonction des séquences d'entrée, des actions, des fonctions, de la sortie et du flux de données (7). Avec un modèle, les testeurs peuvent facilement déterminer tous ces comportements et générer automatiquement des tests basés sur les modèles du système (45).

Les tests basés sur des modèles sont une technique de test de logiciels relativement nouvelle qui gagne en popularité dans l'industrie (46). Il s'agit de générer des cas de test à partir d'un modèle qui décrit les aspects fonctionnels du système testé. Cette approche utilise une implémentation secondaire, légère et rapide d'une version logicielle appelée modèle (12). En prenant ce modèle avec les exigences du système, les testeurs peuvent générer des cas de test efficaces qui garantissent que le système fonctionne comme prévu (25).

Cette méthode de test ne se limite pas aux tests logiciels mais peut également être appliquée aux tests matériels (29). Il s'agit d'une approche polyvalente et puissante qui permet d'économiser du temps et des ressources tout en garantissant que le système répond aux normes de qualité souhaitées (39). L'adoption de tests basés sur des modèles peut aider les organisations à améliorer leur processus de développement logiciel et à fournir des produits logiciels de haute qualité à leurs clients (3).

3 Pourquoi et comment cela améliore-t-il le flux de travail ?

Dans le monde trépidant du développement de logiciels d'aujourd'hui, l'automatisation des tests est devenue une pratique essentielle. Il permet aux testeurs de logiciels d'effectuer des tests plus rapides et plus efficaces, rationalise leurs flux de travail et leur permet de tirer parti des dernières méthodologies de développement pour améliorer continuellement leurs processus de test (16).

Cependant, la création et la mise à jour des cas de test peuvent être un défi pour les développeurs de logiciels et les équipes, en particulier dans un environnement de dépendances et d'exigences en constante évolution. L'écriture et la mise à jour manuelle des scénarios de test pour chaque scénario peut prendre du temps et être source d'erreurs (6).

Pour relever ce défi, diverses méthodes de test ont été développées pour améliorer la fiabilité et l'efficacité des tests. Des tests fonctionnels simples aux méthodes lourdes telles que les tests de bout en bout (E2E), il existe un large éventail d'approches de test disponibles. Cependant, bien que ces méthodes soient efficaces, elles nécessitent toujours la création et la mise à jour manuelles de cas de test pour chaque scénario, ce qui peut être une tâche longue et fastidieuse (8).

Pour surmonter ce défi, des outils de test automatisés ont été développés qui permettent aux testeurs de logiciels de créer et de mettre à jour automatiquement des cas de test. Ces outils peuvent analyser les exigences du système et générer des cas de test basés sur des modèles et des règles prédéfinis. Cette approche permet non seulement de gagner du temps, mais également de réduire le risque d'erreurs et d'incohérences dans les cas de test (49).

En résumé, si les méthodes de test manuelles sont toujours valables, l'automatisation des tests est devenue une pratique incontournable dans le développement logiciel. En tirant parti des outils de test automatisés, les développeurs de logiciels et les équipes peuvent rationaliser leurs flux de travail, améliorer leurs processus de test et garantir la qualité de leurs produits logiciels (48).

Ces modèles sont utilisés pour générer des cas de test automatisés à l'aide d'outils MBT car ils décrivent le comportement attendu du système testé. Nous avons essentiellement deux étapes ici (2) :

1. Créer des modèles pour décrire le comportement et les processus du système.
2. Utilisation d'outils MBT tels que Spec Explorer, Graphwalker, fMBT ou Modbat, pour interpréter les modèles et générer des scripts de test pour les tests automatisés.

Lorsque vous travaillez avec des tests basés sur des modèles, la phase de création du modèle doit faire partie du cycle de vie du développement logiciel et être intégrée dans le cadre de la conception du produit à partir de la phase de spécification des exigences.

Cela permet à l'équipe de développement de logiciels de se concentrer sur la création d'un produit testable et de modèles qui améliorent l'expérience utilisateur. Les tests basés sur des modèles peuvent améliorer notre flux de travail en (26) :

1. Réduire le temps passé à écrire des tests et permettre aux développeurs de se concentrer sur l'écriture de modèles pour couvrir uniquement les exigences du système et créer une application testable dès le départ.
2. Réduction de la maintenance des suites de tests et génération de plus de tests.
3. Aider l'équipe à créer des scripts automatisés et à augmenter la couverture des tests lorsqu'ils sont utilisés avec des outils de test et des cadres d'automatisation.

4 Comment mettre en œuvre des tests basés sur des modèles

La mise en œuvre de tests basés sur des modèles n'est pas quelque chose qui peut être fait soudainement ou du jour au lendemain. Elle nécessite une approche progressive et systématique pour assurer son adoption réussie. Il convient mieux à la phase initiale d'un produit lorsque les choses sont encore relativement petites et simples. Cela facilite l'intégration des modèles aux exigences du système et garantit qu'ils couvrent tous les aspects nécessaires du comportement du système (44).

Pour implémenter des tests basés sur des modèles, vous devez commencer par créer les modèles eux-mêmes. Ces modèles peuvent couvrir n'importe quel niveau d'exigences, de la logique métier aux user stories, et peuvent être connectés les uns aux autres. Cela permet aux testeurs de naviguer facilement et de comprendre le comportement du système, et d'identifier tout défaut ou problème potentiel pouvant survenir (33).

Une fois les modèles créés, les testeurs peuvent générer automatiquement des cas de test basés sur eux. Cette approche permet de gagner du temps et réduit le risque d'erreurs et d'incohérences dans le processus de test. Il garantit également que les tests couvrent tous les aspects nécessaires du comportement du système, tels que définis par les modèles (37).

Si des modifications sont apportées aux modèles, les tests seront automatiquement mis à jour, garantissant qu'ils restent précis et pertinents. Cela facilite le maintien du processus de test dans le temps et garantit que le système est toujours testé de manière approfondie (35).

Enfin, une fois les tests automatisés générés, ils peuvent être facilement intégrés dans les processus et outils d'intégration continue (CI) de l'organisation. Cela permet aux testeurs d'exécuter les tests automatiquement et de s'assurer que tout défaut ou problème est détecté et résolu dès que possible. En fin de compte, la mise en œuvre de tests basés sur des modèles peut aider les organisations à améliorer la qualité et la fiabilité de leurs produits logiciels, à réduire les coûts et à accroître la satisfaction des clients (32).

5 Avantages des tests basés sur des modèles

Les tests basés sur des modèles offrent plusieurs avantages qui peuvent aider les organisations à optimiser le temps et les coûts de test de leurs logiciels. L'un des principaux avantages est la possibilité de générer des scripts de test pour chaque scénario en appuyant simplement sur un bouton. Cela facilite la création de scénarios de test complets qui couvrent tous les aspects nécessaires du comportement du système, garantissant que l'équipe de test logiciel peut communiquer efficacement le comportement attendu du système (50).

Un autre avantage des tests basés sur des modèles est la détection précoce des irrégularités des exigences. En créant des modèles qui représentent avec précision le comportement du système, les testeurs peuvent identifier tout défaut ou problème potentiel dans les exigences du système dès le début du processus de développement. Cela peut aider à réduire le temps et les coûts nécessaires pour résoudre ces problèmes ultérieurement (43).

La génération et l'exécution automatisées des cas de test rendent également la solution de test globale plus efficace et moins sujette aux erreurs. En générant automatiquement des cas de test basés sur les modèles, les testeurs peuvent gagner du temps et s'assurer que les tests couvrent tous les aspects nécessaires du comportement du système, réduisant ainsi le risque d'erreurs et d'incohérences dans le processus de test.

Les tests basés sur des modèles permettent de générer un nombre minimal de cas de test pour valider un flux fonctionnel ou de données donné afin de garantir que le système testé fonctionne parfaitement et ne fait jamais rien d'indésirable. Cette approche peut aider les organisations à économiser du temps et des ressources tout en garantissant la qualité et la fiabilité de leurs produits logiciels (40).

Enfin, les tests basés sur des modèles nécessitent une maintenance minimale du projet. Une fois que les modèles sont créés et que les tests automatisés sont générés, le processus de test peut être exécuté automatiquement sans nécessiter d'intervention ou de surveillance importante de la part des testeurs. Cela peut aider les organisations à rationaliser leurs processus de test et à réduire la charge de travail de leur équipe de test de logiciels (42).

Dans l'ensemble, les tests basés sur des modèles constituent une approche puissante et efficace des tests de logiciels qui peut aider les organisations à optimiser leurs processus de test, à réduire les coûts et à fournir des produits logiciels de haute qualité à leurs clients (11).

6 Inconvénients des tests basés sur des modèles

Bien que les tests basés sur des modèles offrent plusieurs avantages, il existe également des inconvénients potentiels dont les organisations doivent être conscientes (36). L'un des principaux inconvénients est qu'il nécessite un investissement important, à la fois en termes de temps et d'efforts. La création de modèles précis et la génération de tests automatisés peuvent être un processus complexe et chronophage, nécessitant une contribution importante de la part de testeurs de logiciels qualifiés et disciplinés (13).

Un autre inconvénient potentiel des tests basés sur des modèles est que le premier cas de test prend plus de temps à générer que les tests manuels traditionnels. En effet, il faut un travail plus poussé pour créer les modèles et s'assurer qu'ils représentent fidèlement le comportement du système. Cependant, une fois les modèles créés, la génération de cas de test ultérieurs devient beaucoup plus rapide et efficace (38).

La courbe d'apprentissage des tests basés sur des modèles peut également être abrupte, et sa complexité peut la rendre plus difficile à comprendre pour les débutants. Cela peut constituer un obstacle à l'entrée pour certaines organisations, en particulier celles qui disposent d'équipes de test moins expérimentées (34).

Enfin, les tests basés sur des modèles peuvent ne pas convenir à tous les types de projets logiciels. Dans certains cas, les tests manuels peuvent être plus appropriés ou efficaces, en particulier pour les projets plus petits ou moins complexes.

Malgré ces inconvénients potentiels, les tests basés sur des modèles restent une approche puissante et efficace des tests de logiciels qui peut aider les organisations à améliorer leurs processus de test et à fournir des produits logiciels de haute qualité à leurs clients. En comprenant ces inconvénients potentiels et en sélectionnant l'approche de test appropriée à leurs besoins spécifiques, les organisations peuvent s'assurer qu'elles tirent le meilleur parti de leurs ressources de test et fournissent les meilleurs produits logiciels possibles.

7 En quoi c'est différent des tests d'interface utilisateur

Les tests d'interface utilisateur sont un type de test de logiciel qui se concentre sur le test de la fonction de l'interface utilisateur. Cela implique des tests manuels, chaque scénario de test étant écrit à la main. Cela peut prendre du temps et toute modification apportée à l'interface utilisateur peut interrompre l'ensemble du scénario de test, à moins qu'il ne soit mis à jour avec les modifications. Pour simuler la façon dont les utilisateurs interagissent avec l'interface et valider le résultat attendu, des pilotes Web et des outils comme Selenium sont couramment utilisés.

L'un des avantages des tests d'interface utilisateur est qu'ils peuvent être utilisés à n'importe quelle étape du produit, car ils ne nécessitent pas un haut niveau d'expertise technique ou des modèles complexes. Cela en fait une méthode de test accessible et facile à apprendre qui peut être utilisée par les développeurs ayant des connaissances de base en test.

Le coût des tests est relativement faible pour les tests d'interface utilisateur, et le temps passé peut également être minime. Cependant, les exigences de maintenance peuvent être élevées, en particulier pour les interfaces de produits complexes.

Dans l'ensemble, bien que les tests basés sur des modèles et les tests d'interface utilisateur aient des cas d'utilisation et des avantages différents, les deux sont des éléments essentiels du processus de test de logiciels. En comprenant les forces et les faiblesses de chaque méthode de test, les organisations peuvent sélectionner l'approche appropriée à leurs besoins spécifiques et s'assurer qu'elles fournissent des produits logiciels de haute qualité à leurs clients.

8 Conclusion

Les tests basés sur des modèles sont une technique puissante, rentable et rentable pour les grandes entreprises à long terme. Cependant, l'introduction de cette approche dans les processus des grandes entreprises peut être un défi de taille, en particulier lorsqu'il s'agit de revoir l'ensemble de leur approche en matière de développement et de test de logiciels. Les tests basés sur des modèles doivent devenir une partie du flux de travail de développement, mais cela s'accompagne de ses propres défis, notamment des modifications de l'ensemble de l'infrastructure. Cela rend également une courbe d'apprentissage déjà abrupte encore plus difficile. Heureusement, certaines choses peuvent aider à identifier quand les tests basés sur des modèles peuvent vraiment être utiles. Par exemple, si vous avez un ensemble infini de systèmes avec des exigences que vous pouvez couvrir de différentes manières. Ou si vous avez un système distribué ou réactif, cela peut également être une raison d'envisager cette approche. Les tests basés sur des modèles peuvent faire beaucoup pour les tests et économiser beaucoup de temps et d'efforts lorsqu'ils sont correctement mis en œuvre. Comme perspectives possibles, Il serait intéressant d'étudier comment les techniques d'apprentissage automatique (ML) (1; 30; 4) peuvent être utilisées dans le contexte du test logiciel pour augmenter les niveaux de sécurité et pour augmenter les performances. Il sera également important d'appliquer des approches de test formelles (20; 27; 5; 28; 21; 19) pour le test logiciel afin d'améliorer leur qualité et d'augmenter leur robustesse (22; 24; 23).

Références

[1] Qasem Abu Al-Haija, Moez Krichen, and Wejdan Abu Elhaija. Machine-learning-based darknet traffic detection system for iot applications. *Electronics*, 11(4) :556, 2022.

[2] Abbas Ahmad, Fabrice Bouquet, Elizabeta Fourneret, Franck Le Gall, and Bruno Legeard. Model-based testing as a service for iot platforms. In *Leveraging Applications of Formal Methods, Verification and Validation : Discussion, Dissemination, Applications : 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part II 7*, pages 727–742. Springer, 2016.

[3] Bernhard K Aichernig, Wojciech Mostowski, Mohammad Reza Mousavi, Martin Tappler, and Masoumeh Taromirad. Model learning and model-based testing.

In *Machine Learning for Dynamic Software Analysis : Potentials and Limits : International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers*, pages 74–100. Springer, 2018.

[4] Ouissem Ben Fredj, Alaeddine Mihoub, Moez Krichen, Omar Cheikhrouhou, and Abdelouahid Derhab. Cybersecurity attack prediction : a deep learning approach. In *13th International Conference on Security of Information and Networks*, pages 1–6, 2020.

[5] Nathalie Bertrand, Amélie Stainer, Thierry Jéron, and Moez Krichen. A game approach to determinize timed automata. *Formal Methods in System Design*, 46(1) :42–80, 2015.

[6] Siddhartha R Dalal, Ashish Jain, Nachimuthu Karunanithi, JM Leaton, Christopher M Lott, Gardner C Patton, and Bruce M Horowitz. Model-based testing in practice. In *Proceedings of the 21st international conference on Software engineering*, pages 285–294, 1999.

[7] Arilo C Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H Travassos. A survey on model-based testing approaches : a systematic review. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies : held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, pages 31–36, 2007.

[8] Arilo Claudio Dias-Neto and Guilherme Horta Travassos. Model-based testing approaches selection for software projects. *Information and Software Technology*, 51(11) :1487–1504, 2009.

[9] Nasir U Eisty and Jeffrey C Carver. Testing research software : a survey. *Empirical Software Engineering*, 27(6) :138, 2022.

[10] Gordon Fraser and José Miguel Rojas. Software testing. *Handbook of Software Engineering*, pages 123–192, 2019.

[11] Ceren Sahin Gebizli and Hasan Sözer. Improving models for model-based testing based on exploratory testing. In *2014 IEEE 38th International Computer Software and Applications Conference Workshops*, pages 656–661. IEEE, 2014.

[12] Havva Gulay Gurbuz and Bedir Tekinerdogan. Model-based testing for software safety : a systematic mapping study. *Software Quality Journal*, 26 :1327–1372, 2018.

[13] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. Reducing the cost of model-based testing through test case diversity. In *Testing Software and Systems : 22nd IFIP WG 6.1 International Conference, ICTSS 2010, Natal, Brazil, November 8-10, 2010. Proceedings 22*, pages 63–78. Springer, 2010.

[14] Timo Hynninen, Jussi Kasurinen, Antti Knutas, and Ossi Taipale. Software testing : Survey of the industry practices. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1449–1454. IEEE, 2018.

[15] Muhammad Abid Jamil, Muhammad Arif, Normi Sham Awang Abubakar, and

Akhlaq Ahmad. Software testing techniques : A literature review. In *2016 6th international conference on information and communication technology for the Muslim world (ICT4M)*, pages 177–182. IEEE, 2016.

[16] Paul C Jorgensen. *The craft of model-based testing*. CRC Press, 2017.

[17] Mohamad Kassab, Joanna F DeFranco, and Phillip A Laplante. Software testing : The state of the practice. *IEEE Software*, 34(5) :46–52, 2017.

[18] Manju Khari and Manoj Kumar. Search-based secure software testing : A survey. In *Software Engineering : Proceedings of CSI 2015*, pages 375–381. Springer, 2018.

[19] Moez Krichen. *Model-based testing for real-time systems*. PhD thesis, PhD thesis, PhD thesis, Universit Joseph Fourier (December 2007), 2007.

[20] Moez Krichen. *Contributions to model-based testing of dynamic and distributed real-time systems*. PhD thesis, École Nationale d'Ingénieurs de Sfax (Tunisie), 2018.

[21] Moez Krichen and Stavros Tripakis. State identification problems for timed automata. In *IFIP International Conference on Testing of Communicating Systems*, pages 175–191. Springer, Berlin, Heidelberg, 2005.

[22] Mariam Lahami and Moez Krichen. A survey on runtime testing of dynamically adaptable and distributed systems. *Software Quality Journal*, 29(2) :555–593, 2021.

[23] Mariam Lahami, Moez Krichen, Hajer Barhoumi, and Mohamed Jmaiel. Selective test generation approach for testing dynamic behavioral adaptations. In *IFIP International Conference on Testing Software and Systems*, pages 224–239. Springer, Cham, 2015.

[24] Mariam Lahami, Moez Krichen, and Mohamed Jmaïel. Runtime testing approach of structural adaptations for dynamic and distributed systems. *International Journal of Computer Applications in Technology*, 51(4) :259–272, 2015.

[25] Wenbin Li, Franck Le Gall, and Naum Spaseski. A survey on model-based testing tools for test case generation. In *Tools and Methods of Program Analysis : 4th International Conference, TMPA 2017, Moscow, Russia, March 3-4, 2017, Revised Selected Papers 4*, pages 77–89. Springer, 2018.

[26] Malte Lochau, Ina Schaefer, Jochen Kamischke, and Sascha Lity. Incremental model-based testing of delta-oriented software product lines. In *Tests and Proofs : 6th International Conference, TAP 2012, Prague, Czech Republic, May 31–June 1, 2012. Proceedings 6*, pages 67–82. Springer, 2012.

[27] Afef Jmal Maâlej, Moez Krichen, and Mohamed Jmaiel. Conformance testing of ws-bpel compositions under various load conditions. In *2012 IEEE 36th annual computer software and applications conference*, pages 371–371. IEEE, 2012.

[28] Afef Jmal Maâlej, Moez Krichen, and Mohamed Jmaiel. Model-based conformance testing of ws-bpel compositions. In *2012 IEEE 36th annual computer software and applications conference workshops*, pages 452–457. IEEE, 2012.

[29] Raluca Marinescu, Cristina Seceleanu, Hèléne Le Guen, and Paul Pettersson. A research overview of tool-supported model-based testing of requirements-based

designs. *Advances in Computers*, 98 :89–140, 2015.

[30] Alaeddine Mihoub, Ouissem Ben Fredj, Omar Cheikhrouhou, Abdelouahid Derhab, and Moez Krichen. Denial of service attack detection and mitigation for internet of things using looking-back-enabled machine learning techniques. *Computers & Electrical Engineering*, 98 :107716, 2022.

[31] Deepti Bala Mishra, Rajashree Mishra, Kedar Nath Das, and Arup Abhinna Acharya. A systematic review of software testing using evolutionary techniques. In *Proceedings of Sixth International Conference on Soft Computing for Problem Solving : SocProS 2016, Volume 1*, pages 174–184. Springer, 2017.

[32] Mohamed Mussa, Samir Ouchani, Waseem Al Sammane, and Abdelwahab Hamou-Lhadj. A survey of model-driven testing techniques. In *2009 Ninth International Conference on Quality Software*, pages 167–172. IEEE, 2009.

[33] Arilo Dias Neto, Rajesh Subramanyan, Marlon Vieira, Guilherme Horta Travassos, and Forrest Shull. Improving evidence about software technologies : A look at model-based testing. *IEEE software*, 25(3) :10–13, 2008.

[34] Joao Felipe S Ouriques, Emanuela G Cartaxo, and Patrícia DL Machado. Test case prioritization techniques for model-based testing : a replicated study. *Software Quality Journal*, 26 :1451–1482, 2018.

[35] Alexander Pretschner. Model-based testing. In *Proceedings of the 27th international conference on Software engineering*, pages 722–723, 2005.

[36] Alexander Pretschner and Jan Philipps. 10 methodological issues in model-based testing. *Model-Based Testing of Reactive Systems : Advanced Lectures*, pages 281–291, 2005.

[37] Hassan Reza and Suhas Lande. Model based testing using software architecture. In *2010 Seventh International Conference on Information Technology : New Generations*, pages 188–193. IEEE, 2010.

[38] Bernhard Rumpe. Model-based testing of object-oriented systems. In *Formal Methods for Components and Objects : First International Symposium, FMCO 2002, Leiden, The Netherlands, November 5-8, 2002, Revised Lectures 1*, pages 380–402. Springer, 2003.

[39] Aneesa Saeed, Siti Hafizah Ab Hamid, and Mumtaz Begum Mustafa. The experimental applications of search-based techniques for model-based testing : Taxonomy and systematic literature review. *Applied Soft Computing*, 49 :1094–1117, 2016.

[40] Ina Schieferdecker and Andreas Hoffmann. Model-based testing. *IEEE software*, 29(1) :14–18, 2012.

[41] Karuturi Sneha and Gowda M Malle. Research on software testing techniques and software automation testing tools. In *2017 international conference on energy, communication, data analytics and soft computing (ICECDS)*, pages 77–81. IEEE, 2017.

[42] Mark Timmer, Ed Brinksma, and Mariëlle Stoelinga. Model-based testing. In *Software and systems safety*, pages 1–32. IOS Press, 2011.

[43] Mark Utting. Position paper : Model-based testing. *Verified Software : Theories, Tools, Experiments. ETH Zürich, IFIP WG*, 2, 2005.

[44] Mark Utting and Bruno Legeard. *Practical model-based testing : a tools approach.* Elsevier, 2010.

[45] Mark Utting, Bruno Legeard, Fabrice Bouquet, Elizabeta Fourneret, Fabien Peureux, and Alexandre Vernotte. Recent advances in model-based testing. *Advances in computers*, 101 :53–120, 2016.

[46] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software testing, verification and reliability*, 22(5) :297–312, 2012.

[47] Andrei-Mihai Vadan and Liviu-Cristian Miclea. Software testing techniques for improving the quality of smart-home iot systems. *Electronics*, 12(6) :1337, 2023.

[48] Leonardo Villalobos-Arias, Christan Quesada-López, Alexandra Martinez, and Marcelo Jenkins. Model-based testing areas, tools and challenges : A tertiary study. *CLEI Electronic Journal*, 22(1) :3–1, 2019.

[49] Sebastian Wieczorek and Alin Stefanescu. Improving testing of enterprise systems by model-based testing on graphical user interfaces. In *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, pages 352–357. IEEE, 2010.

[50] Qian Yang, J Jenny Li, and David Weiss. A survey of coverage based testing tools. In *Proceedings of the 2006 international workshop on Automation of software test*, pages 99–103, 2006.