Les défis du Test Logiciel - Bilan et Perspectives

Frédéric Dadeau, Hélène Waeselynck

Les défis du Test Logiciel - Bilan et Perspectives

Réponse à l'appel à défis GDR-GPL 2025 – Mars 2014

Frédéric Dadeau – FEMTO-ST Hélène Waeselynck – LAAS

Résumé

Ce document dresse un bilan des défis identifiés par le groupe de travail Méthodes de Test pour la Vérification et la Validation (MTV2) lors de l'appel lancé par le GDR GPL en 2010. Pour chaque défi initialement identifié, nous évaluons si des réponses ont été apportées durant ces 4 dernières années, nous présentons les éventuelles avancées réalisées, et proposons le cas échéant de nouveaux défis liés aux technologies émergentes.

1 Le défi des techniques de test

1.1 Test à partir de modèles et de code

Lors du précédent appel à défis, nous partions du constat que code et modèles, deux artefacts de la génération de test, étaient réalisés indépendamment. Un défi consistait à chercher une plus grande intégration entre *code-based testing* (CBT) et *model-based testing* (MBT). Différentes approches ont mis en avant l'utilisation de langages d'annotations, notamment au CEA, avec la définition du langage ACSL (ANSI-C Specification Language) et le couplage entre la plateforme Frama-C et l'outil de génération de test PathCrawler [9]. En déportant les éléments de modèle au sein du code, les langages d'annotation apportent également une réponse au besoin de faire évoluer conjointement code et modèle lors du cycle de vie du logiciel [1]. Néanmoins, l'expressivité des langages d'annotations restreint leur utilisation à la génération de tests unitaires, et n'adresse pas directement la problématique du test de recette, comme pourraient le faire d'autres formalismes. La conception conjointe de modèle pour le test et de code n'est donc pas encore d'actualité.

1.2 Passage à l'échelle de technologies

L'un des enjeux techniques majeurs est le passage à l'échelle des techniques et des technologies de génération de test. Le défi ici est d'être capable de traiter des modèles, ou des systèmes, de taille industrielle, qui présentent un très grand nombre de comportements, et un espace d'états quasiment infini. De nombreux progrès ont été faits ces dernières années, avec l'avènement des techniques symboliques, et notamment les solveurs SMT (Z3, CVC4, etc.) dont les performances s'améliorent sans cesse [8, 15]. Similairement, l'exploration utilisant des techniques issues du model-checking, notamment basées sur des aspects aléatoires ou probabilistes permettait d'améliorer le passage à l'échelle et de parcourir de vastes espaces d'états [11]. Pour finir, les techniques d'algorithmique distribuée dans des centres de calcul ou en cloud-computing ont permis de repousser la limite technologique des architectures matérielles sur lesquelles s'exécutaient les générateurs de test.

Pour autant, le passage à l'échelle reste encore d'actualité, de par la taille des systèmes à considérer qui ne pourra se combattre que par la définition de critères de sélection de tests pertinents qui limitent le nombre de cas de tests à générer, et les besoins d'explorer l'espace d'états des programmes ou des modèles [14].

2 Le défi des attentes sociétales : tester la sécurité logicielle

La sécurité logicielle est au coeur des préoccupations actuelles. Les dernières années ont vu grandir le phénomène des Software-as-a-Service (SaaS), accélérés par la démocratisation des "clouds". Par ailleurs, le déploiement des smartphones, et des applications mobiles, a entraîné l'apparition de nouvelles formes d'exploitation des vulnérabilités (par exemple, l'accès aux données personnelles de l'utilisateur) qui impactent directement les citoyens. Aussi, les technologies web représentent un domaine en constante évolution, dans lequel les problèmes de sécurité se règlent le plus souvent en aval, par le biais de mises à jour. Les techniques de test actifs, comme le test de pénétration permettent désormais d'assurer en amont de la sécurité du système en termes d'absences de vulnérabilités exploitables. De nombreux travaux ont également émergé autour des vulnérabilités logiques (notamment le test de protocoles de sécurité) [24, 16, 7].

Dans ce contexte, un défi est lié au développement des techniques de test actif pour la sécurité, telle que le test de pénétration, qui requiert la connaissance de la logique applicative pour permettre d'explorer exhaustivement les points de vulnérabilités potentiels. Par ailleurs, un aspect non négligeable de ces approches est lié à la testabilité des applications considérées lors de la recherche de failles logiques.

3 Le défi des environnements

3.1 Test de systèmes mobiles/ubiquitaires

Le défi précédent identifiait une montée en puissance des appareils mobiles, et les besoins associés en termes de modélisation d'infrastructures mobiles, d'exécutions de tests ciblant la topologie d'un réseau de terminaux, Un système mobile inclut des dispositifs qui se déplacent dans le monde physique tout en étant connectés aux réseaux par des moyens sans fil. Des travaux récents ont défini une approche de test passif pour de tels systèmes vérifiant des propriétés les traces d'exécution prenant en compte à la fois les configurations spatiales des nœuds du système et leurs communications [2, 3]. Pour compléter ces vérifications macroscopiques relatives aux interactions entre noeuds, on peut également s'intéresser à l ?observation fine de l ?exécution au niveau d ?un dispositif (ex : tablette ou téléphone mobile). Selon les propriétés à vérifier, un défi serait d'exploiter au mieux des instrumentations matérielles et logicielles pour enregistrer les données d ?exécution pertinentes. L ?idée serait d ?utiliser ces enregistrements non seulement lors des tests, mais également après déploiement pour effectuer un suivi des problèmes opérationnels.

3.2 Test d'architectures reconfigurables

Une approche de conception actuellement en vogue consiste à utiliser une bibliothèque de micro-mécanismes, qui sont composés pour construire des mécanismes de tolérance aux fautes et attachés au code applicatif. L'objectif est de permettre des manipulations à grain fin de l'architecture résultante, pour qu'un intégrateur ou un administrateur puisse facilement la reconfigurer à des fins d'adaptation à un nouveau contexte opérationnel. Plusieurs technologies logicielles sont actuellement étudiées pour composer le code applicatif et les mécanismes de tolérance aux fautes, comme la programmation orientée-aspect [18] et des technologies basées sur des composants et des services [12]. Dans tous les cas, le défi est de valider le comportement émergeant de la composition des mécanismes avec le code applicatif. La conception du test va alors dépendre de la technologie considérée, en particulier des opérateurs de composition offerts. Pour les technologies permettant des reconfigurations à l'exécution, des problèmes additionnels concernent la validation des transitions entre configurations. Cette problématique se rapproche de celle du test de lignes de produits, qui représente un domaine d'application émergeant, dans lequel le problème de la réutilisation de cas de test va également se heurter aux problèmes de variabilité, et de nouveaux comportements issus de combinaisons inattendues [20, 17].

3.3 Données et monde aléatoires

Dans le cadre de la génération aléatoire, un défi concerne la notion même de domaine d'entrée, dans le cas de systèmes autonomes évoluant dans un environnement incertain. Si l'on considère le test de services de base d'un système autonome –par exemple, le test de la navigation d'un robot– le domaine de génération est un espace de mondes dans lequel le système est susceptible d'évoluer ! En pratique, les services sont testés en simulation sur une poignée d'exemples de mondes, ce qui est tout à fait insuffisant. Pour assurer plus de diversité, on peut envisager de mettre en œuvre des techniques issues de la génération procédurale de mondes, utilisées notamment dans la création de scènes pour des jeux vidéo [4]. Se posent alors des problèmes amont de modélisation de l'espace des mondes, incluant des caractéristiques stressantes pour le service testé (en liaison avec des analyses de sécurité), et sur la définition de critères de couverture pour guider l'échantillonnage de l'espace.

3.4 Objets connectés et Internet des objets

Les objets connectés commencent à apparaître dans les foyers. Téléviseurs, imprimantes, réfrigérateurs sont désormais connectés en permanence à Internet, et répondent à des standards ou à des normes qu'ils doivent satisfaire. Ces objets représentent un challenge évident du point de vue de la sécurité et des questions de protection de la vie privée des citoyens, déjà abordé précédemment. Au delà de ces problèmes se pose la question de la validation des normes, et du respect des standards, par le biais de techniques de test de *compliance* (conformité au standard et compatibilité des interfaces). Cette problématique existe déjà pour certains systèmes, comme les cartes à puce, et sera amenée à s'accroître dans le futur (on estime à 80 milliards le nombre d'objets connectés d'ici 2020, contre 15 milliards aujourd'hui). En ce sens, le test à partir de modèle peut apparaître comme une solution pertinente pour s'assurer de la cohérence du standard (lors de la construction du modèle) et de la conformité des applications à la norme.

4 Le défi des pratiques du test

4.1 IDM et méthodes agiles pour le test

Lors de l'appel à défi précédent, le groupe MTV2 avait identifié un défi lié aux méthodes agiles et aux aspects IDM, alors émergeantes dans les pratiques de développement. Les méthodes agiles ont bousculé les pratiques des équipes de développement ces dernières années. Une large majorité des équipes se sont (ré)organisées autour de ces méthodes, telles que SCRUM par ailleurs outillées, qui remettent le développeur au sein des décisions et font une part importante aux phases de validation. Par ailleurs, la démocratisation d'environnements d'intégration continue, tels que Jenkins, qui permettent un couplage entre un gestionnaire de version et un environnement d'exécution de tests, offrent un support de qualité à ces pratiques. Ainsi le test prend une place de plus en plus centrale dans les processus de développement actuels.

Dans ce contexte, deux challenges se posent alors. Le premier concerne l'accroissement intrinsèque du nombre de tests, rendant problématique la ré-exécution systématique de tout le référentiel de tests. Ainsi des techniques de prioritisation des cas de tests doivent être mises en oeuvre, pour maintenir un bon niveau de service des outils. Le second challenge est lié aux évolutions constantes du logiciel en cours de développement, qui entraine à la fois la nécéssité de tests de non-régression, et l'invalidation de tests devenus obsolètes. Il est donc nécessaire trouver des solutions pour gérer l'évolution du code et du référentiel de test, en particulier dans le cadre des méthodes de développement agiles.

Une autre idée serait d'exploiter la connaissance de tests déjà existants pour suggérer de nouveaux tests, généraliser les tests existants ou les faire évoluer. Pour cela, on pourra s'inspirer de techniques issues d'un domaine de recherche très actif, le software repository mining en les adaptant à la fouille de tests [5].

4.2 Démocratiser le test à partir de modèles

Le test à partir de modèles (Model-Based Testing – MBT) représente le moyen principal pour automatiser la génération et l'exécution de tests fonctionnels. En outre, cette approche permet d'assurer la traçabilité entre les exigences informelles exprimées au niveau du modèle, et les tests produits, fournissant ainsi des métriques séduisantes d'un point de vue industriel (notamment dans le cadre de certifications de type Critères Communs) [10]. Néanmoins, l'adoption du MBT dans l'industrie se heurte à deux problèmes majeurs. Le premier concerne la conception de modèle, qui constitue un effort conséquent demandé à l'ingénieur validation. Par ailleurs, l'ingénieur validation se heurtera également au problème corrolaire de valider le modèle de test, pour s'assurer que celui-ci représente fidèlement le système modélisé. Le second problème concerne le passage à l'échelle des techniques de génération de test qui peinent à convaincre les industriels. Outre ces challenges techniques, le défi consiste à faire en sorte que, dans les cas les plus adaptés, des approches de type MBT soient favorisées et par des industriels. Cela passe par de l'accompagnement des équipes de validation vers ces pratiques, ainsi que la construction de formalismes et d'outils adaptés, qui réduisent le difficulté d'apprentissage (par exemple, en passant par des DSL) et apportent des solutions adaptées aux problèmes du passage à l'échelle [6].

4.3 Découverte de propriétés de services externes

On peut attribuer un autre rôle au test et à la surveillance en-ligne que la validation de systèmes, notamment, ces techniques peuvent être utilisées pour la découverte de propriétés. Cette approche est notamment pertinente dans le cadre de systèmes ouverts, caractérisés par la composition de services développés et maintenus en dehors des applications cibles. Plusieurs travaux ont déjà porté sur l'inférence de modèles comportementaux à partir de l'observation de traces d'exécution [19, 13, 21, 23]. Dans ce contexte, un défi serait de définir de nouvelles méthodes d'inférence active, où l'information apportée par des traces existantes serait complétée en sélectionnant des tests additionnels [22, 25]. Les critères de sélection de test pourraient alors être liés à la structure du modèle déjà inféré et à d'éventuelles hypothèses sur des généralisations/abstractions possibles des comportements observés.

4.4 Formation des étudiants et des professionnels au test logiciel

Le dernier défi initialement identifié concernait l'enseignement du test. En effet, l'activité de test devient de plus en plus importante suite à la délocalisation du développement des logiciels dans des pays à faibles coûts de main d'oeuvre. De fait, il revient aux équipes de désormais valider le code développé hors des frontières. Ainsi, le métier d'ingénieur validation devient de plus en plus central au sein des équipes de développement et il est nécessaire de former les étudiants et les professionnels au métier de testeur. L'arrivée dans le paysage universitaire des Cursus Master en Ingénierie (CMI) [1] apparaît comme une solution au besoin de formation d'ingénieurs validation. Les CMIs visent en effet à former des diplômés à Bac+5 (niveau Master) qui acquièrent une solide connaissance de l'état de l'art des pratiques et des théories issues du monde académique, au travers une interaction forte entre formation et recherche. Ils fournissent ainsi un moyen, dans le cadre d'une spécialité "test de logiciels" d'initier les futurs diplômés aux techniques et concepts les plus avancés dans ce domaine, issus des laboratoire de recherche à la pointe de ces thématiques. Par ailleurs, la généralisation des cours en ligne ouverts et massifs (MOOC) offre à tous un accès à des enseignements des différentes techniques du test logiciel. Pour finir, les certifications proposées par l'International Software Testing Qualification Board (ISTQB) [2] offrent une formation ad hoc et permettent de valider un certain niveau de connaissance des pratiques du test.

1. www.reseau-figure.fr
2. www.istqb.org

Références

[1] BernhardK. Aichernig. Contract-based testing. In BernhardK. Aichernig and Tom Maibaum, editors, *Formal Methods at the Crossroads. From Panacea to Foundational Support*, volume 2757 of *Lecture Notes in Computer Science*, pages 34–48. Springer Berlin Heidelberg, 2003.

[2] P. André, H. Waeselynck, and N. Rivière. A uml-based environment for test scenarios in mobile settings. In *Computer, Information and Telecommunication Systems (CITS), 2013 International Conference on*, pages 1–5, May 2013.

[3] Pierre André, Nicolas Rivière, and Hélène Waeselynck. Graphseq revisited : More efficient search for patterns in mobility traces. In Marco Vieira and Joao Carlos Cunha, editors, *Dependable Computing*, volume 7869 of *Lecture Notes in Computer Science*, pages 88–95. Springer Berlin Heidelberg, 2013.

[4] James Arnold and Rob Alexander. Testing autonomous robot control software using procedural content generation. In Friedemann Bitsch, Jérémie Guiochet, and Mohamed Kaaniche, editors, *Computer Safety, Reliability, and Security*, volume 8153 of *Lecture Notes in Computer Science*, pages 33–44. Springer Berlin Heidelberg, 2013.

[5] SuriyaPriyaR. Asaithambi and Stan Jarzabek. Towards test case reuse : A study of redundancies in android platform test libraries. In John Favaro and Maurizio Morisio, editors, *Safe and Secure Software Reuse*, volume 7925 of *Lecture Notes in Computer Science*, pages 49–64. Springer Berlin Heidelberg, 2013.

[6] Julien Botella, Fabrice Bouquet, Jean-François Capuron, Franck Lebeau, Bruno Legeard, and Florence Schadle. Model-based testing of cryptographic components – lessons learned from experience. In *ICST'13, 6th IEEE Int. Conf. on Software Testing, Verification and Validation*, pages 192–201, March 2013.

[7] Matthias Büchler, Karim Hossen, Petru Florin Mihancea, Marius Minea, Roland Groz, and Catherine Oriat. Model inference and security testing in the spacios project. In Serge Demeyer, Dave Binkley, and Filippo Ricca, editors, *CSMR-WCRE*, pages 411–414. IEEE, 2014.

[8] Cristian Cadar and Koushik Sen. Symbolic execution for software testing : three decades later. *Commun. ACM*, 56(2) :82–90, 2013.

[9] Omar Chebaro, Nikolaï Kosmatov, Alain Giorgetti, and Jacques Julliand. Combining static analysis and test generation for C program debugging. In G. Fraser and A. Gargantini, editors, *TAP'10, 4th Int. Conf. on Tests and Proofs*, volume 6143 of *LNCS*, pages 94–100, Malaga, Spain, July 2010.

[10] Frédéric Dadeau, Kalou Cabrera Castillos, Yves Ledru, Taha Triki, German Vega, Julien Botella, and Safouan Taha. Test generation and evaluation from high-level properties for common criteria evaluations - the TASCCC testing tool. In B. Baudry and A. Orso, editors, *ICST 2013, 6th Int. Conf. on Software Testing, Verification and Validation, Testing Tool track*, pages 431–438, Luxemburg, Luxemburg, March 2013. IEEE Computer Society Press.

[11] Alain Denise, Marie-Claude Gaudel, Sandrine-Dominique Gouraud, Richard Lassaigne, Johan Oudinet, and Sylvain Peyronnet. Coverage-biased random exploration of large models and application to testing. *STTT*, 14(1) :73–93, 2012.

[12] Quentin Enard, Miruna Stoicescu, Emilie Balland, Charles Consel, Laurence Duchien, Jean-Charles Fabre, and Matthieu Roy. Design-Driven Development Methodology for Resilient Computing. In *CBSE'13 : Proceedings of the 16th International ACM Sigsoft Symposium on Component-Based Software Engineering*, Vancouver, Canada, June 2013.

[13] C. Ghezzi, A. Mocci, and M. Sangiorgio. Runtime monitoring of component changes with spy@runtime. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 1403–1406, June 2012.

[14] Hadi Hemmati, Andrea Arcuri, and Lionel C. Briand. Achieving scalable model-based testing through test case diversity. *ACM Trans. Softw. Eng. Methodol.*, 22(1) :6, 2013.

[15] Thierry Jéron, Margus Veanes, and Burkhart Wolff. Symbolic Methods in Testing (Dagstuhl Seminar 13021). *Dagstuhl Reports*, 3(1) :1–29, 2013.

[16] Ralph LaBarge and Thomas McGuire. Cloud penetration testing. *CoRR*, abs/1301.1912, 2013.

[17] Beatriz Pérez Lamancha, Macario Polo, and Mario Piattini. Systematic review on software product line testing. In José Cordeiro, Maria Virvou, and Boris Shishkov, editors, *Software and Data Technologies*, volume 170 of *Communications in Computer and Information Science*, pages 58–71. Springer Berlin Heidelberg, 2013.

[18] Jimmy Lauret, Jean-Charles Fabre, and Hélène Waeselynck. Fine-grained implementation of fault tolerance mechanisms with aop : To what extent ? In Friedemann Bitsch, Jérémie Guiochet, and Mohamed Kaaniche, editors, *Computer Safety, Reliability, and Security*, volume 8153 of *Lecture Notes in Computer Science*, pages 45–56. Springer Berlin Heidelberg, 2013.

[19] Choonghwan Lee, Feng Chen, and Grigore Roşu. Mining parametric specifications. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 591–600, New York, NY, USA, 2011. ACM.

[20] Jihyun Lee, Sungwon Kang, and Danhyung Lee. A survey on software product line testing. In *Proceedings of the 16th International Software Product Line Conference - Volume 1*, SPLC'12, pages 31–40, New York, NY, USA, 2012. ACM.

[21] David Lo, Leonardo Mariani, and Mauro Santoro. Learning extended {FSA} from software : An empirical assessment. *Journal of Systems and Software*, 85(9) :2063 – 2076, 2012. Selected papers from the 2011 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA 2011).

[22] Harald Raffelt, Bernhard Steffen, Therese Berg, and Tiziana Margaria. Learnlib : a framework for extrapolating behavioral models. *STTT*, 11(5) :393–407, 2009.

[23] G. Reger, H. Barringer, and D. Rydeheard. A pattern-based approach to parametric specification mining. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 658–663, Nov 2013.

[24] M. I. P. Salas and E. Martins. Security testing methodology for vulnerabilities detection of xss in web services and ws-security. *Electr. Notes Theor. Comput. Sci.*, 302 :133–154, 2014.

[25] Muzammil Shahbaz and Roland Groz. Analysis and testing of black-box component based systems by inferring partial models. *Software Testing, Verification and Reliability*, feb 2013.