International Journal on Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

TSTL: The Template Scripting Testing Language

Josie Holmes · Alex Groce · Jervis Pinto · Pranjal Mittal · Pooria Azimi · James O'Brien

Received: date / Accepted: date

Abstract A test harness, in automated testing, defines the set of valid tests for a system, and usually also defines a set of correctness properties. The difficulty of writing test harnesses is a major obstacle to the adoption of automated test generation and model checking. Languages for writing test harnesses are usually tied to a particular tool, unfamilliar to programmers, and often limit expressiveness. Writing test harnesses directly in the language of the Software Under Test (SUT) is a tedious, repetitive, and error-prone task, offers little or no support for test case manipulation and debugging, and produces repetitive, hard-to-read code. Both approaches tend to tie a test harness to one method of test generation, with little ability to explore alternative methods. In this paper, we present TSTL, the Template Scripting Testing Language, a domain-specific language (DSL) for writing test harnesses that compiles harness definitions into an abstract graph-based interface that abstracts away the details of the SUT, making generic test generation and manipulation tools possible. TSTL includes a compiler to Python, and a suite of tools for generating, manipulating, and analyzing test cases, including various explicit-state model checkers. This paper showcases the features of TSTL by focusing on one large-scale testing effort, directed by an end-user, to find in faults the most widely used Geographic Information System (GIS) tool, Esri's ArcGIS, using its ArcPy scripting interface. We also demonstrate how TSTL makes prototyping and evaluating new test generation algorithms easy for researchers, and briefly discuss other testing efforts using TSTL.

F. Author
first address
Tel.: +123-45-678910
Fax: +123-45-678910
E-mail: fauthor@example.com

S. Author
second address

Keywords Software testing · Domain-specific languages · Explicit-state model checking

1 Introduction

Software test automation can be divided into two aspects: automated execution and determination of results for existing, human-created tests, and the automatic generation of tests, based on some definition of valid tests. Both are critical for effective, efficient software testing, but only automatic generation offers the promise of discovering faults without human determination that a particular execution scenario has the potential to behave incorrectly. This paper presents a case study of applying a new automated test case generation language and toolchain to a large, real-world software system. The test effort has been driven and directed not by a software testing researcher (as is the usual case), but by a domain expert in the Geographic Information System (GIS) Software Under Test (SUT). In the process, multiple faults and undocumented restrictions of the API under test have been discovered.

1.1 Testing Esri ArcPy

Esri is the single largest GIS software vendor, with about 40% of global market share. Esri's ArcGIS tools are extremely widely used for GIS analysis, in government, scientific research, commercial enterprises, and education. Automation of complex GIS analysis and data management is essential, and Esri has long provided tools for programming their GIS software tools. The newest such method, introduced in ArcGIS 10.0, is a Python site-package, ArcPy [23]. ArcPy is a complex library, with dozens of classes and hundreds of functions distributed over a variety of of toolboxes. Most of the code executed in carrying out ArcPy functions is the code for the ArcGIS engine itself. This source code, written in C++, is not available. The source code for the latest version (10.3) of the Python site-package alone, however, which interfaces with the ArcGIS engine, is over 50,000 lines of code. This is a very large system (especially given the compactness of Python code), comparable in size to the largest software systems previously tested using automated test generation, such as core Java and Apache libraries [25,51].

In order to improve the reliability of ArcPy, we are developing a framework for automated testing of ArcPy itself, as well as libraries based on ArcPy. The source code for defining the structure of ArcPy tests, written in the TSTL[1][38, 63,40] domain-specific language [24] for testing, is already more than six times as large as the next-largest such definition previously implemented in TSTL, even though the harness so far only includes a small portion of ArcPy API (Application Program Interface) calls. The first stage of testing has resulted in discovery of multiple faults in ArcPy/ArcGIS, and has required modifications

[1] TSTL stands for Template Scripting Testing Language.

to the TSTL language and, especially, to the toolchain supporting test replay, debugging, and test case understanding.

One of the contributions of this paper is a more in-depth discussion of the problems, challenges, and tool utility aspects of testing software than is typical in most research papers in the field. Such papers are (understandably) typically focused on novel algorithms or empirical evaluations of known methods, rather than the practical aspects of finding and understanding faults in a real-world software system.

1.2 Automated Testing for the Rest of the World

Previous work on automated testing for APIs has been largely carried out by software testing researchers only, or (at most) by software testing researchers working with individuals who are primarily software developers. This paper presents work largely directed by the first author, who is not a software developer by profession or education, but a GIS analyst.

The problem of end-user testing [12,13,53] is known to be difficult. Previous work has focused on end-users of systems that are not traditional programming languages: e.g. spreadsheets [53], visual languages, or machine-learning systems [37]. This paper aims to describe a case study in how a user who is familiar with a software library but not expert in software testing techniques can use (and adapt) existing tools to test a traditional software API library. In one sense, this is a less difficult end-user testing scenario than testing spreadsheets or visual forms, in that the testing is directed by an individual used to writing and thinking about software source code. The concepts in modern automated software testing are most easily understood by those who are also familiar with the structure and semantics of conventional programming languages. On the other hand, the system to be tested is large, not a small user-developed program: ArcPy is a modern, complex, library, comparable in complexity and size to core language libraries. ArcPy was also not written by the end-user, or by any of the authors of this paper, nor have the authors received any assistance in the effort from Esri.

Automated testing systems more advanced than a simple hand-written loop generating a few random inputs to a handful of functions, or more complicated to use than a fully push-button system are often considered too difficult for practical use even by software developers or software QA staff [31]. In fact, in the experience of the second author of this paper (an academic software testing researcher), even “push-button” tools for automated testing are often difficult for expert users to install, apply, and configure [32,35,31]. Such tools typically do not approach the ease-of-use seen in commercial static analysis tools [2,3,1]. One goal of this work has been to mature the TSTL language and toolchain so Python programmers from all backgrounds can easily apply it to their automated testing problems.

The second major contribution of this paper is therefore a presentation of an approach to automated testing that has been chosen by a GIS analyst, not a software developer or testing researcher. Moreover, we present this paper as a proof-of-concept that modern automated testing, even in a highly interactive, non-push-button form, can be used by a motivated domain expert, with the support of a domain-specific language [24] and a set of tools for generating, analyzing, and replaying tests.

2 Related Work

There is a vast amount of previous work on automated generation of tests for (API-based) software systems [51,25,28] and random testing in particular [34,51,9,17,10,56,42,41,19,18,11,36,8,7,22,61], some dating back to the early 1980s. It is far beyond the scope of this paper to explore that literature in detail. The interested reader is directed to the cited papers, as well as general surveys of automated test generation in particular [5] or recent software testing research in general [50].

To our knowledge, there has been no previous proposal of a concise domain-specific-language [24] like TSTL, to assist users in building test harnesses. One line of related work is previous work on building common frameworks for random testing and model checking [36] and proposing common terminology for imperative harnesses [30]. Earlier publications on TSTL [38,39] presented a language considerably more limited in functionality and with a more difficult-to-read syntax than the presentation in this paper, and omitted details of the tools provided in the TSTL distribution for off-the-shelf testing.

There exist various testing tools and languages of a somewhat different flavor: e.g. Korat [48], which has a much more fixed input domain specification, or the tools built to support the Next Generation Air Transportation System (NextGen) software [26]. The closest of these is the UDITA language [27], an extension of Java with non-deterministic choice operators and `assume`, which yields a very different language that shares our goal. TSTL aims more at the *generation* of tests than the *filtering* of tests (as defined in the UDITA paper), while UDITA supports both approaches. This goal of UDITA (and resulting need for first-class `assume` statements) means that it must be hosted inside a complex (and sometimes non-trivial to install/use) tool, JPF [58], rather than generating a stand-alone simple interface to a test space, as with TSTL. Building "UDITA" for a new language is far more challenging than porting TSTL. UDITA also supports far fewer constructs to assist test harness development.

The design of the SPIN model checker [44] and its model-driven extension to include native C code [43] inspired the flavor of TSTL's domain-specific language, though our approach is more declarative than the "imperative" model checker produced by SPIN. Similarly, work at JPL on languages for analyzing spacecraft telemetry logs in testing [33] provided a working example of a Python-based declarative language for testing purposes. The pool approach to test case construction is derived from work on canonical forms and enumeration of unit tests [6].

There is relatively little written about software testing in the academic literature that focuses on the pitfalls, best practices, and engineering challenges of testing real-world systems. The series of papers by the NASA Jet Propulsion Laboratory on testing the file systems for the Curiosity Mars Rover is notable, though even there the focus is more on general technique than details of approach [34,35,32]. Literature that at least touches on more practical aspects, however, is plentiful: e.g., Lei and Andrews [46] demonstrated that random test generation essentially requires test-case minimization [62] in order to be useful. Recent work, in particular, has given more attention to issues of test case usability than in the past; we suspect this shows a growing technological maturity for automated test case generation. For example, we found that test case readability was important in our efforts, and recent academic work [21,20] has introduced systematic, principled methods for improving the readability of test cases for humans, even though this does not improve fault detection, coverage, or other traditional measures of test effectiveness.

The literature on testing GIS (Geographic Information System) software in particular seems to consist of one paper proposing a very limited application of automated testing to assist GIS users, primarily in model development [47]. That work does not target the reliability or correctness of the underlying GIS engine, or GIS libraries. There is also some discussion of automated testing for GIS in various blog posts and discussion groups (e.g., [59,4]), but no formal academic case studies. These discussions also tend to focus on application testing or GUI testing, rather than testing of library code used across GIS applications. There is a simple extension to Python unit testing modules for the GRASS open source GIS system [29], but this does not provide any automated test generation.

There is a significant body of work on end-user testing of software, part of the larger field of end-user software engineering [12,13]. End-user software engineering examines how software can best be produced by developers who do not have a traditional computer science background, and are often primarily interested in an application of programming, rather than software development as a profession. GIS developers are (we believe) a typical example [55]: they are technically skilled individuals whose primary expertise is not in software development, but who, in order to pursue their goals, must develop, maintain, and test significant software systems.

The earliest work focusing on software testing for end-user software engineers explored how to test spreadsheets [53,54]. Other work has focused on errors end-users make in specifying systems [52], and how end-users of machine learning systems (who may be machine learning experts, or individuals with no programming knowledge at all) can test such systems [37,45,57]. To our knowledge, no previous work considers API-sequence testing for end-users. Previous work on API testing has largely not even included traditional software developers, but been performed by software testing researchers only.

Chen et al. propose metamorphic testing [15,49,60,14] as a possible solution to the oracle problem (defining correct behavior of a software system) for end-user programmers [16]. In metamorphic testing, while the correct output of a program is not known, the effect of certain modifications to an input is known (e.g., increasing a certain input should increase a certain output), allowing faults to be detected even without a definition of the correct result. In this paper, we largely focus on building a basic framework that works for crash bugs and uncaught exceptions, and is capable of extension to more complex correctness properties. In the long run, however, more sophisticated oracles are critical to finding deep faults.

References

1. CodeSonar: GrammaTech static analysis. http://www.grammatech.com/products/codesonar
2. Software development testing and static analysis tools: Coverity. http://www.coverity.com
3. Source code analysis tools for software security & reliability: Klockwork. http://klocwork.com
4. AbSharma: Functional testing of GIS applications (automated testing). http://osgeo-org.1560.x6.nabble.com/Functional-Testing-of-GIS-applications-Automated-Testing-td4493673.html
5. Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., McMinn, P.: An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software **86**(8), 1978–2001 (2013)
6. Andrews, J., Zhang, Y.R., Groce, A.: Comparing automated unit testing strategies. Tech. Rep. 736, Department of Computer Science, University of Western Ontario (2010)
7. Andrews, J.H., Groce, A., Weston, M., Xu, R.G.: Random test run length and effectiveness. In: Automated Software Engineering, pp. 19–28 (2008)
8. Andrews, J.H., Haldar, S., Lei, Y., Li, C.H.F.: Tool support for randomized unit testing. In: Proceedings of the First International Workshop on Randomized Testing, pp. 36–45. Portland, Maine (2006)
9. Andrews, J.H., Menzies, T., Li, F.C.: Genetic algorithms for randomized unit testing. IEEE Transactions on Software Engineering (TSE) **37**(1), 80–94 (2011)
10. Arcuri, A., Briand, L.: Adaptive random testing: An illusion of effectiveness. In: International Symposium on Software Testing and Analysis, pp. 265–275 (2011)
11. Arcuri, A., Iqbal, M.Z.Z., Briand, L.C.: Formal analysis of the effectiveness and predictability of random testing. In: International Symposium on Software Testing and Analysis, pp. 219–230 (2010)
12. Burnett, M., Cook, C., Rothermel, G.: End-user software engineering. Comm. ACM **47**(9), 53–58 (2004)
13. Burnett, M.M., Myers, B.A.: Future of end-user software engineering: beyond the silos. In: Future of Software Engineering, pp. 201–211 (2014)
14. Chen, T., Tse, T., Quan Zhou, Z.: Fault-based testing without the need of oracles. Information and Software Technology **45**(1), 1–9 (2003)
15. Chen, T.Y., Cheung, S.C., Yiu, S.: Metamorphic testing: a new approach for generating next test cases. Tech. Rep. HKUST-CS98-01, Hong Kong Univ. Sci. Tech. (1998)
16. Chen, T.Y., Kuo, F.C., Zhou, Z.Q.: An effective testing method for end-user programmers. In: Proceedings of the First Workshop on End-user Software Engineering, pp. 1–5 (2005)
17. Chen, T.Y., Leung, H., Mak, I.K.: Adaptive random testing. In: Advances in Computer Science, pp. 320–329 (2004)
18. Ciupa, I., Leitner, A., Oriol, M., Meyer, B.: Experimental assessment of random testing for object-oriented software. In: D.S. Rosenblum, S.G. Elbaum (eds.) International Symposium on Software Testing and Analysis, pp. 84–94. ACM (2007)
19. Claessen, K., Hughes, J.: QuickCheck: a lightweight tool for random testing of haskell programs. In: ICFP, pp. 268–279 (2000)

20. Daka, E., Campos, J., Dorn, J., Fraser, G., Weimer, W.: Generating readable unit tests for Guava. In: Search-Based Software Engineering - 7th International Symposium, SSBSE 2015, Bergamo, Italy, September 5-7, 2015, Proceedings, pp. 235–241 (2015)
21. Daka, E., Campos, J., Fraser, G., Dorn, J., Weimer, W.: Modeling readability to improve unit tests. In: Foundations of Software Engineering, ESEC/FSE, pp. 107–118 (2015)
22. Duran, J.W., Ntafos, S.C.: Evaluation of random testing. IEEE Transactions on Software Engineering **10**(4), 438–444 (1984)
23. Esri: What is ArcPy? http://resources.arcgis.com/EN/HELP/MAIN/10.1/ index.html000v000000v7000000
24. Fowler, M.: Domain-Specific Languages. Addison-Wesley Professional (2010)
25. Fraser, G., Arcuri, A.: EvoSuite: automatic test suite generation for object-oriented software. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11, pp. 416–419. ACM (2011)
26. Giannakopoulou, D., Howar, F., Isberner, M., Lauderdale, T., Rakamarić, Z., Raman, V.: Taming test inputs for separation assurance. In: International Conference on Automated Software Engineering, pp. 373–384 (2014)
27. Gligoric, M., Gvero, T., Jagannath, V., Khurshid, S., Kuncak, V., Marinov, D.: Test generation through programming in UDITA. In: International Conference on Software Engineering, pp. 225–234 (2010)
28. Godefroid, P., Klarlund, N., Sen, K.: DART: directed automated random testing. In: Programming Language Design and Implementation, pp. 213–223 (2005)
29. GRASS Development Team: Testing GRASS GIS source code and modules. https://grass.osgeo.org/grass71/manuals/libpython/gunittest_testing.html
30. Groce, A., Erwig, M.: Finding common ground: choose, assert, and assume. In: Workshop on Dynamic Analysis, pp. 12–17 (2012)
31. Groce, A., Fern, A., Pinto, J., Bauer, T., Alipour, A., Erwig, M., Lopez, C.: Lightweight automated testing with adaptation-based programming. In: IEEE International Symposium on Software Reliability Engineering, pp. 161–170 (2012)
32. Groce, A., Havelund, K., Holzmann, G., Joshi, R., Xu, R.G.: Establishing flight software reliability: Testing, model checking, constraint-solving, monitoring and learning. Annals of Mathematics and Artificial Intelligence **70**(4), 315–349 (2014)
33. Groce, A., Havelund, K., Smith, M.: From scripts to specifications: The evolution of a flight software testing effort. In: International Conference on Software Engineering, pp. 129–138 (2010)
34. Groce, A., Holzmann, G., Joshi, R.: Randomized differential testing as a prelude to formal verification. In: International Conference on Software Engineering, pp. 621–631 (2007)
35. Groce, A., Holzmann, G., Joshi, R., Xu, R.G.: Putting flight software through the paces with testing, model checking, and constraint-solving. In: Workshop on Constraints in Formal Verification, pp. 1–15 (2008)
36. Groce, A., Joshi, R.: Random testing and model checking: Building a common framework for nondeterministic exploration. In: Workshop on Dynamic Analysis, pp. 22–28 (2008)
37. Groce, A., Kulesza, T., Zhang, C., Shamasunder, S., Burnett, M.M., Wong, W., Stumpf, S., Das, S., Shinsel, A., Bice, F., McIntosh, K.: You are the only possible oracle: Effective test selection for end users of interactive machine learning systems. IEEE Trans. Software Eng. **40**(3), 307–323 (2014)
38. Groce, A., Pinto, J.: A little language for testing. In: NASA Formal Methods Symposium, pp. 204–218 (2015)
39. Groce, A., Pinto, J., Azimi, P., Mittal, P.: TSTL: a language and tool for testing (demo). In: ACM International Symposium on Software Testing and Analysis, pp. 414–417 (2015)
40. Groce, A., Pinto, J., Azimi, P., Mittal, P., Holmes, J., Kellar, K.: TSTL: the template scripting testing language. https://github.com/agroce/tstl
41. Hamlet, R.: Random testing. In: Encyclopedia of Software Engineering, pp. 970–978. Wiley (1994)
42. Hamlet, R.: When only random testing will do. In: International Workshop on Random Testing, pp. 1–9 (2006)

43. Holzmann, G., Joshi, R.: Model-driven software verification. In: SPIN Workshop on Model Checking of Software, pp. 76–91 (2004)
44. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional (2003)
45. Kulesza, T., Burnett, M., Stumpf, S., Wong, W.K., Das, S., Groce, A., Shinsel, A., Bice, F., McIntosh, K.: Where are my intelligent assistant's mistakes? a systematic testing approach. In: Intl. Symp. End-User Development, pp. 171–186 (2011)
46. Lei, Y., Andrews, J.H.: Minimization of randomized unit test cases. In: International Symposium on Software Reliability Engineering, pp. 267–276 (2005)
47. Maogui, H., Jinfeng, W.: Application of automated testing tool in gis modeling. In: Proceedings of the 2009 WRI World Congress on Software Engineering, pp. 184–188 (2009)
48. Milicevic, A., Misailovic, S., Marinov, D., Khurshid, S.: Korat: A tool for generating structurally complex test inputs. In: International Conference on Software Engineering, pp. 771–774 (2007)
49. Murphy, C., Shen, K., Kaiser, G.: Automatic system testing of programs without test oracles. In: Intl. Symp. Software Testing and Analysis, pp. 189–200 (2009)
50. Orso, A., Rothermel, G.: Software testing: A research travelogue (2000–2014). In: Proceedings of the on Future of Software Engineering, FOSE 2014, pp. 117–132 (2014)
51. Pacheco, C., Lahiri, S.K., Ernst, M.D., Ball, T.: Feedback-directed random test generation. In: International Conference on Software Engineering, pp. 75–84 (2007)
52. Phalgune, A., Kissinger, C., Burnett, M., Cook, C., Beckwith, L., Ruthruff, J.: Garbage in, garbage out? an empirical look at oracle mistakes by end-user programmers. In: IEEE Symp. Visual Languages and Human-Centric Computing, pp. 45–52 (2005)
53. Rothermel, G., Burnett, M., Li, L., DuPois, C., Sheretov, A.: A methodology for testing spreadsheets. ACM Trans. Software Eng. and Methodology **10**(1), 110–147 (2001)
54. Rothermel, K., Cook, C., Burnett, M., Schonfeld, J., Green, T., Rothermel, G.: WYSIWYT testing in the spreadsheet paradigm: An empirical evaluation. In: Intl. Conf. Software Eng., vol. 22, pp. 230–240 (2000)
55. Segal, J.: Some problems of professional end user developers. In: IEEE Symp. Visual Languages and Human-Centric Computing (2007)
56. Sharma, R., Gligoric, M., Arcuri, A., Fraser, G., Marinov, D.: Testing container classes: Random or systematic? In: Fundamental Approaches to Software Engineering, pp. 262–277 (2011)
57. Shinsel, A., Kulesza, T., Burnett, M.M., Curan, W., Groce, A., Stumpf, S., Wong, W.K.: Mini-crowdsourcing end-user assessment of intelligent assistants: A cost-benefit study. In: IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 47–54 (2011)
58. Visser, W., Havelund, K., Brat, G., Park, S., Lerda, F.: Model checking programs. Automated Software Engineering **10**(2), 203–232 (2003)
59. XBOSOFT: GIS software testing - lessons learned. http://xbosoft.com/gis-software-testing-lessons-learned/
60. Xie, X., Ho, J., Murphy, C., Xu, B., Chen, T.Y.: Application of metamorphic testing to supervised classifiers. In: Intl. Conf. Quality Software, pp. 135–144 (2009)
61. Yang, X., Chen, Y., Eide, E., Regehr, J.: Finding and understanding bugs in C compilers. In: ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 283–294 (2011)
62. Zeller, A., Hildebrandt, R.: Simplifying and isolating failure-inducing input. Software Engineering, IEEE Transactions on **28**(2), 183–200 (2002)
63. Zhang, C., Groce, A., Alipour, M.A.: Using test case reduction and prioritization to improve symbolic execution. In: International Symposium on Software Testing and Analysis, pp. 160–170 (2014)