Dear editor and anonymous reviewers,

We would like to thank the editor and anonymous reviewers for their helpful comments. The paper overall has been heavily revised, and we hope that all typos and similar faults in readability have been removed. We added about 30 new references to more adequately address related work, and about 10 pages of additional material to more effectively describe TSTL and explain contributions of the paper. Section 6 of the paper clarifies the original contribution of TSTL to testing, and explores ideas completely original to this paper. Below we address specific comments of each reviewer.

1 Editor's Report

> Both reviewers express quite some reservations against the paper. These concern the very limited discussion of related work, the lack of a discussion of the limitation of the approach (what is easy to generalize, and which limitation are inherent in the approach?), and a clear description of the benefits for a potential tester (what does he need to do, what becomes easier?). The latter is particular important, if the paper is intended to also serve as a TSTL cookbook. Thus the paper can only be accepted under the condition of some major revision.

As noted, references have been greatly expanded and the discussion of related work is revised to better place TSTL in the literature. The limitations are now made clear in Section 4.4 on TSTL and other languages, and a new emphasis on the original approach to testing made possible in TSTL is emphasized. The benefits in terms of interactive testing, ease-of-use, and practical terms such as customized regression should now be clear.

2 Reviewer 1

> The paper proposes TSTL, a concise domain specific-language to assist users in building test harnesses. It includes a compiler to Python and a suite of tools for generating, manipulating and analyzing test cases as well as the possibility to use model checkers. The paper is difficult to follow. The concepts are fragmented. I do not expect to see the main contributions of the paper as subsections (sec 1.1, sec 1.2 and sec 1.3) of the introduction section. In section 1.3 the authors discuss about the results of the experimentation whereas the case study is presented in section 2. Moreover, it is not clear which are the goals addressed by this language.

The beginning of the paper has been re-organized to a more conventional presentation of contributions and introductory material. The goals of the lan-

guage should be clearer now, in part due to the re-organization. We also better describe the structure of the (now quite lengthy) paper at the end of the introduction.

> Another limitation of the approach is that it can be applied only for Python development. The actions of TSTL, as in the examples, are very simple including only assignments and integer addition. Is it possible to enable more complex actions? And also generated test cases are very simple and deal with random tests generation. How can the tester include new test generation strategies? What about the expected output? There are facilities helping the tester to define the expected output? Finally, it is not clear which is the main improvement of the TSTL version presented in this paper with respect to the previous ones.

The potential for application to other languages is now explicitly addressed in Section 4.4 (and elsewhere). The paper now presents the aims of TSTL and core nature (vs. Python specifics) better, which we hope helps with these questions.

We clarify that actions can be arbitrary fragments of code in any underlying language: (some of our) examples are simpler just to make things easier for the reader. Other examples do include code that goes far beyond assignments and integer addition, e.g., calling GIS operations or modifying a global structure used to check a property. Generated test cases are, we emphasize, arbitrary sequences of arbitrary fragments of code in a language, which is quite general as shown by Andrews et. al and assumed in other tools such as Randoop. See Section 4.1 for a revised explanation of TSTL generality.

New generation strategies are discussed at more length, including how to write some generators in a functional style (Section 6). We note this is aimed at property-directed testing, where the user supplies general properties (as in QuickCheck, or model checking for that matter), rather than specific outcome values. Expected output can also be determined using a reference implementation (differential testing) (Section 4.2). However, is a user wishes to add hand-written value oracles, the standalone test production facility can be used (Section 5.3) and there is a utility to capture current values of all pools in such a test, for use in regression testing.

> More than 15 references are of the same authors and 15 references are about random testing even if the authors claim that test case generation is not the main focus of the paper. Please, fix some language typos.

We now (we think) make the improvements over past papers (and versions of TSTL) more clear. The paper now has close to 100 references, and we tried to fix all typos.

3 Reviewer 2

> The paper presents TSTL (Template Scripting Testing Language), a domain-specific language (DSL) for writing test harnesses, i.e. an approach to define a set of correctness properties in automated test generation.
>
> As the approach is tool-based and formal, it fits for STTT. The paper is written understandable, but its contribution has to be highlighted more clearly. It is especially needed to also position the approach in context of related work. In the contribution section the paper defines itself as "we showcase the TSTL approach" and "This paper thus serves as a basic 'TSTL cookbook'". This is ok for a paper on a tool demo paper, but not for a journal paper. I also miss more details on the applied model checking approach.

We believe the contributions beyond those of a tool paper are now clear: the language-interface approach and methods of TSTL are, to our knowledge, quite novel. The cookbook aspect of the paper is present for users (which fits STTT aims, we think), but the organization emphasized more conceptual contributions and ideas. There is much more detail on model checking, in Section 5.2 now, including how abstraction works, the core of a simple BFS exploration, and some comments on how to implement LTL property checking in TSTL.

> The related work is by far too limited and too much focused on the GIS domain (probably due to the background of the authors). It misses many approaches that can be considered as DSLs for testing. At least the following types of approaches are missing: model-based testing approaches (Utting, Mark, Alexander Pretschner, and Bruno Legeard. "A taxonomy of model-based testing approaches." Software Testing, Verification and Reliability 22.5 (2012): 297-312.), Behavior Driven Development (Chelimsky, David, et al. The RSpec book: Behaviour driven development with Rspec, Cucumber, and friends. Pragmatic Bookshelf, 2010.), and special DSL-based approaches like Telling TestStories (Felderer, Michael, et al. "A tool-based methodology for system testing of service-oriented systems." Advances in System Testing and Validation Lifecycle (VALID), 2010 Second International Conference on. IEEE, 2010.)

These — and many other (over 30) — references have been added, and we hope the paper now makes the place of TSTL in the literature more clear. Thanks for the references!

> The evaluation presented in Sectioin 6 (Faults Discovered Using TSTL) is not convincing. Are the found faults, faults which could not have been found otherwise. Even more interesting in case of a TSTL is the acceptance by its users and its return on investment.

> How does the approach handle evolution of the system? How would regerssion testing be addressed.

We believe some of the faults would be easy to accidentally run into during use of ArcPy (which is why discovering and documenting them is particularly useful), but very difficult to succinctly capture in clear test cases to report. The GIS users involved in this paper found TSTL useful and a good return on investment, but of course this is not an unbiased argument. We think the key here is not a heavy empirical/human study evaluation, but genuinely novel approaches to DSLs for testing and to testing in general. We discuss other testing efforts in somewhat more detail now, and include more faults detected using TSTL.

Evolution of the system is handled by various regression testing strategies. We added a new section, 5.3.1, discussing regression tools developed for ArcPy but useful elsewhere, and note that since TSTL test cases can easily be saved, replayed, made stand-alone, modified to include additional assertions, and augmented with expected results from a past version, TSTL is perhaps unusually full-featured in terms of regression capabilities compared to many tools in the field. Figure 7 shows how to write a simple "regression check" over stored tests in very few lines of code. Custom prioritizations would also be easy, as noted.

Sincerely,
Josie Holmes, Alex Groce, Jervis Pinto, Pranjal Mittal, Pooria Azimi, Kevin Kellar, and James O'Brien