Dear Tien Nguyen,  
   
We are pleased to inform you that your paper  
   
Interaction-based Tracking of Program Entities for Test Case Evolution  
(paper ID: icsme-research-167)  
   
has been accepted for inclusion in the research track of the 33rd IEEE International Conference on Software Maintenance and Evolution (ICSME), to be held in Shanghai, China, September 20-22, 2017.  
   
The program committee selected 42 of 151 submissions (27.8%) for inclusion in the research program. All submissions were reviewed by at least three members of the program committee and discussed online.  
   
We enclose the reviews of your paper. In your preparation of the final paper, please make sure to incorporate the suggestions of the reviewers.    
   
IMPORTANT INFORMATION:   
   
1. Each accepted paper must have at least one author registered at a professional member rate by July 9, 2017. Papers for which the only registered authors are students must have one of the students register at the full (i.e., non-student) rate. The registration site will be open soon. In the registration form, please indicate the paper ID: icsme-research-167  
   
2. The camera-ready paper is due on August 3, 2017. The information about the IEEE online author kit will be sent to you in a separate email once it is available. Papers must not exceed 10 pages (including figures and appendices) plus up to 2 pages that contain ONLY references.  
   
3. Got artifacts? The ICSME artifacts track promotes, celebrates, and catalogs excellent  research artifacts in software engineering. Submit your artifact by June 26 to receive special recognition and an extra age in the proceedings to describe the artifact. <https://icsme2017.github.io/cfp/ArtifactsTrackCFP.html>  
   
4. The deadlines for New Ideas and Emerging Result, Industry Papers, Tool Demos, and the Doctoral Symposium are approaching. Abstracts are due June 19, paper are due June 26.  
   
5. We encourage authors to use open science practices to make their research and data accessible to anybody in the world. Here are guidelines and recommendations for open access, open data, and open source. <https://icsme2017.github.io/cfp/OpenSciencePractices.html>   
   
Congratulations again on your paper acceptance. We look forward to seeing you in ICSME 2017 in Shanghai!  
   
Best regards,  
   
Lu Zhang and Thomas Zimmermann  
ICSME 2017 Program Co-chairs  
  
<https://icsme2017.github.io/>  
  
  
----------------------- REVIEW 1 ---------------------  
PAPER: 167  
TITLE: Interaction-based Tracking of Program Entities for Test Case Evolution  
AUTHORS: Hoan Nguyen, Tung Nguyen, Tien Nguyen and Hung Nguyen  
  
  
----------- Overall evaluation -----------  
1. Paper summary  
The paper proposes an approach to matching entities from different versions of the same application. It differs from other similar work in that it matches entities based on their interaction similarity, i.e., how they interact with other entities. Based on the matching results, the paper suggests how to fix obsolete test cases by redirecting class references or method calls. Finally, a case study is presented to evaluate the approach, and results suggest that the approach is more accurate than related approach.  
  
2. Points for the paper  
.Well-structured and easy to followed .  
  
3. Points against the paper  
.The motivation should be restructured. It is not clear why test cases (especially unit testing) should be updated separately? I suppose that such test cases should be updated before production code evolves. With automatic refactoring tools, renaming a method or class in production code would result in automatic updating of its reference, including those in test cases. Consequently, why should we manually update the test cases?  
  
.Once roles (responsibility) of methods are changes, updating the references in test cases is often not enough. Much more edition is needed, e.g., to satisfy coverage criteria. Consequently, even with automatic updating of references, developers or testers should manually modify corresponding test cases.   
  
.For a depressed method that is to be replaced with a new way, redirecting its references in test cases to the new method is not a good solution. The old test cases is still needed to validate that no new error is introduced concerning the old method (since the old method is available for existing clients as well, we should make sure such clients work well on the new version).   
  
.Attributes and interaction have been defined for classes. However, they should be defined for methods as well.   
  
.page 7: the accuracy defined in the paper is in fact RECALL.   
  
. The number of test cases involved in the evaluation is small, making the evaluation results less convictive.    
  
.While measuring the performance of entity matching, the paper considers only test case related entities. It may be bias and potentially unfair for KPW that is not designed for test case repair.   
  
.For the examples in Fig5 and Fig 6, can your approach suggest how to set the arguments?   
  
.For the example Fig5 and Fig 6, I am not sure the new methods are UPDATING of the old ones or just brand new methods. Could you send questionnaires to the developers for confirmation?  
  
.Why not compare the approach against BEAGLE that is closely related to the proposed approach?  
  
  
----------------------- REVIEW 2 ---------------------  
PAPER: 167  
TITLE: Interaction-based Tracking of Program Entities for Test Case Evolution  
AUTHORS: Hoan Nguyen, Tung Nguyen, Tien Nguyen and Hung Nguyen  
  
  
----------- Overall evaluation -----------  
1. Paper summary  
This paper presents iTrack, an approach (i.e., series of algorithms) to identify potential candidates in the code to be used for updating/correcting method calls in broken test cases. The approach is based on name similarity, interaction (dependencies) similarity, and other heuristics such as clone analysis. The approach was tested with an oracle that was developed from the analysis of two OSS: JFreeChart and JTopas and was compared to another tool that performs a similar task i.e., KPW.  
  
2. Points for the paper  
+ Is tackling a relevant problem (broken tests during code evolution)  
+ Rationale behind the algorithms is well explained  
  
3. Points against the paper  
- Evaluation is rather preliminary  
- No details on implementation  
- Not clear how the approach can be integrated in dev's work-flow  
  
4. Supporting argumentation for your points  
\*\*\*Evaluation is rather preliminary\*\*\*  
As the author(s) themselves state it, the oracle is not very comprehensive (two systems) and no information is given on how this sample is representative/similar to other systems. Also, one should ideally report both precision and recall when evaluating classifiers or in recommender systems. Is also not clear why the approach was (only) compared to KPW, and not instead/in addition to iDiff? Finally, the comparison between iTrack and related work can definitively be better structured and explained in order to enable a fair comparison.  
  
\*\*\*No details on implementation\*\*\*  
There is absolutely no information on how the algorithms were implemented in order to perform the evaluation, also, some details/rationale on thresholds are not well explained (e.g., why it was decided that the threshold for interaction similarity should be 0.75?).  
  
\*\*\*Not clear how the approach can be integrated in dev's work-flow\*\*\*  
By looking at the analysis presented in the Scalability section, I wonder if the authors could explain better in which part of the working flow the tool is intended to be used? as a batch? as a on-the-fly recommender tool (e.g., hot-fixes?) how are these different scenarios going to affect the suitability of the approach? For example, is the code meant as a "refactoring" recommender while performing changes in the production code or it is after unit tests are run and the developer needs to figure out how to fix the broken tests? Depending of the scenario and how is implemented, 15 minutes latency could or not be, an acceptable timeframe.  
  
5. Suggested paper improvements  
Other improvements besides the above mentioned ones are:  
- Specify earlier (intro) what exactly you mean by interaction  
- Differentiate clearly (conceptually) between attribute and interaction (the distinction is not clear and is used together in several parts of the paper)  
- Provide a reference for a multigraph  
- The example for definition 2 is not clear (i.e. how we arrived at the numbers used in the example)  
- Why SimpleMatch is referred twice in the beginning of Section IV? (i.e. as the main algorithm and greedy algorithm) maybe some text restructuring can make the paragraph a bit more clear.  
- Please discuss a bit more in detail what is the major reason behind the limitation that you point about iTrack with respect to BEAGLE.  
  
  
----------------------- REVIEW 3 ---------------------  
PAPER: 167  
TITLE: Interaction-based Tracking of Program Entities for Test Case Evolution  
AUTHORS: Hoan Nguyen, Tung Nguyen, Tien Nguyen and Hung Nguyen  
  
  
----------- Overall evaluation -----------  
== Paper summary ==  
The paper presents a technique, called iTrack, to keep unit tests synchronized with changes to the code subject to tests. In other words, the approach tries to automatize the cases where tested code has been modified (e.g. in terms of invoked methods / used classes), the test code results to be broken, and needs to be fixed to perform, for example, regression testing to the new version. The core algorithm of iTrack is an iterative algorithm that matches program entities across versions, identifying entities with the same role leveraging the similarity of their interactions with other entities (i.e., two entities that among two versions have similar interaction with other entities are likely to be the same entity). The evaluation shows very promising results on the ability of fixing test cases after changes in the tested classes, achieving from 84% to 99% accuracy in identifying calls that need to be changed to fix the tests, improving the state of the art of the approach by !  
 Kim, Pan, and Whitehead (KPW [21]).  
  
  
== Points for the paper ==  
+ Important and interesting problem regarding test and tested code evolution, and automatic fixing of test cases that result to be broken after evolution  
+ Technique proposed to solve the problem is interesting and well explained in the paper, from key concepts, similarity computation, and matching algorithms  
+ Good motivating examples   
+ Extremely good performance of the approach as illustrated in the evaluation  
  
== Points against the paper ==  
- Evaluation is done on a relatively small benchmark dataset, and it is unclear how it has been selected.  
  
== Detailed Review and Suggestion for Improvement ==  
The paper is pretty much clear and readable and I found it very well structured. The introduction is very well discussed and I particularly appreciated the motivating examples with the discussion and implications on Section II. The technical core of iTrack is clear and appears to be sound as well, illustrated through Section III - the formulation of key concepts like similarity between entities, and Section IV, with the discussion of the algorithm from the design ideas, the simple matching algorithm, the system matching, and the recommendation.   
The evaluation, in the end, shows very effective results, and I liked the case discussions at the end. However, I have some minor criticisms on it.  
  
-) Issues on Evaluation  
One could argue that considering only 4 pairs of versions coming from 2 projects is a relatively limited form of evaluation, and I might agree. In the case of the paper, it seems like the approach works very effectively on its purpose for these cases and this results to be in any case a form of assessment of the approach that is valuable.   
However, one could also argue that the two projects and the two versions have been ‘cherrypicked’ to show a simple case where the approach is extremely effective. I do not think that this is the case, but it would add value to the paper to describe the selection process for the project and the two versions (that justifies their importance). Moreover, I do not understand why only one element in the benchmark dataset is coming from the motivating examples. In fact, I kind of expected to find also the JHotDraw example in the evaluation.  
At a certain point I wondered if the benchmark was based from some other related approach, but I could not find evidence for it. It seems that the benchmark is not constructed, for example, from [21]. Was it already available anywhere, or has it been manually constructed for this paper?  
  
Solution for improvement: Illustrate with a dedicated paragraph the selection process for the projects and the specific versions among all the available ones. Even better - it would be good to provide more evidence in a replication package, potentially containing a bigger evaluation.