On Recommending Actionable Changes For Software Projects

Rahul Krishna
NC State University
i.m.ralk@gmail.com

1 CRITERIA

In order to assess this project the following criteria will be explored. The proposed software analysis tool to be used in this project is XTREE. Listed below are some of the criteria that are used to assess a software analytics tool. Of these, some of the criteria are already satisfied by XTREE. The focus of this project will be to attempt to address some of relevant aspects from below that are not currently supported.

(1) ***Model Readability:*** The proposed software analysis tool to be used in this project (XTREE) address this problem by constructing a decision tree and evaluating possible solutions by reasoning across the leaves of that decision tree. This decision tree approach greatly assists model readability.
(2) ***Actionable Conclusions:*** Generation of conclusions are an inherent property of XTREE. But, the key question with the first version of XTREE is that some of the conclusions were not really actionable. So, in this work, we explore ways in which XTREE can limit it's conclusions to only actionable ones.
(3) ***Learnability and repeatability of the results:*** Memory consumption of XTREE is quite low for small projects. So, this criteria has been addressed in the tool.
(4) ***Multi-goal reasoning:*** This is another area that XTREE needs enhancement. However, in the interest of time, this extension is left to future work.
(5) ***Anomaly detection:*** XTREE is perfectly suited to detect anomalous data. Once the decision tree has been constructed for XTREE, we can easily detect anomalies by reflecting on the tree. That is, if a new data point has features that are not in the tree, then it is possibly worth closer inspection.
(6) ***Incremental:*** Incremental learning is currently not supported in XTREE. This represents a clear direction for future enhancements, but for this term project this is not explored.
(7) ***Sharable:*** XTREE can summarize large quantities of data and represent them as a simple tree in a JSON format. This json representation can easily be shared without having to share the entire raw data. This criterion has been inherently satisfied.
(8) ***Succinctness:*** Since XTREE works by constructing a decision tree and reasoning across the leaves of that decision tree, it presents a succinct representation of the data. Thus, this criteria is already satisfied by the decision tree.
(9) ***Context aware*** XTREE generates different conclusions from different regions of the tree. This enables a context aware decision making approach.

Criteria	Supported	This work	Future work
Model Readability	✓		
Actionable Conclusions		✓	
Memory Usage	✓		
Multigoal Reasoning			✓
Anomoly Detection	✓		
Incremental			✓
Sharability	✓		
Succintness	✓		
Context Aware	✓		
Transfer knowledge	✓		

Figure 1: Satisfiable Criteria

2 KEY CRITERIA

The key area of focus of this work will be to enable actionable analytics in XTREE. In it's current version, XTREE sometimes generates plans that are not actionable. For instance, in order to reduce the likelihood of defects, XTREE sometime recommends reducing lines of code by around one thousand. Although this may seem possible, it is very likely that developers are not willing to do this. This may limit the usability of tools such as XTREE.

2.1 Why this is important?

Generating valid plans is the first step towards enabling better analytics. In addition, it is only after this step will it be possible to explore multi-goal reasoning and incremental reasoning.

First, in order to reason across multiple goals, it is vital that a tool first generate valid goals. Since the current version of XTREE is limited by invalid recommendations, it cannot be used as is to reason for multiple goals.

Secondly, it is also important that valid plans be generated for creating incremental plans. The framework required for generating valid plans can be extended to choosing important data points for incremental learning. As the tree grows in size with the arrival of new data, it will be prohibitively hard to seek valid plans post hoc.

3 REVIEW OF RELATED WORK

Planning has been a subject of much research in artificial intelligence. Here, planning usually refers to generating a sequence of actions that enables an *agent* to achieve a specific *goal* [18].

This can be achieved by classical search-based problem solving approaches or logical planning agents. Such planning tasks now play a significant role in a variety of demanding applications, ranging from controlling space vehicles and robots to playing the game of bridge [8]. Some of the most common planning paradigms include: (a) classical planning [21]; (b) probabilistic planning [1]; and (c) preference-based planning [2].

In software engineering, the planning problem translates to proposing changes to software artifacts. Solving this has been undertaken via the use of some search-based software engineering techniques [10]. Examples of algorithms include SWAY, NSGA-II, etc. [6, 17].

These search-based software engineering techniques require access to some trustworthy models that can be used to explore novel solutions. In some software engineering domains there is ready access to such models which can offer assessment of newly generated plans. Examples of such domains within software engineering include automated program repair [14, 20], software product line management [11, 19], etc.

In summary, for domains with readily accessible models, we recommend the tools used by the search-based software engineering community. For domains, where domain-models are not available, we recommend tools such as XTREE.

4 CRITIQUE

Existence of a model precludes the use of each of these planning approaches. This is a limitation of all these planning approaches since, not all domains come with ready-to-use models. For example, consider software defect prediction and all the intricate issues that may lead to defects in a product. A model that includes *all* those potential issues would be very large and complex. Further, the empirical data required to validate any/all parts of that model can be hard to find.

In fact, our experience has been that accessing and/or commissioning a model can be a labor-intensive process. For example, in previous work [16] we used models developed by Boehm's group at the University of Southern California. Those models inputted project descriptors to output predictors of development effort, project risk, and defects. Some of those models took decades to develop and mature (from 1981 [3] to 2000 [4]).

Furthermore, even when there is an existing model, they can require constant maintenance lest they become out-dated. Elsewhere, we have described our extensions to the USC models to enable reasoning about agile software developments. It took many months to implement and certify those extensions [15]. The problem of model maintenance is another motivation to look for alternate methods that can be automatically updated with new data.

Fortunately, sometimes it is easier to access cases of a model's behaviour than the model itself. For example, in prior work with Martin Feather from the Jet Propulsion Laboratory [7], our research partner could not share a propriety model from within the NASA

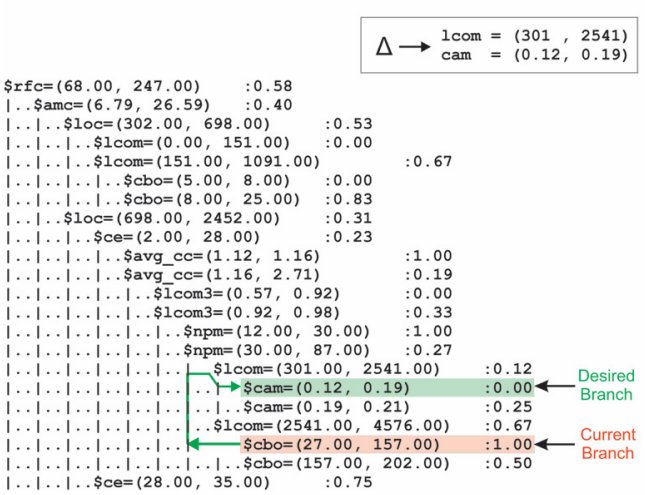

Figure 2: Generating thresholds using XTREE. The recommended changes are shown in the bounded box above.

firewalls. However, they could share logs of the input/output of that model.

That is, in domains that do not have ready-to-use models, we may seek alternate methods for planning that can be automatically updated with new data without a need for comprehensive models. For this, we propose the use of data mining approaches to create a quasi-model of the domain and make of use observable states from this data to generate an estimation of the model. Our preferred tools in this paper (XTREE) takes this approach by constructing decision trees on available data (discussed in § 5.2).

5 CURRENT TECHNOLOGY

5.1 What is planning?

We distinguish planning from prediction for software quality as follows: Quality prediction points to the likelihood of defects. Predictors take the form: $out = f(in)$, where in contains many independent features and out contains some measure of how many defects are present. For software analytics, the function f is learned via data mining (with static code attributes for instance). Contrary to this, quality planning generates a concrete set of actions that can be taken (as precautionary measures) to significantly reduce the likelihood of defects occurring in the future. For a formal definition of plans, consider a test example $Z = \{Z_1, Z_2, ..., Z_n\}$, planners proposes a plan $\forall \delta_j \in \Delta$ to adjust attribute $Z_j \in Z$ as follows:

$$\forall \delta_j \in \Delta : Z_j = \begin{cases} Z_j + \delta_j & \text{if } Z_j \text{ is numeric} \\ \delta_j & \text{otherwise} \end{cases}$$

With this, to (say) simplify a large bug-prone method, our planners might suggest to a developer to reduce its size (i.e. refactor that code by splitting it simpler functions).

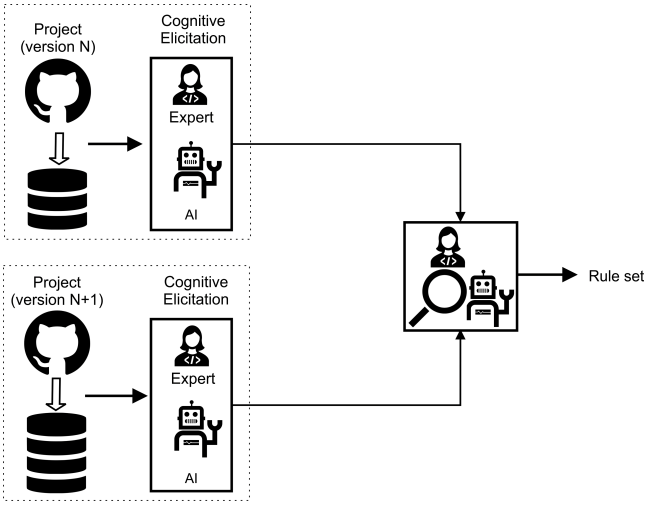

Figure 3: Framework for actionable changes.

5.2 XTREE

XTREE builds a *supervised* decision tree and then generates plans by contrasting the differences between two branches: (1) branch where you are; (2) branch where you want to be.

The specifics of the algorithm used to divide the data and construct the decision tree were presented in greater detail in our previous work [13]. Next, XTREE builds plans from the branches of the decision tree by asking the following three questions for each test case (the last of which returns the plan):

- Which *current* branch does a test instance fall in?
- Which *desired* branch would we want to move to?
- What are the *deltas* between current and desired?

As a motivating example, consider Fig. 5.2 with XTREE constructed with training data consisting of OO code metrics [5] and associated defect counts. A defective test case with the same code metrics is passed into the tree and evaluated down the tree to a leaf node with a defect probability of 1.0 (see the **orange** line in Fig. 5.2). XTREE then looks for a nearby leaf node with a lower defect probability (see the **green** line in Fig. 5.2). XTREE then evaluates the differences (of *deltas*) between **green** and **orange**. These *deltas* can be translated as plans[1]. In Fig. 5.2, these plans are to change *lcom* (lack of coupling among methods) and *cam* (cohesion among methods). For a developer, these may translated to (a) Ensure a class contains more methods that interact with each other to address *lcom*, and (b) Ensure cohesion among methods improves to address *cam*.

[1]Represented as thresholds that are denoted by $[low, high)$ ranges for each OO metric

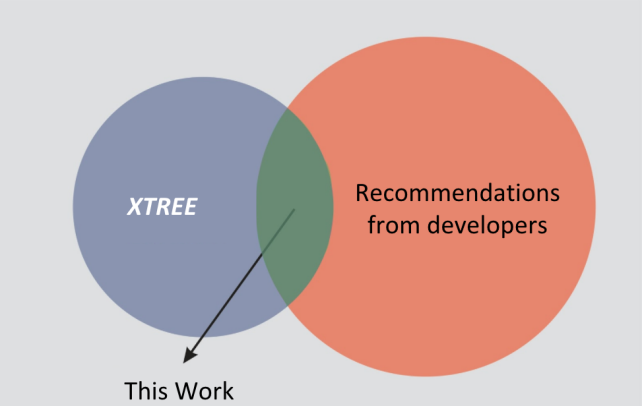

Figure 4: Identifying actionable changes.

6 EXTENDING XTREE

6.1 Why this is necessary?

There were some issues with the initial version of XTREE. First, it was limited to using data from within a project to generate plans. To address this, we developed BELLTREE which used the same framework as XTREE but uses bellwethers as the source of data for planning. Our results comparing BELLTREE with XTREE on a set of open source java projects as reported in [12] was that if within-project data from previous releases are available, we may use XTREE. If not, using bellwethers would be a reasonable alternative.

However, we noted that XTREE suffered from one major demerit. Which was that it recommended plans that were not actionable. For instance, one of the most commonly recommended change was to *reduce line of code*. Although, this is a plausible action, it is likely that in some cases, developers may not be entirely capable of making this change. Therefore, we need to minimized such recommendations.

This is very important for reasons discussed in §2.1. Briefly, generation of valid plans precludes other enhancements such as multi-goal reasoning and incremental learning. Further, it is also crucial to ensure accurate transfer of knowledge.

6.2 What needs to be done?

The proposed framework for addressing the issue of actionable conclusion is shown in Figure 3. The steps are briefly listed below:

- Gather project data and generate plans using an automated approach. In this case, XTREE.
- Next, gather expert opinions on what constitutes a doable change.
- Finally, combine the rules generated by experts and those generated by XTREE to generate a set of actionable plans.

As shown in Figure 4, we seek the shaded green region. Currently, XTREE recommends changes in the **blue**, developers recommendations are shown in **orange**. The subset of desirable changes that we seek is shown in **green**.

Consider a project $\mathbb{X}$ with versions $v = 1, 2, 3, 4, 5$. Here:

(1) First, we use versions $v = 1, 2$ to identify the most changed features.
(2) Then we construct the XTREE with version $v = 3$ and generate plans for version $v = 4$. We call these plans P_{raw}.
(3) Next, we *bias* XTREE with the change frequency information of step (1) and generate different set of plans. We call these plans P_{bias}.
(4) Finally, we look at version $v = 5$ to compare the plans generated with P_{raw} and P_{bias} and identify the ones most explored by the developers.

Figure 5: Proposed experimental procedure.

6.3 How this will be achieved?

This report endorses a four-step procedure to achieve this. These are listed below:

(1) First, we compare the differences between different versions of a project.
(2) Next, we discover the attributes that have most changed between different versions of the project.
(3) After that, we construct an XTREE that are more biased towards the most frequent attributes changed and less so towards the least frequent changes.
(4) Finally, we compare the plans obtained from (3) with plans generated from an unbiased XTREE that allows changes to everything.

Figure 5 shows the proposed experimental setup that uses the above steps.

REFERENCES

[1] Eitan Altman. 1999. *Constrained Markov decision processes*. Vol. 7. CRC Press.

[2] Shirin Sohrabi Jorge A Baier and Sheila A McIlraith. 2009. HTN planning with preferences. In *21st Int. Joint Conf. on Artificial Intelligence*. 1790–1797.

[3] B Boehm. 1981. *Software Engineering Economics*. Prentice Hall.

[4] Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K Clark, Bert Steece, A Winsor Brown, Sunita Chulani, and Chris Abts. 2000. *Software Cost Estimation with Cocomo II*. Prentice Hall.

[5] Shyam R Chidamber and Chris F Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on software engineering* 20, 6 (1994), 476–493.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T Meyarivan. 2002. A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6 (2002), 182–197.

[7] M S Feather and T Menzies. 2002. Converging on the Optimal Attainment of Requirements. In *IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany*.

[8] Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: theory and practice*. Elsevier.

[10] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. 2009. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. *Dept. Comp. Sci, King's College London, Tech. Rep. TR-09-03* (2009).

[11] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon. 2015. Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 517–528. https://doi.org/10.1109/ICSE.2015.69

[12] Rahul Krishna and Tim Menzies. 2017. Learning Effective Changes For Software Projects. *arXiv* abs/1708.05442 (2017).

[13] Rahul Krishna, Tim Menzies, and Lucas Layman. 2017. Less is more: Minimizing code reorganization using XTREE. *Information and Software Technology* (mar 2017). https://doi.org/10.1016/j.infsof.2017.03.012 arXiv:1609.03614

[14] Claire Le Goues, Neal Holtschulte, Edward K Smith, Yuriy Brun, Premkumar Devanbu, Stephanie Forrest, and Westley Weimer. 2015. The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs. *IEEE Transactions on Software Engineering* 41, 12 (dec 2015), 1236–1256. https://doi.org/10.1109/TSE.2015.2454513

[15] B Lemon, A Riesbeck, T Menzies, J Price, J D'Alessandro, R Carlsson, T Prifiti, F Peters, H Lu, and D Port. [n. d.]. Applications of Simulation and AI Search: Assessing the Relative Merits of Agile vs Traditional Software Development. In *ASE'09*.

[16] Tim Menzies, Martin S Feather, Ray Madachy, and Barry W. Boehm. 2007. The Business Case for Automated Software Engineering. In *Proc. ASE*. ACM, New York, NY, USA, 303–312. https://doi.org/10.1145/1321631.1321676

[17] Vivek Nair, Tim Menzies, and Jianfeng Chen. 2016. An (accidental) exploration of alternatives to evolutionary algorithms for sbse. In *International Symposium on Search Based Software Engineering*. Springer, 96–111.

[18] Stuart Russell and Peter Norvig. 1995. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Egnlewood Cliffs.

[19] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Scalable product line configuration: A straw to break the camel's back. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE.

[20] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. 2009. Automatically finding patches using genetic programming. In *Proceedings - International Conference on Software Engineering*. IEEE, 364–374. https://doi.org/10.1109/ICSE.2009.5070536

[21] Michael Wooldridge and Nicholas R Jennings. 1995. Intelligent agents: Theory and practice. *The knowledge engineering review* 10, 2 (1995), 115–152.