Planning Effective Changes Within and Across Software Projects

Rahul Krishna, Tim Menzies
Comptuer Science, North Carolina State University, USA
{i.m.ralk, tim.menzies}gmail.com

Abstract—The current generation of software analytics tools are mostly prediction algorithms (e.g. support vector machines, naive bayes, logistic regression, etc). While prediction is useful, after prediction comes *planning* about what actions to take in order to improve quality. This research seeks methods that support actionable analytics that offer clear guidance on "what to do" within the context of a specific software project. Specifically, we propose the BELLTREE algorithm for generating plans to improve software quality. Each such plan has the property that, if followed, it reduces the probability of future defect reports. When compared to other planning algorithms from the SE literature, we find that BELLTREE is most effective at learning plans from one project, then applying those plans to another.

Index Terms—Data mining, actionable analytics, planning, bellwethers, local models, defect prediction.

1 Introduction

Data mining tools have been applied to many applications in SE. For example, it has been used to estimate how long it would take to integrate new code into an existing project [1], where bugs are most likely to occur [2], [3], or how long it will take to develop this code [4], [5], etc. Large organizations like Microsoft routinely practice data-driven policy development where organizational policies are learned from an extensive analysis of large datasets [6], [7].

Despite these successes, there exists some drawbacks with current software analytic tools. At a recent workshop on "Actionable Analytics" at the 2015 IEEE conference on Automated Software Engineering, business users were very vocal in their complaints about analytics [8]. "Those tools tell us *what is*," said one business user, "But they don't tell us *what to do*".

Accordingly, in this research, we seek new tools that support actionable analytics that offer clear guidance on "what to do" about a specific software project. We seek new tools since the current generation of software analytics tools are mostly *prediction* algorithms such as support vector machines [9], naive Bayes classifiers [10], logistic regression [10], etc. For example, defect prediction tools report what combinations of software project features predict for some dependent variable (such as the number of defects). Note that this is a different task to *planning*, which answers the question: what to *change* in order to *improve* quality.

More specifically, we seek plans that offer *least* changes in software but most *improve* the *quality* where:

- *Quality* = defects reported by the development team;
- *Improvement* = lowered likelihood of future defects.

This paper advocates the use of the *bellwether effect* [11] to generate plans. This effect states that:

> *" . . . When a community of programmers work on a set of projects, then within that community there exists one exemplary project, called the bellwether*[1]*, which can best define quality predictors for the other projects . . . "*

Utilizing the bellwether effect, we propose a variant of our XTREE contrast set learner [12] called "BELLTREE". BELLTREE searches for an exemplar project to construct plans for the other projects. These plans are then applied to all future data generated by that community. Bellwethers and BELLTREE are useful for planning purposes since:

- These bellwethers are stable sources for planning. i.e., as long as the predictions of bellwethers remain the same, the lessons learned from it can be applied to the rest of the community.
- As shown by the experiments of this this paper, bellwethers can be found very early in a projects life cycle (after analyzing only a few dozen code modules);
- Additionally, when compared to other planning methods in software engineering literature (Shatnawi [13], Alves et al. [14] and Oliveira et al. [15]), BELLTREE produces much better plans.

In summary, the contributions of this work are as follows.

1. New kinds of software analytics techniques: This work combines planning [12] with cross-project learning using bellwethers [11]. This is a unique approach since our previous work on bellwethers [11], [16] explored prediction and not planning as described here. Also, the previous work on planning [12] explored with-project problems but not cross-project problems as explored here.

2. Compelling results about planning: Our results show that planning is quite successful in producing actions that can reduce the number of defects; e.g. our result show that our planners can reduce defects by more than 50% in 6 out of the 8 datasets studied here (>90% in the certain cases).

1. According to the Oxford English Dictionary, the bellwether is the leading sheep of a flock, with a bell on its neck.

3. More evidence of generality of bellwethers: The more the bellwether effect is explored, the more we learn about its broad applicability. Originally, we explored this just in the context of prediction [11], it has been shown to work for defect prediction, effort estimation, predicting when issues will close, and detecting code smells [16]. This paper extends those results with a new finding that bellwethers can also be used from cross-project quality planning. This is an important result of much significance since, it suggests that general conclusions about SE are easily found (with bellwethers).

1.1 Research Questions

This section provides a brief overview of the research questions this paper seeks to answer.

RQ1: How similar are the plans generated by XTREE with those undertaken by developers?

In the first research question, we seek to establish the similarity between the plans generated by XTREE and the outcome of developers' changes to projects. If the plans offered by XTREE do not match those undertaken by developers, then it becomes difficult to justify the effectiveness of XTREE in generating actionable plans.

Result: In all the projects studied here, there is a significant overlap between plans recommended by XTREE and those undertaken by developers.

RQ2: How effective is within-project planning with XTREE?

In this research question, we assess the effectiveness of XTREE when it is trained on past data from *within* a project. This research question follows RQ1, where we showed that changes recommended by XTREE has a significant overlap with changes actually undertaken by developers. XTREE recommends changes so that the project might have have fewer defects in the next version, however correlation does not necessarily imply causation, it is theoretically possible that the recommendations generated by XTREE could be misleading.

The results of this paper are in accordance with our previous findings [12] and show that *XTREE* is an effective planner for automatically recommending useful plans.

Result: For within-project planning, we see that greater the overlap with XTREE's recommendations the larger the number of defects reduced in a subsequent release of a project. This was true in 8 out of 9 projects studied here.

RQ3: How does XTREE compare with other threshold based methods for within-project planning?

Alternative methods for planning make use of statistically determined thresholds over source code metrics for reducing defects. Recent work by Shatnawi [13], Alves et al. [14], and Oliveira et al. [15] assume that unusually large measurements in source code metrics point to larger likelihood of defects and these should be avoided since, if left unchanged, they would lead to defect-prone code.

Our results show that their assumption is not effective to generate actionable plans and that XTREE is a better way to plan quality improvement.

Result: For within-project planning, plans generated by XTREE is significantly superior to plans generated by other threshold based methods.

RQ4: How effective is cross-project planning with XTREE?

When projects do not have sufficient within project data (perhaps due to the project being relatively new), we may use data from other similar projects to generate actionable plans for the original project. We achieve this using the *bellwether effect* [12]. We first discover a *bellwether* dataset from the available projects, and then we train XTREE on the bellwether data. In this research question, we assess the effectiveness of plans when using the *bellwether* data.

As with RQ2, we see that in a cross-project setting, plans generated using XTREE and the bellwether data is quite effective in generating actionable plans.

Result: For cross-project planning, we see that larger the overlap with XTREE's recommendations leads to larger reduction in the number of defects for subsequent release of a project. This was true in all 9 projects that were studied here.

RQ5: Are cross-project plans generated by XTREE comparable to within-project plans?

In RQ4, we established that it is possible to generate effective plans for projects by a training a planner (XTREE) on a *bellwether* data. In this research question, we compare plans generated using within-project data to those generated using cross-project data (the bellwether dataset).

Our experiments show that the effectiveness of plans generated from within-project data and XTREE is statistically comparable to plans derived cross-project data using the bellwether dataset. And since the source of data used for cross-project planning is the bellwether, it remains unchanged for relative longer stretches to time [12]. This means that the plans would also remain consistent for long stretches of time. This result is particularly encouraging since it demonstrates a level of conclusion stability required to maintain user confidence in analytics.

Result: Cross-project planning with XTREE is statistically comparable to within-project planning.

RQ6: Are BELLTREE cross-project plans any better than XTREEs within project plans?

In the final research question, we compare plans of XTREE and alternative threshold based methods of Shatnawi [13], Alves et al. [14], and Oliveira et al. [15] for cross-project planning with the bellwether data.

As with RQ4, our results show that threshold based methods not effective in generating actionable plans for cross-project planning. XTREE is a significantly superior approach to cross-project planning.

Result: For cross-project planning, plans generated by XTREE is significantly better than to plans generated by other threshold based methods.

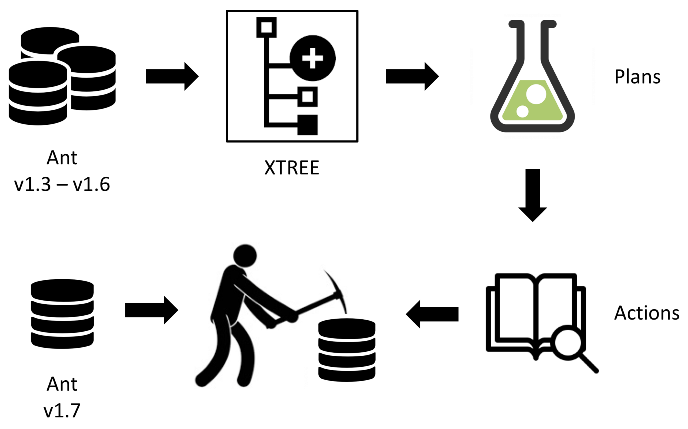


Fig. 1: A proposed planning framework.

2 Motivation

2.1 What exactly 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 (such as OO metrics) and *out* contains some measure of how many defects are present. For software analytics, the function f is learned via data mining (for static code attributes for instance).

On the other hand, 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 defective test example Z, a planner proposes a plan D to adjust attribute Z_j 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 planner, 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, say, splitting it across two simpler functions).

2.2 Example

The proposed framework for the use of XTREE in a real world situation is illustrated in Figure 1. There we see that given within-project data[2] in the form of data from previous releases of a sample project (releases v1.3 — v1.6), we first construct a planner (XTREE). Then using the test data (Ant v1.7), we obtain plans for what changes can be made to improve the quality of the software. Next, we look for specific actions that can be taken (for some example actions, see Figure 2(b)). When a plan recommended by XTREE matches possible action, the developers may choose to use the plan as a guide to undertaking preventive or perfective actions to improve the quality of that software.

2. This within-project data can easily be replaced by the bellwether dataset if necessary.

DIT	NOC	CBO	RFC	FOUT	WMC	NOM	LOC	LCOM
·	·	·	+	·	+	+	+	+

(a) Plans recommended by XTREE

Action	DIT	NOC	CBO	RFC	FOUT	WMC	NOM	LOC	LCOM
Extract Class			+	−	+	−	−	−	−
Extract Method				+		+	+	+	+
Hide Method									
Inline Method				−		−	−	−	−
Inline Temp								−	
Remove Setting Method				−		−	−	−	−
Replace Assignment								−	
Replace Magic Number								+	
Consolidate Conditional				+		+	+	−	+
Reverse Conditional									
Encapsulate Field						+	+	+	+
Inline Class			−	+	−	+	+	+	+

(b) A sample of possible actions developers can take. The action highlighted in green indicates the action that matches the recommendation offered by XTREE with respect the available metrics.

```
class StoreManagement{
  ... // Some Operations
  private static void showMenu() { ... }
  public static void showActions() {
    int choice;
    do {
      showMenu();
      switch(choice) {
        case 1: displayAllInventory();
        case 2: findCompanyName();
        case 3: modifyPrice();
        case 4: saveToDisk();
                update();
                System.out.println("Done!");
                System.exit(0);
      }
    } while(choice != 4)
  }
  .... // Other Methods
}
```

(c) A contrived example of a inventory management task which is possibly defective.

```
class StoreManagement{
  ... // Some Operations
  private static void showMenu() { ... }
  public static void showActions() {
    int choice;
    do {
      showMenu();
      switch(choice) {
        case 1: displayAllInventory();
        case 2: findCompanyName();
        case 3: modifyPrice();
        case 4: exit();
      }
    } while(choice != 4)
  }
  // Extract method to here ..
  public static void exit(){
    saveToDisk();
    update();
    System.out.println("Done!");
    System.exit(0);
  }
  .... // Other Methods
}
```

(d) The example in 2(c) after applying "extract method" operation as per XTREE's recommendations.

Fig. 2: An example of how developers might use XTREE to reduce software defects.

For a more concrete example, consider Figure 2. Here, we apply the framework proposed in Figure 1 and obtain a set of recommendations as shown in 2(a). Then, we use 2(b) to look-up possible actions developers may take, there we see that performing an "extract method" operation may help alleviate certain defects (this is highlighted in green).

In 2(c) we show a simple contrived example of a class where the above operation may be performed. In 2(d), we demostrate how a developer may go about performing the "extract method" operation.

2.3 Planning in Classical AI

In classical AI, planning usually refers to generating a sequence of actions that enables an *agent* to achieve a specific *goal* [26]. In an idealized situation, it is assumed that the possible initial states of the world is known and so are the description of the desired goal in addition to the set of possible and feasible actions. In such a hypothetical situation, planning results in generating a set of actions that is guaranteed to enable one to reach a desired goal. 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 [27].

As discussed in the rest this section, there are many types of planning. We introduce the premise of those different planning paradigms, then dedicate the rest of the section to discuss its implications for planning in software engineering.

2.3.1 Classical Planning

A simple abstraction of the planning problem is known as classical planning. Classical planning assumes that the initial state is fully-observable and any action is deterministic. Such an assumption enables us to predict the outcome of an action accurately every single time. Further, plans can be defined as sequences of actions, because it is always known in advance which actions will be needed [28].

Classical Planning is most useful when the problem is not very complex and the simplifying assumptions mentioned above lead to a development of a well-founded model [29]. In the case of complex domains like software engineering, performing a search of plans is highly inefficient. It becomes very difficult to pick the correct search space, algorithm, and heuristics for finding these plans [27].

2.3.2 Probabilistic Planning

Probabilistic planning tackles a slightly different problem in planning — one where the outcomes may be random and are partly controlled by the decision maker [30], [31], [32]. These kinds of planning problems are usually solved using dynamic programming and reinforcement learning. The key idea here is to represent the planning problem as an optimization problem [27]. Planning for such problems are best achieved when the state-space is small [27]. Additionally, such a planning approach works even with partial observability, i.e., it works even when the planning agent cannot fully observe the underlying problem space. Planning over partially observable states is achieved by maintaining a distribution of probabilities over the possible states and planning based on these distributions [33].

2.3.3 Preference-based Planning

The preference based planning is an extension of the above planning schemes with a focus on producing plans that satisfy as many user-defined constraints (preferences) as possible [34]. These preferences as defined by the user are generally not hard constrains, rather they define the quality of the plan, which increases as more of the preferences are satisfied. Algorithms that can solve constraint satisfaction problems are well suited to solve these forms of planning problems. Other popular algorithms include PPLAN [34] and HTN [35]. However, existence of a model precludes the use of this planning approach. This is a major impediment, since for reasons discussed in the following section, not every domain has a reliable model.

2.4 Planning in Software Engineering

We know of at least two kinds of planning research in SE. Each kind is distinguishable by *what* is being changed.

In *test-based planning*, some optimization is applied to reduce the number of tests required to achieve to a certain goal or the time taken before tests yield interesting results [36], [37], [38].

In *process-based planning* some search-based optimizer is applied to a software process model to infer high-level business plans about software projects. Examples of that kind of work include our own prior studies combining simulated annealing with the COCOMO models or Ruhe et al.'s work on next release planning in requirements engineering [39], [40]. This paper is about *code-based planning* where the goal is to change a code base in order to improve that code in some way.

In software engineering, the planning problem translates to proposing changes to software artifacts. These are usually a hybrid task combining elements of probabilistic planning (see §2.3.2) and preference based plans (see §2.3.3). Solving this has been undertaken via the use of some search-based software engineering techniques [41], [42]. These search-based techniques are evolutionary algorithms that propose actions guided by a fitness function derived from a well established domain model. They work by: (a) generating an initial population of solutions using an initialization policy (for example they may generate 100s to 100000s instances via random sampling); (b) evaluating each solution using a problem specific fitness function, (c) creating new solutions via mutation and crossover operation of existing samples, and (d) reassessing these solutions using the fitness functions. Examples of algorithms include GALE, NSGA-II, NSGA-III, SPEA2, IBEA, MOEA/D, etc. [43], [44], [45], [46], [47], [48], [49]. These planning tools all require access to some trustworthy models that can be used to explore some highly novel examples. 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 [50], [51], [52], [53], [54], software product line management [55], [56], [57], [58], automated test generation [59], [60], [61], etc.

amc	average method complexity	e.g. number of JAVA byte codes
avg_cc	average McCabe	average McCabe's cyclomatic complexity seen in class
ca	afferent couplings	how many other classes use the specific class.
class.		
cam	cohesion amongst classes	summation of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods.
cbm	coupling between methods	total number of new/redefined methods to which all the inherited methods are coupled
cbo	coupling between objects	increased when the methods of one class access services of another.
ce	efferent couplings	how many other classes is used by the specific class.
dam	data access	ratio of the number of private (protected) attributes to the total number of attributes
dit	depth of inheritance tree	
ic	inheritance coupling	number of parent classes to which a given class is coupled (includes counts of methods and variables inherited)
lcom	lack of cohesion in methods	number of pairs of methods that do not share a reference to an case variable.
locm3	another lack of cohesion measure	if m, a are the number of $methods, attributes$ in a class number and $\mu(a)$ is the number of methods accessing an attribute, then $lcom3 = ((\frac{1}{a}\sum, j^a \mu(a,j)) - m)/(1-m)$.
loc	lines of code	
max_cc	maximum McCabe	maximum McCabe's cyclomatic complexity seen in class
mfa	functional abstraction	number of methods inherited by a class plus number of methods accessible by member methods of the class
moa	aggregation	count of the number of data declarations (class fields) whose types are user defined classes
noc	number of children	
npm	number of public methods	
rfc	response for a class	number of methods invoked in response to a message to the object.
wmc	weighted methods per class	
nDefects	raw defect counts	Numeric: number of defects found in post-release bug-tracking systems.
isDefective	defects present?	Boolean: if $nDefects > 0$ then *true* else *false*

Fig. 3: OO code metrics used for all studies in this paper. Last lines, shown in gray, denote the dependent variables.

Community	Dataset	# of instances			Description	# metrics	Granularity
		Versions	# samples	Bugs (%)			
Jureczko	Lucene	2.0, 2.2,2.4	782	438 (56.01)		20	Class
	Ant	1.3, 1.4, 1.5, 1.6, 1.7	1692	350 (20.69)			
	Ivy	1.1, 1.4,2.0	704	119 (16.90)			
	Jedit	3.2, 4.0, 4.1, 4.2,4.3	1749	303 (17.32)			
	Poi	1.5, 2, 2.5, 3.0	1378	707 (51.31)			
	Camel	1.0, 1.2, 1.4,1.6	2784	562 (20.19)			
	Log4j	1.0, 1.1,1.2	449	260 (57.91)			
	Velocity	1.4, 1.5,1.6	639	367 (57.43)			
	Xalan	2.4, 2.5, 2.6,2.7	3320	1806 (54.40)			
	Xerces	1.0, 1.2, 1.3,1.4	1643	654 (39.81)			

Fig. 4: The figure lists defect datasets used in this paper. The bellwether dataset from each community is highlighted in light gray.

However, 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 [62] 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 [63] to 2000 [64]).

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 [65], [66]. 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 [67], our research partner could not share a propriety model from within the NASA firewalls. However, they could share logs of the input/output of that model.

In summary, for domains with readily accessible models, we recommend the kinds of tools that are widely used in the search-based software engineering community such as GALE, NSGA-II, NSGA-III, SPEA2, IBEA, particle swarm optimization, MOEA/D, etc. In several other cases where this is not an option, 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 and BELLTREE take this approach and as presented elsewhere in this paper, these methodologies have very encouraging results.

3 Research Methods

3.1 Datasets

The defect dataset used in this study comprises a total of 49 datasets grouped into 3 communities taken from previous transfer learning studies. The projects measure defects at various levels of granularity ranging from class-level to file-level. All the datasets are described in the attributes

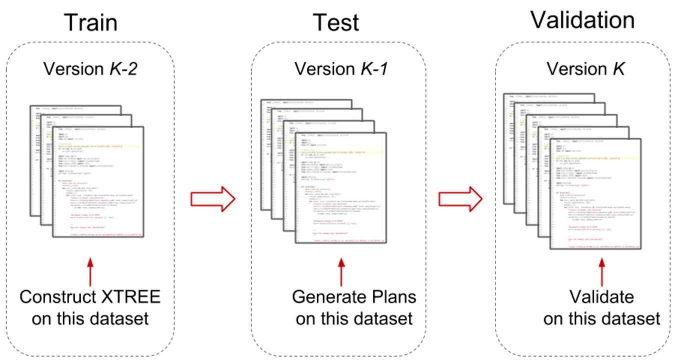


Fig. 5: K-Test to validate plans.

of Figure 3. Figure 4 summarizes all the communities of datasets used in our experiments.

Since we attempt to transfer plans from within the project, we explore homogeneous transfer of plans using the attributes shared by a community. We explored another community of dataset from the SEACRAFT repository[3]. Thisgroup of data contains defect measures from several Apache projects. It was gathered by Jureczko et al. [74]. This dataset contains records the number of known defects for each class using a post-release bug tracking system. The classes are described in terms of 20 OO metrics, including CK metrics and McCabes complexity metrics. Each dataset in the Apache community has several versions totalling 38 different datasets. For more information on this dataset see [11].

3.2 A Strategy for Evaluating Planners

In §4.1 we describe the planners and a way to translate these plans into actionable decisions. To assess the effect of acting on these plans, we propose a *verification oracle*. This is necessary because it can be rather difficult to judge the effects of applying the plan. They cannot be assessed just by a rerun of the test suite for three reasons: (1) The defects were recorded by a post release bug tracking system. It is entirely possible it escaped detection by the existing test suite; (2) Rewriting test cases to enable coverage of all possible scenarios presents a significant challenge; and (3) It make take a significant amount of effort to write new test cases that identify these changes as they are made.

To resolve this problem, SE researchers such as Cheng et al. [75], O'Keefe et al. [76], [77], Moghadam [78] and Mkaouer et al. [79] use a *verification oracle* that is learned separately from the primary oracle. The verification oracles assesses how defective the code is before and after some code changes. For their verification oracle, Cheng, O'Keefe, Moghadam and Mkaouer et al. use the QMOOD hierarchical quality model [80]. A shortcoming of QMOOD is that quality models learned from other projects may perform poorly when applied to new projects [20].

Hence, for this study, we eschew older quality models like QMOOD. Instead, we propose a more rigorous validation criteria, referred to hence forth as the *k-test*.

3. https://zenodo.org/communities/seacraft/

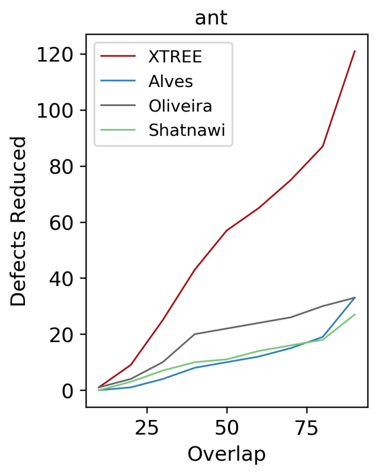
3.2.1 *The k-test*

Given a project $\mathcal{P}$ with versions $v \in \{\mathcal{P}_i, \mathcal{P}_j, \mathcal{P}_k\}$ ordered chronologically (that is, version $i < j < k$ in terms of release dates), we divide the project data into three sets: (1) the *train set*; (2) the *test set*; and (3) the *validation set*. The *k-test* works as follows:

1) First, train the planner on version $\mathcal{P}_i$. Note: this could either be data that is either a previous release (version i), or it could be data from the bellwether dataset.
2) Next, use the planner to generate plans to reduce defects for version $\mathcal{P}_j$.
3) Finally, on version $\mathcal{P}_k$, we measure the OO metrics for each class in $\mathcal{P}_j$, then we (a) measure the overlap between plans recommended by XTREE and the actions undertaken by the developers; (b) count the number of defects reduced/increased when compared to the previous release.

As the outcome of the *k-test* we obtain the number of defects (increased or decreased) and the extent of overlap (from 0% to 100%). These enable us to plot the operating characteristic curve for the planners. The operating characteristic (OC) curve depicts the effectiveness of a planner with respect to the ability to reduce defects. The OC curve plots the overlap of developer changes with XTREE's recommendations versus the number of defects reduced (referred to henceforth as planner effectiveness curve). A sample curve for one of our datasets in Figure 4 is shown in Figure 6.

For each of the datasets with versions i, j, k we (1) train the planner on version i; (2) deploy the planner to recommend plans for version j; and (3) validate plans for version j. Following this, we plot the planner effectiveness curve. Finally, we compute the area under the planner effectiveness



Calculation Area Under Planner Effectiveness Curve (AUPEC):

Fig. 6: Sample report format for Scott-Knott (with contrived values) showing *Treatment A* succeeding with rank 1.

curve (AUPEC) using the trapizoidal rule. See Figure 6

$$R = (1 - \frac{after}{before}) \times 100\% \tag{1}$$

The value of the measure R has the following properties:

- If $R = 0\%$, this means "no change from baseline";
- If $R > 0\%$, this indicates "improvement over the baseline";
- If $R < 0\%$, this indicates "optimization failure".

In addition to this, we count the frequency with which a certain code metric is changed. This is expressed as percentage change, measured using the following equation:

$$Change = \frac{\#times\ M\ changed}{\#test\ cases} \times 100\% \quad \forall M \in OO\ Metrics \tag{2}$$

4 Planning in Software Analytics

4.1 Implementing Planners

In the previous sections, we introduced the notion of planning for decision making. In this section we offer a description of each of these planning methods:

4.1.1 XTREE

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

XTREE takes a *supervised* $Cluster + Contrast$ approach to planning because we hypothesize that it is useful to reflect on the target class. Thus, XTREE uses a supervised decision tree algorithm of Figure **??**.A. To divide continuous numeric data, we use a Fayyad-Irani discretizer of Figure **??**.B. Next, XTREE builds plans from the branches of the decision trees using the description of Figure **??**.C. In doing so, we ask three questions, the last of which returns the plan:

1) Which *current* branch does a test case fall in?
2) Which *desired* branch would the test case want to move to?
3) What are the *deltas* between current and desired branches?

These *deltas* represent the threshold ranges[4] that represent the plans to reduce the defects.

4.1.2 BELLTREE

BELLTREE is structurally similar to XTREE. It differs in the source of data used for analytics. While XTREE uses data from within the project, BELLTREE first starts by looking for the bellwether dataset. After finding the bellwether, BELLTREE constructs a decision tree to generate the plans.

To achieve this, we employ three operators – DISCOVER, PLAN, VALIDATE:

1) DISCOVER: *Check if the community has bellwether.* This step is similar to our previous technique used to discover bellwethers [11]. That is, we see if standard data miners can predict for the number of issues, given the static code attributes. This is done as follows:
   - For a community C obtain all pairs of data from projects such that $P_i, P_j \in C$;
   - Predict for defects in P_j using a quality predictor learned from data taken from P_i;
   - Report a bellwether if one P_i generates consistently high predictions in a majority of $P_j \in C$.
2) PLAN: *Using the bellwether, generate plans to improve a new project data.* That is, having learned the bellwether on past data, we now construct a decision tree similar to XTREE. We use the same methodology to generate these plans.
3) VALIDATE: *Go back to step 1 if the performance statistics seen during PLAN fail to generate useful actions.*

4. Thresholds are denoted by $[low, high)$ ranges for each metric

Note the simplicity of this approach – just wrap a for-loop around some data miners to discover the bellwether. Given the ubiquity of bellwethers [16], this stage identifies bellwethers for each of the dataset studied here. These bellwethers are shaded in light gray in Figure 4.

Further, note that the use of bellwethers enables BELLTREE to leverage data from across different projects within a community. This presents a novel extension to XTREE which is limited to the used of data only from within a project.

4.1.3 Alves

Alves et al. [14] proposed an unsupervised approach that uses the underlying statistical distribution and scale of the OO metrics. It works by first weighting each metric value according to the source lines of code (SLOC) of the class it belongs to. All the weighted metrics are then normalized by the sum of all weights for the system. The normalized metric values are ordered in an ascending fashion (this is equivalent to computing a density function, in which the x-axis represents the weight ratio (0-100%), and the y-axis the metric scale).

Alves et al. then select a percentage value (they suggest 70%) which represents the "normal" values for metrics. The metric threshold, then, is the metric value for which 70% of the classes fall below. The intuition is that the worst code has outliers beyond 70% of the normal code measurements i.e., they state that the risk of there existing a defect is moderate to high when the threshold value of 70% is exceeded.

Here, we explore the correlation between the code metrics and the defect counts with a univariate logistic regression and reject code metrics that are poor predictors of defects (i.e. those with $p > 0.05$). For the remaining metrics, we obtain the threshold ranges which are denoted by $[0, 70\%)$ ranges for each metric. The plans would then involve reducing these metric range to lie within the thresholds discovered above.

4.1.4 Shatnawi

Shatnawi [13] offers a different alternative Alves et al by using VARL (Value of Acceptable Risk Level). This method was initially proposed by Bender [91] for his epidemiology studies. This approach uses two constants (p_0 and p_1) to compute the thresholds which Shatnawi recommends to be set to $p_0 = p_1 = 0.05$. Then using a univariate binary logistic regression three coefficients are learned: α the intercept constant; β the coefficient for maximizing log-likelihood; and p_0 to measure how well this model predicts for defects. (Note:

the univariate logistic regression was conducted comparing metrics to defect counts. Any code metric with $p > 0.05$ is ignored as being a poor defect predictor.)

Thresholds are learned from the surviving metrics using the risk equation proposed by Bender:

$$Defective\ if\ Metric > VARL$$

Where,

$$VARL = p^{-1}(p_0) = \frac{1}{\beta}\left(\log\left(\frac{p_1}{1-p_1}\right) - \alpha\right)$$

In a similar fashion to Alves et al., we deduce the threshold ranges as $[0, VARL)$ for each selected metric. The plans would again involve reducing these metric range to lie within the thresholds discovered above.

4.1.5 Oliveira

Oliveira et al. in their 2014 paper offer yet another alternative to absolute threshold methods discussed above [15]. Their method is still unsupervised, but they propose complementing the threshold by a second piece of information called the *relative threshold*. This measure denotes the percentage of entities the upper limit should be applied to. These have the following format:

$$p\%\ of\ the\ entities\ must\ have\ M \leq k$$

Here, M is an OO metric, k is the upper limit of the metric value, and p (expressed as %) is the minimum percentage of entities are required to follow this upper limit. As an example Oliveira et al. state, " . . . 85% of the methods should have $CC \leq 14$. Essentially, this threshold expresses that high-risk methods may impact the quality of a system when they represent more than 15% of the whole population of methods . . ."

The procedure attempts derive these values of (p, k) for each metric M. They define a function `ComplianceRate(p,k)` that returns the percentage of system that follows the rule defined by the relative threshold pair (p, k). They then define two penalty functions: (1) `penalty1(p,k)` that penalizes if the compliance rate is less than a constant $Min\%$, and (2) `penalty2(k)` to define the distance between k and the median of preset $Tail$-th percentile. (Note: according to Oliveira et al., median of the tail is an idealized upper value for the metric, i.e., a value representing classes that, although present in most systems, have very high values of M). They then compute the total penalty as `penalty = penalty1(p,k) + penalty2(k)`. Finally, the relative threshold is identified as the pair of values (p, k) that has the lowest total `penalty`. After obtaining the (p, k) for each OO metric. As in the above two methods, the plan would involve ensuring the for every metric M $p\%$ of the entities have a value that lies between $(0, k]$[5].

5 Experimental Results

RQ1: Does within project planning offer statistically significant results?

To answer this question, we first identify the sources of data: i.e., data local to a specific project. Next, with these these sources of data we construct models for planning using the methods described in §4.1.

		Metrics			Class	
Version	Module Name	wmc	dit	...	nDefects	isDefective
1.3	taskdefs.ExecuteOn	11	4	...	0	FALSE
1.3	DefaultLogger	14	1	...	2	TRUE
...	...	...	...	...	0	FALSE
1.3	taskdefs.Cvs	12	3	...	0	FALSE
1.3	taskdefs.Copyfile	6	3	...	1	TRUE

Fig. 7: A sample of ant 1.3

Next, using a separate data, that was not used in planning, we construct a verification oracle to study the *impact* of these plans. If planning were to have a significant response to defect reduction, we would ideally like to have $R > 0\%$. In a result consistent with previous reports by Krishna et al. [12], we find that all methods have a positive impact on defect reduction.

Figure **??** shows the results of planning with various planners discussed in §4.1. These figure tabulate the percent improvements in defects defined by Equation 1. We note that, all the planners result in improvements greater than 0% which means that all these planners have a positive impact on reducing the number of defects.

We note that some planners were better than others in reducing defect counts. This is discussed in RQ2. To summarize, for a post-hoc planning, both supervised planners like XTREE and unsupervised methods of Alves, Shatnawi, Oliveria are capable of offering positive and statistically significant improvements. Therefore, we answer **RQ1** as follows:

RQ2: Does using XTREE for within project planning offer benefits over other unsupervised methods?

To answer this question, we first identify the sources of data: i.e., data local to a specific project, as in RQ1. Next, with these sources of data we construct models for planning using the methods described in §4.1. Using a verification oracle, we compare the Scott-Knott results of each of the supervised and unsupervised planners.

Figure **??** shows that in 6 of 8 test datasets, XTREE offers statistically better performance compared to other unsupervised methods. In the Jureczko community of datasets, we note that learning plans from XTREE data has much lower IQR than other planners. We attribute this to the unsupervised nature of the other planners. Specifically, these unsupervised planners use thresholds on metrics to indicate what needs to be changed. These thresholds always remain the same for a dataset and do not change with the test cases. As a result, large variances result from applying them. This, is also what sets XTREE apart from these planners. XTREE adapts the plans based on the test cases thereby resulting in better performance and lower variances. Hence:

RQ3: How stable are the plans generated by XTREE as we move across different projects?

This question naturally arises from the outcomes of the previous research questions. There we noted that plans generated using XTREE produced significantly better improvements compared to other planners. Here, we ask if

5. If certain metrics already satisfy this criterion, they remain unchanged

Project	Group	XTREE	Alves	Shatnawi	Olivera
Ant	ant-1	0.6	0.41	0.41	0.19
	ant-2	0.42	0.35	0.35	0.07
	ant-3	0.53	0.33	0.33	0.21
Camel	camel-1	0.43	0.24	0.24	0.19
	camel-2	0.36	0.27	0.27	0.08
Ivy	ivy-1	0.29	0.24	0.24	0.05
Jedit	jedit-1	0.42	0.26	0.26	0.16
	jedit-2	0.59	0.35	0.35	0.24
	jedit-3	0.73	0.51	0.51	0.22
Log4j	log4j-1	0.14	0.02	0.02	0.11
Lucene	lucene-1	0.56	0.08	0.08	0.48
Poi	poi-1	0.22	0.18	0.18	0.04
	poi-2	0.63	0.04	0.04	0.58
Velocity	velocity-1	0.43	0.07	0.07	0.34
Xalan	xalan-1	0.3	0.1	0.1	0.19
	xalan-2	0.53	0.01	0.01	0.43
Xerces	xerces-1	0.23	0.17	0.17	0.06
	xerces-2	0.24	0.18	0.18	0.06

(a) Plans recommended by XTREE

Project	Group	XTREE	Alves	Shatnawi	Olivera
Ant	ant-1	0.09	0.09	0.09	0
	ant-2	0.21	0.16	0.16	0.05
	ant-3	0.24	0.15	0.15	0.09
Camel	camel-1	0.04	0.01	0.01	0.03
	camel-2	0.07	0.04	0.04	0.03
Ivy	ivy-1	0.09	0.08	0.08	0.01
Jedit	jedit-1	0.13	0.07	0.07	0.06
	jedit-2	0.05	0.03	0.03	0.02
	jedit-3	0.01	0.01	0.01	0
Log4j	log4j-1	0.41	0.22	0.22	0.17
Lucene	lucene-1	0.41	0.17	0.17	0.24
Poi	poi-1	0.52	0.45	0.45	0.08
	poi-2	0.14	0.06	0.06	0.08
Velocity	velocity-1	0.04	0.01	0.01	0.03
Xalan	xalan-1	0.15	0.08	0.08	0.06
	xalan-2	0.56	0.36	0.36	0.2
Xerces	xerces-1	0.06	0.04	0.04	0.02
	xerces-2	0.42	0.31	0.31	0.11

(b) Plans recommended by XTREE

Fig. 8: This figure shows forecasts for future bugs and enhancements using ARIMA models constructed using past issue reports. For both proprietary and opensource projects, the magnitude of average error is very low (close to zero in several cases). Trend graphs show that an increase in actual bugs (or enhancements) leads to a corresponding increase in forecasts.

	Ant					Jedit				
	Cluster + Contrast		Statistical Thresholding			*Cluster + Contrast*		Statistical Thresholding		
	BELLTREE	XTREE	Oliveira	Alves	Shatnawi	BELLTREE	XTREE	Oliveira	Alves	Shatnawi
rfc	100	100	42	62	50	100	100	50	45	100
lcom	·	·	32	70	100	·	·	40	54	100
cam	·	46	·	23	100	·	·	·	63	100
ce	·	·	17	47	100	·	·	40	54	100
mfa	·	46	8	40	100	·	·	27	45	100
amc	·	46	4	17	100	·	50	·	·	100
dit	·	·	23	41	100	·	·	40	45	100
cbo	50	50	9	14	43	·	50	27	54	40
ca	·	·	4	13	100	·	50	18	50	100
max cc	·	·	23	46	100	·	·	18	59	100
avg cc	·	·	14	56	100	·	·	13	31	100
npm	·	·	24	38	53	·	·	31	36	50
loc	·	46	22	43	10	·	·	22	54	13
ic	·	·	9	18	100	·	·	13	18	100
wmc	·	·	30	43	61	·	·	36	40	68
dam	·	·	·	·	100	·	·	·	18	50
noc	·	·	5	15	100	·	·	·	·	100
moa	·	·	8	35	51	·	·	36	45	100
cbm	·	·	16	12	50	·	·	13	18	50
lcom3	·	3	·	21	·	·	·	·	63	·

	Ivy					Poi				
	Cluster + Contrast		Statistical Thresholding			*Cluster + Contrast*		Statistical Thresholding		
	BELLTREE	XTREE	Oliveira	Alves	Shatnawi	BELLTREE	XTREE	Oliveira	Alves	Shatnawi
rfc	·	53	46	60	50	100	50	8	14	100
lcom	·	·	35	76	100	·	·	15	69	100
cam	100	·	·	52	100	·	·	·	64	100
ce	1	53	27	31	100	·	·	5	42	100
mfa	32	·	·	20	100	·	·	2	50	100
amc	22	50	11	3	100	·	50	3	8	100
dit	·	·	16	17	100	·	·	·	50	100
cbo	·	46	25	53	48	·	99	4	5	13
ca	1	50	11	18	100	·	·	3	3	100
max cc	·	·	13	35	100	·	·	5	14	100
avg cc	50	·	6	27	100	·	·	4	8	100
npm	27	·	41	52	50	·	49	25	24	100
loc	51	96	27	48	18	99	·	5	17	3
ic	·	·	2	18	100	·	·	·	34	100
wmc	·	·	35	50	50	·	·	17	23	50
dam	·	·	·	31	100	·	·	·	47	100
noc	·	·	3	5	100	·	·	3	6	100
moa	·	·	11	30	57	·	·	6	17	·
cbm	·	·	6	20	50	·	·	49	55	50
lcom3	27	·	·	66	·	·	·	·	45	·

Fig. 9: Results for **RQ5**. Percentage counts of how often each planner recommends changing a code metric (in 40 runs). "100" means that this code metric was always recommended. Cells marked with "." indicate 0%. For the Shatnawi, Alves et al., and Oliveira et al. columns, metrics score 0% if they always fail the p ≤ 0.05 test proposed by shatnawi in §4.1. Note that textitCluster+Contrast mentions specific code metrics far fewer times than other methods.

that is due to XTREE recommending a consistent set of plans for each dataset. To answer this, we aggregate the recommended plans for each dataset that used XTREE.

Figure 9 shows the frequency with which the source code metrics are recommended for change by the planners under study. The frequency of changes were measured using Equation 2. We make the following observations with regards to XTREE with this:

1) XTREE propose changes to much fewer code metrics compared to the other unsupervised approaches. At the same time, as per Figure **??**, XTREE produce the overall best performance.
2) More interestingly, XTREE recommends different plans for different datasets. Only certain metrics are changed in all the datasets, E.g., rfc and amc (in Jureczko datasets);

From the above, we note that it is not necessary to change multiple metrics to improve defects. Using different sources of data to train a planner, a similar result, that of reducing defects, can be achieved by changing different subsets of metrics. There is no single metric, that can be changed in all datasets to achieve consistently good performance. This is disconcerting, it is highly likely that these plans will change entirely when new training data arrives, or as dataset drifts

Project	Dataset	Within-Project	Cross-Project
Ant	ant-1	0.15	0.15
	ant-2	0.57	0.57
	ant-3	0.78	0.79
Camel	camel-1	0.28	0.29
	camel-2	0.48	0.46
Ivy	ivy-1	0.09	0.09
Jedit	jedit-1	0.38	0.37
	jedit-2	0.15	0.14
	jedit-3	0.03	0.03
Log4j	log4j-1	0.4	0.4
Poi	poi-1	0.22	0.23
	poi-2	0.49	0.49
Velocity	velocity-1	0.09	0.09
Xalan	xalan-1	0.48	0.48
	xalan-2	0.29	0.27
Xerces	xerces-1	0.26	0.26
	xerces-2	0.28	0.28

Fig. 10: This figure shows forecasts for future bugs and enhancements using ARIMA models constructed using past issue reports. For both proprietary and opensource projects, the magnitude of average error is very low (close to zero in several cases). Trend graphs show that an increase in actual bugs (or enhancements) leads to a corresponding increase in forecasts.

over time. This leads to unstable conclusions.

Additionally, on further inspection of Figure 9, it can be noticed that unsupervised planners (Shatnawi, Alves, and Oliveira) recommend changing almost all of the code metrics. However, this is not a practical alternative, and may render much of these suggestions useless. Note that these unsupervised methods also suffer from "conjunctive fallacy". That is, they assume that the best way to reduce defects is to decrease multiple code metrics to lie below a threshold while not accounting for the intricate dependencies between the code metrics. We therefore deprecate these unsupervised planners in favor of supervised planners like XTREE. In conclusion, we answer **RQ3** as follows:

RQ4: Does cross-project planning with BELLTREE offer statistically significant improvements compared to other methods?

In this research question, we compare the use of BELLTREE planner with other unsupervised planners. To answer this research question, we conduct experiments similar to RQ2. That is we, train the planners on bellwether data and generate plans for reducing the number of defects.

We then use Scott-Knott to statistically compare BELLTREE with other planners similar to the procedure in RQ2. We note that the use of BELLTREE offered statistically significant improvements compared to other methods.

Figure **??** shows the comparison between BELLTREE and other unsupervised planners. Note that, as with RQ2, BELLTREE outperforms other unsupervised planners.

Further, in some planners such as Alves et al. using the bellwether data to generate plans in an unsupervised fashion results in increasing the number of defects instead of decreasing it (the R scores are negative). Additionally, as noted in Figure 9, except for BELLTREE other planners recommend changes to almost all the metrics, most of the times, this cannot be considered a realistic approach as we discussed in RQ3. Thus, our answer to this research question is as follows:

RQ5: Are BELLTREE's cross-project plans any better than XTREE's within project plans?

To answer this research question, we train the planners on within-project data and generate plans for reducing the number of defects. We then compare this with plans derived from the bellwether data for that community. We hypothesized that since bellwethers have been demonstrated to be efficient in prediction tasks, learning from the bellwethers for a specific community of projects would produce performance scores comparable to within-project data. We found that this was indeed the case; i.e. when viewed in terms of *impact*, both XTREE and BELLTREE are similar.

Figure **??** tabulates the comparison between the use of XTREE and BELLTREE planning using the bellwether data set. The bellwethers for each community are highlighted in Figure 4. We note that in 6 out of 8 datasets, both plans generated from the bellwether and the local datasets are ranked similarly by the scott-knott test.

This finding has a major significance. The use of bellwether data implies that we use the same dataset to generate plans for all the project. As long as the bellwether data remains unchanged, so would the plans derived from it. This means that we can expect stable plans from BELLTREE. This is in contrast to using XTREE, where plans may change over time leading to unstable conclusions.

In summary, we make the following observations:

6 Discussion

2. Why use automatic methods to find quality plans? Why not just use domain knowledge; e.g. human expert intuition?

Much recent work has documented the wide variety of conflicting opinions among software developers, even those working within the same project. According to Passos et al. [23], developers often assume that the lessons they learn from a few past projects are general to all their future projects. They comment, past experiences were taken into account without much consideration for their context [34]. Jorgensen and Gruschke [24] offer a similar warning. They report that the supposed software engineering "gurus rarely use lessons from past projects to improve their future reasoning and that such poor past advice can be detrimental to new projects [24]. Other studies have shown some widely-held views are now questionable given new evidence. Devanbu et al. examined responses from 564 Microsoft software developers from around the world. They comment programmer beliefs can vary with each project, but do not necessarily correspond with actual evidence in that project [25].

Given the diversity of opinions seen among humans, it seems wise to explore automatic oracles for planning quality improvement change.

3. Do bellwethers guarantee that software managers will never have to change their plans?

No. Software managers should evolve their policies when the evolving circumstances require such an update. But how to know when to retain currently policies or when to switch to new ones? Bellwether methods can answer this question.

Specifically, we advocate continually retesting the bellwether's status against other data sets within the community. If a new bellwether is found, then it is time for the community to accept very different policies. Otherwise, it is valid for managers to ignore most the new data arriving into that community.

7 Threats to Validity

7.1 Sampling Bias

Sampling bias threatens any classification experiment; what matters in one case may or may not hold in another case. For example, even though we use a number of open-source datasets in this study which come from several sources, they were all supplied by individuals.

That said, this paper addresses this sampling bias problem. As researchers, all we can do is document our selection procedure for data and suggest that other researchers try a broader range of data in future work.

7.2 Learner Bias

For building the verification oracle in this study, we elected to use random forests. We chose this learner because past studies shows that, for prediction tasks, the results were superior to other more complicated algorithms [10] and can act as a baseline for other algorithms.

Apart from this choice, one other limitation to our current study is that we have focused here on one supervised planner (XTREE) and a number of unsupervised planners. The implications of our findings are limited by the efficacy of the verification oracle. We understand the implication of this, and thus propose undertaking a preliminary study to validate the verification oracle. The rest of the analysis can be limited to those datasets where the learner has satisfactory performance.

7.3 Evaluation Bias

This paper uses one measure for the quality of prediction the oracle (Recall and False Alarm) and for the planners we use R (see Equation 1). Other quality measures often used in software engineering to quantify the effectiveness of prediction [94] [3] [95]. A comprehensive analysis using these measures may be performed with our replication package. Additionally, other measures can easily be added to extend this replication package.

7.4 Order Bias

With random forest and the planners, there is invariably some degree of randomness that is introduced by both the algorithms. Random Forest, as the name suggests, randomly samples the data and constructs trees which it then uses in an ensemble fashion to make predictions.

To mitigate these biases, we run the experiments 30 times (the reruns are equal to 30 in keeping with the central limit theorem). Note that the reported variations over those runs were very small. Hence, we conclude that while order bias is theoretically a problem, it is not a major problem in the particular case of this study.

8 Conclusions and Future Work

Most software analytic tools that are currently in use today are mostly prediction algorithms. These algorithms are limited to making predictions. We extend this by offering "planning": a novel technology for prescriptive software analytics. Our planner offers users a guidance on what action to take in order to improve the quality of a software project. Our preferred planning tool is BELLTREE, which performs cross-project planning with encouraging results. With our BELLTREE planner, we show that it is possible to reduce defects in software projects by $> 50\%$ in 6 out of 8 cases (note that in several cases, we achieved $> 90\%$ reduction).

It is also worth noting that BELLTREE is a novel extension of our prior work on (1) the bellwether effect, and (2) within-project planning with XTREE. With the bellwether effect we demonstrated that we can use stable sources of data to perform cross-project defect prediction without the need for more complex transfer learners. With XTREE we introduced a supervised $Cluster + Contrast$ planning framework that can be used to generate effective within-project plans. In this work, we show that it is possible to use bellwether effect and within-project planning (with XTREE) to perform cross-project planning using BELLTREE. Our results from Figure **??** show that in 4 out of 8 cases, BELLTREE is just as good as XTREE and in the remaining 4 cases XTREE performs only marginally better than BELLTREE.

Further, we can see from Figure 9 that both BELLTREE and XTREE recommend changes to very few metric, while other unsupervised planners such as Shatnawi, Alves, and Olivera, recommend changing most of the metrics. This is not practical in many real world scenarios.

Finally, we note that BELLTREE can offer stable solutions. As long as the bellwether data from which BELLTREE is constructed remains unchanged, so would plans that are derived from it. In this sense, practitioners can expect stable plans for relatively longer periods of time. This is in contrast to our XTREE planner. As more within-project data is gathered, it is important that XTREE planner be updated. Such constant updates would invariably lead to unstable and often contradicting plans.

As for future work, we would like to undertake the following tasks:

1) *Industrial validation*: One of immediate goals is to validate the usefulness of these planners in a realistic development environment. As the first steps towards this, we are currently collaborating with a software company in RTP to deploy XTREE in their pipeline.
2) *Developer survey:* It would also be valuable to solicit developers' feedback on planners such as XTREE. A study of their willingness to use such tools would greatly benefit the software analytics community.

3) *Scaling planners:* It must be noted that the datasets studied here are relatively small. In order to be able to use XTREE as a real time planner for very large projects. It is very important to scale these planners to accommodate very large datasets.

REFERENCES

[1] J. Czerwonka, R. Das, N. Nagappan, A. Tarvo, and A. Teterev, "Crane: Failure prediction, change analysis and test prioritization in practice – experiences from windows," in *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, march 2011, pp. 357 –366.

[2] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," in *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*. New York, NY, USA: ACM, 2004, pp. 86–96.

[3] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with Precision: A Response to "Comments on 'Data Mining Static Code Attributes to Learn Defect Predictors'"," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 637–640, sep 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4288197

[4] B. Turhan, A. Tosun, and A. Bener, "Empirical evaluation of mixed-project defect prediction models," in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*. IEEE, 2011, pp. 396–403.

[5] E. Kocaguneli, T. Menzies, A. Bener, and J. Keung, "Exploiting the essential assumptions of analogy-based effort estimation," *IEEE Transactions on Software Engineering*, vol. 28, pp. 425–438, 2012, available from http://menzies.us/pdf/11teak.pdf.

[6] A. Begel and T. Zimmermann, "Analyze this! 145 questions for data scientists in software engineering," in *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, June 2014. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=208800

[7] C. Theisen, K. Herzig, P. Morrison, B. Murphy, and L. Williams, "Approximating attack surfaces with stack traces," in *ICSE'15*, 2015.

[8] J. Hihn and T. Menzies, "Data mining methods and cost estimation models: Why is it so hard to infuse new ideas?" in *2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*, Nov 2015, pp. 5–9.

[9] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[10] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, jul 2008. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4527256$\delimiter"026E30F$nhttp://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=4527256http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4527256

[11] R. Krishna, T. Menzies, and W. Fu, "Too much automation? the bellwether effect and its implications for transfer learning," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016*. New York, New York, USA: ACM Press, 2016, pp. 122–131. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2970276.2970339

[12] R. Krishna, T. Menzies, and L. Layman, "Less is more: Minimizing code reorganization using XTREE," *Information and Software Technology*, mar 2017. [Online]. Available: http://arxiv.org/abs/1609.03614http://linkinghub.elsevier.com/retrieve/pii/S0950584916301641

[13] R. Shatnawi, "A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 216–225, March 2010.

[14] T. L. Alves, C. Ypma, and J. Visser, "Deriving metric thresholds from benchmark data," in *2010 IEEE Int. Conf. Softw. Maint.* IEEE, sep 2010, pp. 1–10. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5609747

[15] P. Oliveira, M. T. Valente, and F. P. Lima, "Extracting relative thresholds for source code metrics," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, feb 2014, pp. 254–263. [Online]. Available: http://ieeexplore.ieee.org/document/6747177/

[16] R. Krishna and T. Menzies, "Simpler Transfer Learning (Using "Bellwethers")," *TSE (under review)*, pp. 1–18, mar 2017. [Online]. Available: http://arxiv.org/abs/1703.06218

[17] A. Hassan, "Remarks made during a presentation to the ucl crest open workshop," March 2017.

[18] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction," *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering on European software engineering conference and foundations of software engineering symposium - E*, p. 91, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1595696.1595713

[19] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE transactions on software engineering*, vol. 33, no. 1, pp. 2–13, 2007.

[20] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann, "Local versus Global Lessons for Defect Prediction and Effort Estimation," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 822–834, jun 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6363444/

[21] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.

[22] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2017.

[23] C. Passos, A. P. Braun, D. S. Cruzes, and M. Mendonca, "Analyzing the impact of beliefs in software project practices," in *ESEM'11*, 2011.

[24] M. Jørgensen and T. M. Gruschke, "The impact of lessons-learned sessions on effort estimation and uncertainty assessments," *Software Engineering, IEEE Transactions on*, vol. 35, no. 3, pp. 368 –383, May-June 2009.

[25] P. Devanbu, T. Zimmermann, and C. Bird, "Belief & evidence in empirical software engineering," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 108–119.

[26] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Egnlewood Cliffs, 1995.

[27] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Elsevier, 2004.

[28] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[29] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *The knowledge engineering review*, vol. 10, no. 2, pp. 115–152, 1995.

[30] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[31] E. Altman, *Constrained Markov decision processes*. CRC Press, 1999, vol. 7.

[32] X. Guo and O. Hernández-Lerma, "Continuous-time markov decision processes," *Continuous-Time Markov Decision Processes*, pp. 9–18, 2009.

[33] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[34] T. C. Son and E. Pontelli, "Planning with preferences using logic programming," *Theory and Practice of Logic Programming*, vol. 6, no. 5, pp. 559–607, 2006.

[35] S. S. J. A. Baier and S. A. McIlraith, "Htn planning with preferences," in *21st Int. Joint Conf. on Artificial Intelligence*, 2009, pp. 1790–1797.

[36] S. Tallam and N. Gupta, "A concept analysis inspired greedy algorithm for test suite minimization," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 1, pp. 35–42, 2006.

[37] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[38] D. Blue, I. Segall, R. Tzoref-Brill, and A. Zlotnick, "Interaction-based test-suite minimization," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 182–191.

[39] G. Ruhe and D. Greer, "Quantitative studies in software release planning under risk and resource constraints," in *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*. IEEE, 2003, pp. 262–270.

[40] G. Ruhe, *Product release planning: methods, tools and applications*. CRC Press, 2010.

[41] M. Harman, S. A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," *Department of Computer Science, Kings College London, Tech. Rep. TR-09-03*, 2009.

[42] M. Harman, P. McMinn, J. De Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," *Search*, vol. 2012, pp. 1–59, 2011. [Online]. Available: http://discovery.ucl.ac.uk/1340709/

[43] J. Krall, T. Menzies, and M. Davies, "Gale: Geometric active learning for search-based software engineering," *IEEE Transactions on Software Engineering*, vol. 41, no. 10, pp. 1001–1018, 2015.

[44] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2002.

[45] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization," in *Evolutionary Methods for Design, Optimisation, and Control*. CIMNE, Barcelona, Spain, 2002, pp. 95–100.

[46] E. Zitzler and S. Künzli, "Indicator-Based Selection in Multiobjective Search," in *Parallel Problem Solving from Nature - PPSN VIII*, ser. Lecture Notes in Computer Science, X. Yao, E. Burke, J. Lozano, J. Smith, J. Merelo-Guervós, J. Bullinaria, J. Rowe, P. Tio, A. Kabán, and H.-P. Schwefel, Eds. Springer Berlin Heidelberg, 2004, vol. 3242, pp. 832–842. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30217-9_84

[47] K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," *Evolutionary Computation, IEEE Transactions on*, vol. 18, no. 4, pp. 577–601, Aug. 2014.

[48] X. Cui, T. Potok, and P. Palathingal, "Document clustering using particle swarm optimization," *… Intelligence Symposium, 2005. …*, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1501621

[49] Q. Zhang and H. Li, "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition," *Evolutionary Computation, IEEE Transactions on*, vol. 11, no. 6, pp. 712–731, Dec. 2007.

[50] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest, "Automatically finding patches using genetic programming," in *Proceedings - International Conference on Software Engineering*. IEEE, 2009, pp. 364–374. [Online]. Available: http://ieeexplore.ieee.org/document/5070536/

[51] S. Forrest, T. Nguyen, W. Weimer, and C. Le Goues, "A genetic programming approach to automated software repair," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09*. New York, New York, USA: ACM Press, 2009, p. 947. [Online]. Available: http://portal.acm.org/citation.cfm?id=1570031http://portal.acm.org/citation.cfm?doid=1569901.1570031

[52] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, "A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, jun 2012, pp. 3–13. [Online]. Available: http://ieeexplore.ieee.org/document/6227211/

[53] E. Schulte, S. Forrest, and W. Weimer, "Automated program repair through the evolution of assembly code," in *Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10*. New York, New York, USA: ACM Press, 2010, p. 313. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1858996.1859059

[54] C. Le Goues, N. Holtschulte, E. K. Smith, Y. Brun, P. Devanbu, S. Forrest, and W. Weimer, "The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs," *IEEE Transactions on Software Engineering*, vol. 41, no. 12, pp. 1236–1256, dec 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7153570/

[55] P. Clements and L. Northrop, *Software product lines*. Addison-Wesley,, 2002.

[56] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "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, 2013, pp. 465–474.

[57] A. Metzger and K. Pohl, "Software product line engineering and variability management: achievements and challenges," in *Proceedings of the on Future of Software Engineering*. ACM, 2014, pp. 70–84.

[58] C. Henard, M. Papadakis, M. Harman, and Y. L. Traon, "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, May 2015, pp. 517–528.

[59] J. Andrews and T. Menzies, "On the Value of Combining Feature Subset Selection with Genetic Algorithms: Faster Learning of Coverage Models," in *PROMISE'09*, 2009.

[60] J. H. Andrews, F. C. H. Li, and T. Menzies, "Nighthawk: A Two-Level Genetic-Random Unit Test Data Generator," in *IEEE ASE'07*, 2007.

[61] J. H. Andrews, T. Menzies, and F. C. H. Li, "Genetic Algorithms for Randomized Unit Testing," *IEEE Transactions on Software Engineering*, Mar. 2010.

[62] T. Menzies, M. S. Feather, R. Madachy, and B. W. Boehm, "The Business Case for Automated Software Engineering," in *Proc. ASE*. New York, NY, USA: ACM, 2007, pp. 303–312. [Online]. Available: http://portal.acm.org/citation.cfm?id=1321631.1321676

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

[64] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts, *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[65] P. Green, T. Menzies, S. Williams, and O. El-waras, "Understanding the Value of Software Engineering Technologies," in *IEEE ASE'09*, 2009.

[66] B. Lemon, A. Riesbeck, T. Menzies, J. Price, J. D'Alessandro, R. Carlsson, T. Prifiti, F. Peters, H. Lu, and D. Port, "Applications of Simulation and AI Search: Assessing the Relative Merits of Agile vs Traditional Software Development," in *IEEE ASE'09*, 2009.

[67] M. S. Feather and T. Menzies, "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*, 2002.

[68] J. Nam and S. Kim, "Heterogeneous defect prediction," in *Proc. 2015 10th Jt. Meet. Found. Softw. Eng. - ESEC/FSE 2015*. New York, New York, USA: ACM Press, 2015, pp. 508–519. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2786805.2786814

[69] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empir. Softw. Eng.*, vol. 17, no. 4-5, pp. 531–577, aug 2012. [Online]. Available: http://link.springer.com/10.1007/s10664-011-9173-9

[70] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "ReLink," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng. - SIGSOFT/FSE '11*. New York, New York, USA: ACM Press, 2011, p. 15. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-80053202000{&}partnerID=40{&}md5=0bc9b3583d933d489020762df57b9c31http://dl.acm.org/citation.cfm?doid=2025113.2025120

[71] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *Software Engineering, IEEE Transactions on*, vol. 22, no. 10, pp. 751–761, 1996.

[72] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *Software Engineering, IEEE Transactions on*, vol. 22, no. 12, pp. 886–894, 1996.

[73] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE, 2011, pp. 481–490.

[74] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proc. 6th Int. Conf. Predict. Model. Softw. Eng. - PROMISE '10*. New York, New York, USA: ACM Press, 2010, p. 1. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1868328.1868342

[75] B. Cheng and A. Jensen, "On the use of genetic programming for automated refactoring and the introduction of design patterns," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 1341–1348. [Online]. Available: http://doi.acm.org/10.1145/1830483.1830731

[76] M. O'Keeffe and M. O. Cinnéide, "Search-based refactoring: An empirical study," *J. Softw. Maint. Evol.*, vol. 20, no. 5, pp. 345–364, Sep. 2008. [Online]. Available: http://dx.doi.org/10.1002/smr.v20:5

[77] M. K. O'Keeffe and M. O. Cinneide, "Getting the most from search-based refactoring," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '07. New York, NY, USA: ACM, 2007, pp. 1114–1120. [Online]. Available: http://doi.acm.org/10.1145/1276958.1277177

[78] I. H. Moghadam, *Search Based Software Engineering: Third International Symposium, SSBSE 2011, Szeged, Hungary, September 10-12, 2011. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ch. Multi-level Automated Refactoring Using Design Exploration, pp. 70–75. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23716-4_9

[79] M. W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, and M. Ó Cinnéide, "Recommendation system for software refactoring using innovization and interactive dynamic optimization," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. New York, NY, USA: ACM, 2014, pp. 331–336. [Online]. Available: http://doi.acm.org/10.1145/2642937.2642965

[80] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4–17, Jan. 2002. [Online]. Available: http://dx.doi.org/10.1109/32.979986

[81] L. Breiman, "Random forests," *Machine learning*, pp. 5–32, 2001. [Online]. Available: http://link.springer.com/article/10.1023/A:1010933404324

[82] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings - International Conference on Software Engineering*, 2013, pp. 382–391.

[83] J. Nam and S. Kim, "Heterogeneous defect prediction," in *Proc. 2015 10th Jt. Meet. Found. Softw. Eng. - ESEC/FSE 2015*. New York, New York, USA: ACM Press, 2015, pp. 508–519. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2786805.2786814

[84] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: is it really necessary?" *Information and Software Technology (submitted)*, 2016, read on-line at https://goo.gl/Jp5VIm.

[85] T. Menzies, E. Kocaguneli, B. Turhan, L. Minku, and F. Peters, *Sharing data and models in software engineering*. Morgan Kaufmann, 2014.

[86] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *37th ICSE-Volume 1*. IEEE Press, 2015, pp. 789–800.

[87] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *ICSE'11*, 2011, pp. 1–10.

[88] U. Fayyad and K. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," *NASA JPL Archives*, 1993.

[89] V. Papakroni, "Data carving: Identifying and removing irrelevancies in the data," Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia Unviersity, 2013.

[90] N. Mittas and L. Angelis, "Ranking and clustering software cost estimation models through a multiple comparisons algorithm," *IEEE Trans. Software Eng.*, vol. 39, no. 4, pp. 537–551, 2013.

[91] R. Bender, "Quantitative risk assessment in epidemiological studies investigating threshold effects," *Biometrical Journal*, vol. 41, no. 3, pp. 305–319, 1999. [Online]. Available: http://dx.doi.org/10.1002/(SICI)1521-4036(199906)41:3⟨305::AID-BIMJ305⟩3.0.CO;2-Y

[92] J. A. Swets, "Measuring the accuracy of diagnostic systems," *Science*, vol. 240, no. 4857, p. 1285, 1988.

[93] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[94] Y. Ma and B. Cukic, "Adequate and precise evaluation of quality models in software engineering studies," in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, May 2007, pp. 1–1.

[95] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: is it really necessary?" *Information and Software Technology (submitted)*, 2016, read on-line at https://goo.gl/Jp5VIm.