Release Readiness Classification – An Explorative Case Study

S M Didar Al Alam
SEDS Laboratory, Dept. of CPSC,
University of Calgary
Calgary, AB, Canada
smdalam@ucalgary.ca

Dietmar Pfahl
Institute of Computer Science
University of Tartu
Tartu, Estonia
dietmar.pfahl@ut.ee

Guenther Ruhe
SEDS Laboratory, Dept. of CPSC,
University of Calgary
Calgary, AB, Canada
ruhe@ucalgary.ca

ABSTRACT

Context: To survive in a highly competitive software market, product managers are striving for frequent, incremental releases in ever shorter cycles. Release decisions are characterized by high complexity and have a high impact on project success. Under such conditions, using the experience from past releases could help product managers to take more informed decisions.

Goal and research objectives: To make decisions about when to make a release more operational, we formulated release readiness (RR) as a binary classification problem. The goal of our research presented in this paper is twofold: (i) to propose a machine learning approach called RC* (Release readiness Classification applying predictive techniques) with two approaches for defining the training set called incremental and sliding window, and (ii) to empirically evaluate the applicability of RC* for varying project characteristics.

Methodology: In the form of explorative case study research, we applied the RC* method to four OSS projects under the Apache Software Foundation. We retrospectively covered a period of 82 months, 90 releases and 3722 issues. We use Random Forest as the classification technique along with eight independent variables to classify release readiness in individual weeks. Predictive performance was measured in terms of precision, recall, F-measure, and accuracy.

Results: The incremental and sliding window approaches respectively achieve an overall 76% and 79% accuracy in classifying RR for four analyzed projects. Incremental approach outperforms sliding window approach in terms of stability of the predictive performance. Predictive performance for both approaches are significantly influenced by three project characteristics i) release duration, ii) number of issues in a release, iii) size of the initial training dataset. Our initial analysis shows, incremental approach achieves higher accuracy when releases have long duration, low number of issues and classifiers are trained with large training set. On the other hand, sliding window approach achieves higher accuracy when releases have short duration and classifiers are trained with small training set.

Keywords

Release readiness classification; explorative case study; comparative analysis; evaluation.

1. INTRODUCTION

Context: Product managers strive for fast, short, and incremental releases of their products to survive in the highly competitive software industry. Release decisions are often made based on a product manager's experience and gut feeling. Wrong release decisions such as an early or late release may yield negative consequences for the software supplier with respect to cost/benefit ratio, customer satisfaction and overall product success. In consideration of the complexity and risk associated with any release decision made, a technique that systematically analyzes past releases to predict current release readiness (RR) might be helpful.

Motivation: RR prediction has received attention in the scientific literature. The majority of existing approaches quantifies RR with fuzzy values [4, 10]. The disadvantage of these approaches is the difficulty of interpreting fuzzy values purely based on a product manager's experience [10, 12]. In our former study published in RAISE 2016 [1], we simplify RR prediction by formulating it as a binary classification problem. Here classification refers to the task of identifying whether a software release is likely going to be ready or non-ready at the end of a release cycle. Comparing nine machine learning techniques for RR classification, we found that Random Forest (RF) performed best. Therefore, in this paper, we use Random Forest as the classifier in our RC* method (Release readiness Classification applying predictive techniques).

To evaluate RC*, cross validation is not applicable, because this randomly creates and selects training and testing data folds. Since this approach does not preserve the actual order of releases we used different approaches to create training and test datasets. We arranged releases in the temporal order in which they were published. Taking the chronological ordering of release-related data under consideration, we used two different approaches for constructing training and testing data sets for the evaluation of RC*.

- **Incremental Approach:** Inclusion of a new week's information increases the training dataset size. This approach considers all past weeks prior to the current week as training dataset.
- **Sliding Window Approach:** Inclusion of a new week's information and elimination of the oldest week from the training dataset. This approach considers a fixed number of most recent weeks in the training dataset.

Case study and Results: In the form of explorative case study research, we empirically analyze four OSS projects from Jira issue tracking system under Apache Software Foundation (ASF). RC* predicts the release readiness (RR) class with respect to eight independent variables. The training dataset is defined applying two approaches, i.e., incremental and sliding window. Our explorative analysis of OSS projects yielded the following key results (details in Section 5).

- RC* can predict release readiness class with 76% and 79% overall accuracy when applying the incremental and sliding window approaches, respectively.

- Incremental approach outperforms sliding window approach in terms of stability of their predictive performance in applying RC*.
- Three project characteristics under investigation i.e. i) release duration, ii) number of issues in a release, iii) size of the initial training dataset has shown significant influence on predictive performance for both approaches.
- Incremental approach achieves higher accuracy when releases have long duration, low number of issues and classifiers are trained with large training set. On the other hand, sliding window approach achieves higher accuracy when releases have short duration and classifiers are trained with small training set.

2. RELATED WORK

The value of measuring and predicting RR has been described in [12]. While the problem of predicting RR received attention in research, the definition of RR is not yet well established. Quah et al. [9] defined RR mainly based on the status of defect tracking. Wild et al. [15] considered multiple factors (e.g. requirements, functionality, reliability) in defining RR. Commercial tools like Borland Team Inspector and PTC Integrity extract and visualize multiple metrics related to functionality, code analysis, and test coverage etc. to verify RR.

We identified three issues which are often evident in existing RR prediction methods.

- **Lack of comprehensiveness:** The majority of existing approaches measure RR exclusively based on quality metrics [4, 10]. This restricts a product manager's view on RR and does not consider other factors relevant for judging the completeness of a product, e.g. requirements coverage.
- **Lack of data availability:** Some approaches [10, 12] attempt to aggregate multiple RR attributes in one single quantitative RR measure. Unavailability of required data in past releases make these approaches dependent on domain experts' assumptions needed to fill data gaps [11–13].
- **Lack of continuity:** The majority of existing approaches identify RR towards the end of a release cycle [4, 8, 11]. Due to lack of continuity in monitoring readiness of the product, product managers are unable to detect potential release problems early and thus cannot take actions to address them.

The proposed RC* method predicts the RR class (ready or non-ready) at any point in time within a release cycle. It applies eight RR attributes selected from multiple dimensions (e.g. implementation, quality) for prediction. Thus, RC* offers improved visibility of overall completeness and continuity in monitoring. RC* simplifies the interpretation of RR by answering the binary question "whether the release is going to be ready at the end of the planned release duration?" It utilizes past release history to minimize expert dependency. RC* applies the machine learning technique Random Forest (RF) for classification. Similar techniques have been found to successfully solve various software engineering prediction problems, e.g., in the context of quality prediction [6] and effort estimation [5].

3. RESEARCH OBJECTIVES

Extending former investigations published at RAISE 2016 [1], we further analyze our RR classification problem for both learning approaches and varying project characteristics. In total, we address two research objectives:

- **RO-1:** Analysis of the impact of *incremental* and *sliding window* training set approach in terms of quality and stability of prediction of RC*.
- **RO-2:** Analysis of RC* performance for varying project characteristics.

In our study, RO1 emphasize quality and stability of the prediction, and RO2 focus on the influence of three project characteristics i) release duration, ii) number of issues in a release, iii) size of the initial training dataset.

4. CASE STUDY DESIGN

In this section, we present the design of our explorative case study along step by step.

4.1 Preparation

There is a lack of published data documenting release practices. In absence of access to data from proprietary projects, we chose to use OSS projects in our case study. We selected projects from the JIRA issue tracking system[1] having the following characteristics:

- Follows a planned development process (e.g., RUP, Kanban, Scrum).
- Provides data on development progress in issue, code and bug repositories (e.g., JIRA, GitHub, Bugzilla).
- Reports release completeness information on former releases.
- Has existing release history.

From crawling more than 250 OSS projects, we identified 56 projects partially fulfilling the above criteria. To further filter inactive and toy projects, we verified access to JIRA, Scrum board and corresponding code repository. To ensure availability of historical data, we discarded projects with less than five releases. This further narrowed down the number of selected projects to four. The projects are Aurora (A), Slider (S), Usergrid (U), and Helix (H). For data analysis, we used the open source machine learning tool Weka[2].

4.2 Definition of Project Measures

Release and Observation Period: The selected projects practice Scrum development. We defined our observation period and release dates with respect to the time horizon available in the Scrum board. Table 1 summarizes key characteristics of the selected projects.

Table 1. Key characteristics of the selected projects.

Project Name	Studied duration (months)	Number of releases	Number of issues	Age of project (months)
Aurora (A)	21	34	1511	50
Slider (S)	21	17	1023	21
Usergrid (U)	28	34	1163	28
Helix (H)	12	5	25	28

RR Attributes: Following our goal-oriented approach in [2], we empirically investigate a set of product and process related RR attributes which were shown to influence release readiness (cf. Table 2). We consider these RR attributes as independent variables when building the classifier using the RF technique. These variables represent three dimensions of the RR problem, i.e.

[1] https://www.atlassian.com/software/jira

[2] http://www.cs.waikato.ac.nz/ml/weka/

(i) Quality, (ii) Implementation and (iii) Time. Details related to these dimensions are described in [2].

Table 2. List of independent variables and their definitions

Dimension	RR Attributes	Definition
Quality	Quality issue incoming rate at week k	Identified quality issues in week k / Total number of identified quality issues up to that week
Quality	Quality issue resolution rate at the end of week k	Solved quality issues in week (k) / Total number of identified quality issues up to that week
Quality	Open quality issues at the end of a week	Unresolved quality issues at that week
Implement ation	Implementation issues incoming rate at the end of week k	Requested implementation issues in week (k) /Requested implementation issues up to week (k)
Implement ation	Implementation issue completion rate at the end of week k	Completed implementation issues in week (k) / Total number of requested implementation issues up to that week
Implement ation	Unsolved implementation issues at the end of week k	Incomplete implementation issues in week (k)
Time	Release duration	Expected duration of current release (days)
Time	Elapsed duration	Days passed of current release / Release duration (days)

4.3 Data Collection

We selected four OSS projects under *Apache Software Foundation (ASF)*. *ASF* is a decentralized developer community, where projects are carried out using a collaborative, consensus-based development process and managed by self-selected teams that actively contribute to the project. We extract issue data from the *JIRA Issue Tracking System*[3], using *JRJC*[4] (*Jira Rest Java Client)* with a Java program. Initially extracted unstructured text files are further filtered using text filters to retrieve issue relevant information. We collect data on a weekly basis. Issues are resolved in multiple releases following their opening and resolution dates.

4.4 Preparation of Training Dataset

In preparation of the training dataset, first we calculate selected RR attributes from extracted data using the definitions presented in Table 2. We further label all past releases as either *ready* or *non-ready***.** This classification is performed retrospectively based on the percentage (denoted as *release policy*) of issues resolved in a release with respect to the expected release plan.

Definition (Release Policy): For a given project P and observation period $[0,T]$, $Release\ policy\ (y)$ considers all releases within $[0,T]$ as *ready* if y% of all issues from that release are resolved, and *non-ready* otherwise.

[3] https://www.atlassian.com/software/jira

[4] https://marketplace.atlassian.com/plugins/com.atlassian.jira.jira-rest-java-client/server/overview

The number of *ready* and *non-ready* releases change depending on the applied release policy. We assume that releases are classified as `ready` even though not all issues are resolved. The number of *ready* releases decreases with higher release policies. From a more technical perspective, the numbers of *ready* and *non-ready* releases can become extremely unbalanced. For example, under $Release\ policy\ (100)$ the numbers of *ready* releases for Aurora, Slider, and Usergrid are 4, 2, and 0, respectively.

We avoided oversampling by balancing the training dataset due to two reasons: (i) this may lead to over-fitted models, and (ii) the minority class members are significantly low and insufficient for creating meaningful synthetic data. As a result, for each project we investigate multiple release policies as shown in equation (1). We restrict our analysis to release policies where the numbers of *ready* and *non-ready* releases are most balanced (cf. Figure 1).

$$Release\ policy\ (y)\ where\ y = \{50,60,70,80,90,100\} \quad (1)$$

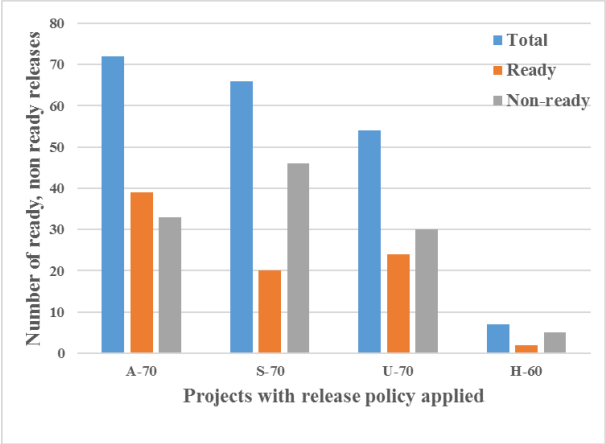

Figure 1: Number of *ready, non-ready* releases analyzed.

4.5 Evaluation Set-up

At any point in time during the release cycle, RR classification helps product managers to understand the status of the ongoing release. In [1], when comparing nine different classifiers, we found that Random Forest (RF) [3] performs best for RR classification. RF combines an ensemble of decision trees for classification. Each tree is built based on the value of an independent set of random vectors. Key benefits of this approach are (i) less sensitive to outliers, (ii) over-fitting is a less serious concern, (iii) accuracy and variable importance can be generated automatically.

We assume that for any project P with an observation period $[0,T]$, at any given week $t = t_0 \in [0,T]$, we have a set of variables available as presented in equation (2).

$$V(t_0) = \{A(t_0), RRC\ (t_0)\} \quad (2)$$

- n is the number of independent variables (n=8)
- $A(t_0)$ refers to the values of independent variables at week $t = t_0$ i.e. $A(t_0) = \{a_1(t_0), a_2(t_0), \ldots, a_n(t_0)\}$
- $RRC(t_0)$ - binary variable describing *Release Readiness* at the end of week $t = t_0$.

For predicting $RRC(t_x)$ for any week $t = t_x$, we assume that (i) weights of independent variables are equal, (ii) the values of all independent and dependent variables for all previous weeks are known, and (iii) values of the independent variables in the current week are known.

To predict $RRC(t_x)$

$$RRC(t_x) = RC * \ (D(t_x), A(t_x)) \quad (3)$$

we apply RC* with training set $D(t_x)$, where

$$D(t_x) = \{V(t_0), V(t_1), \ldots . V(t_{(x-1)})\} \quad (4)$$

Values for the training dataset are dynamically changing and updated with inclusion of each new week's information. The inclusion of a new week's information $V(t_x)$ in the updated training dataset $D(t_{x+1})$ follows the incremental or sliding window approach. In the *incremental* variant of RC*, any new week is added to the existing training dataset, thus each time increasing the training size by 1. The sliding window variant of RC* includes a new week's information and eliminates the oldest week from the training dataset. Thus always maintains a fixed training dataset size with most recent weeks.

5. CASE STUDY RESULTS

In what follows, we present the analysis and findings from studying our research objectives (RO). In the form of an explorative study, we first compare the performance of RC* using the *incremental* approach against RC* using the *sliding window* approach (RO1). Then we investigate the performance of RC* in dependence of project characteristics influence (RO2).

5.1 RO1: Analysis of incremental and sliding window training set approach

RO1 investigates whether it is feasible to build classifiers applying incremental or sliding window approaches to predict RR class. It comparatively analyze the predictive performance of the incremental ($RC *_{inc}$) and sliding window ($RC *_{slw}$) approaches in terms of quality and stability from applying them to four OSS projects. $RC *_{inc}\ (x)$ and $RC *_{slw}\ (x)$ are determined following Section 4.5 while Quality and Stability are defined as below:

Definition (Quality): Quality of any approach is defined by comparing predicted result $RRC(t_i)$ for each week $t = t_i \in [t_x,\ T]$ with actual performance.

Definition (Stability): Stability refers to the variance between the prediction accuracy of consecutive experiments. We determine stability following equation (5) where $RC *(x)$ refers to prediction of $RRC(t_x)$.

$$stability = \frac{\sum_{t_x} abs(RC * (i+1) - RC * (i))}{(T - t_x)} \quad (5)$$

The value of higher stability is that in that case predictions are less sensitive to changes in the time. We investigate quality and stability for both approaches while applying different initial training set and sliding window sizes, i.e.,$x \in [5,40]$. Figure 2 presents the prediction accuracy (y-axis) of individual projects achieved applying both approaches with different size of the initial training set (x-axis). In Figure 3, we further summarize quality and stability measures of each approach for individual projects and across projects.

For RO1, we made the following observations:

Finding 1.1: Incremental and sliding-window approaches effectively predict RR classes with overall accuracy of 76% and 79%, respectively.

Finding 1.2: Incremental approach outperforms sliding window approach in terms of stability of the prediction.

- **Finding 1.3:** None of the approaches clearly out-performs the other in terms of quality of the prediction.

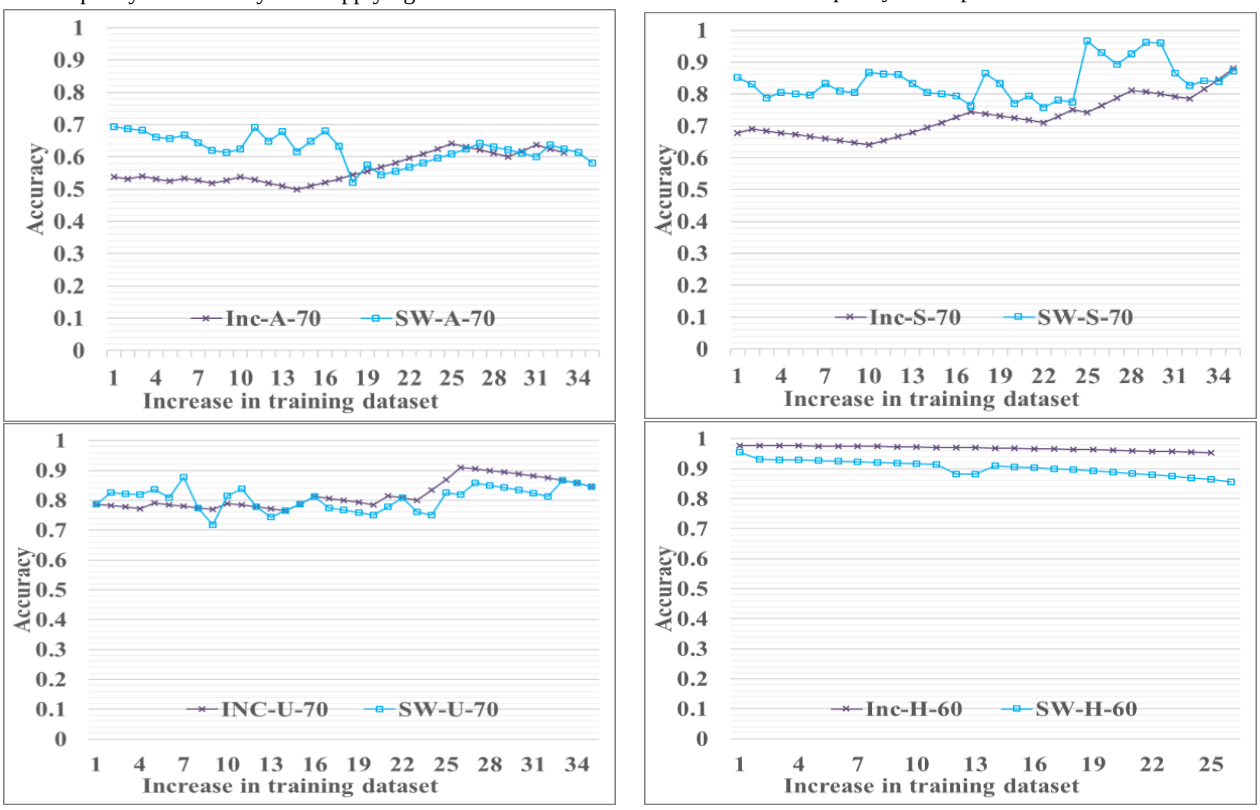

Figure 2. Prediction accuracy achieved applying incremental and sliding window variant of RC* method on projects Aurora (A-70) Slider (S-70), Helix (H-60), Usergrid (U-70) (clockwise from top left corner).

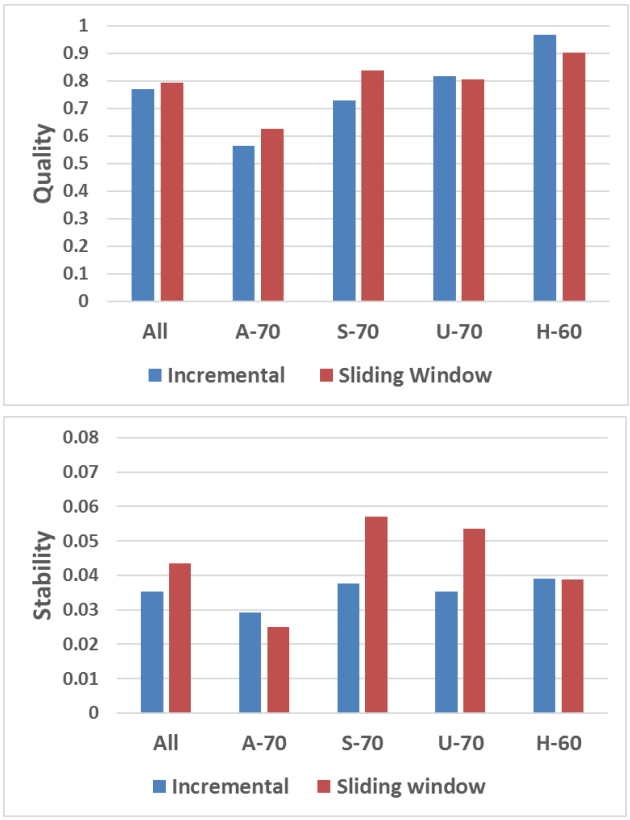

Figure 3. Comparison of incremental and sliding window approach in terms of quality (top) and stability (bottom).

5.2 RO2: Analysis of RC* quality of results for varying project characteristics

RO1 investigates quality and stability of the RC* method. It is equally important to investigate applicability of the RC* method i.e. whether the method is equally effective under different project setup. Going beyond overall performance of $RC*_{inc}$ and $RC*_{slw}$, RO2 investigates whether project characteristics significantly influence the predictive performance of RC*. In this paper, we limit our investigation in three individual project characteristics i.e. i) number of issues in a release, ii) release duration and iii) size of the training dataset. For individual project characteristics, we subdivide all releases of a project into two groups as defined below:

- ***High and Low*** refers to release groups, where the number of expected issues are higher resp. lower than the mean number of issues per release observed in a project.
- ***Long* and *Short*** refers to release groups, where release duration is longer resp. shorter than 14 days (i.e., mean sprint size).
- ***Large and Small*** refer to experiments, where the initial training dataset size is larger resp. smaller than 17 weeks (i.e., median of our investigation range [5,40]).

Predictive performance is measured by the accuracy (Acc.) and F-measure of ready (F-R), and non-ready (F-NR) groups. To investigate the influence of these project characteristics, we further applied two-tailed Mann-Whitney U-test [7] to test corresponding null hypotheses:

- **Null hypothesis $H_{2.1}$:** There is no significant difference in predictive performance of $RC*$ when applied on *High* versus *Low* release groups.
- **Null hypothesis $H_{2.2}$:** There is no significant difference in predictive performance of $RC*$ when applied on *Long* versus *Short* release groups.
- **Null hypothesis $H_{2.3}$:** There is no significant difference in predictive performance of $RC*$ when applied on *Large* versus *Small* size initial training sets.

Table 3 and 4 report the p-values achieved from performing non-parametric Mann-Whitney test [7] with respect to Accuracy and F measure for ready (F-R), non-ready (F-NR) releases. In case of identifying significant difference, we also report the group, which achieved higher predictive performance. Key findings observed in these investigations are:

Table 3. Results of two-tailed Mann-Whitney U-test for investigating influence of project characteristics on $RC*_{inc}$ performance.

$RC*_{inc}$ with project characteristcs			F-R	F-NR	Acc.
Number of issues	High (H)	Low (L)	0.329	0.563	**0.046 (L)**
Release duration	Long (L)	Short (S)	0.278	**0.000 (L)**	**0.000 (L)**
Size training set	Large (L)	Small (S)	0.602	**0.023 (S)**	**0.032 (L)**

Table 4. Results of two-tailed Mann-Whitney U-test for investigating influence of project characteristics on $RC*_{sw}$ performance.

$RC*_{slw}$ with project characteristcs			F-R	F-NR	Acc.
Number of issues	High (H)	Low (L)	0.486	**0.005 (L)**	0.956
Release duration	Long (L)	Short (S)	0.664	**0.064 (S)**	**0.040 (S)**
Size training set	Large (L)	Small (S)	0.983	**0.062 (S)**	**0.039 (S)**

For RO2, we made the following observations:

Finding 2.1: Release duration significantly (at level 0.005) influence predictive performance of the incremental approach. Observed Acc. and F-NR are higher in *Long* release group.

Finding 2.2: The number of issues in a release significantly influence (at level 0.05) predictive performance of the incremental approach. Observed Acc. is higher in the *Low* release group.

Finding 2.3: The number of issues in a release significantly influence (at level 0.005) predictive performance of the sliding window approach. Observed F-NR is higher in *Low* release group.

Finding 2.4: Release duration significantly (at level 0.05) influences predictive performance of sliding window approach. Observed Acc. and F-NR are higher in *Short* release group.

Finding 2.5: The size of the initial training set significantly influences (at level 0.05) predictive performance of the sliding window approach. Observed Acc. and F-NR are higher in *Small* release group.

These findings further allow us to understand applicability of the RC* approach under varying project setup. Our initial observation shows, incremental approach achieves higher accuracy when releases have long duration, low number of issues and classifiers are trained with large training set. On the other hand, sliding window approach achieves higher accuracy when releases have short duration and classifiers are trained with small training set. Product manager may apply this knowledge to choose between different learning approaches while applying them on certain projects.

6. THREATS TO VALIDITY

This study is exploratory in nature and should be seen as the initial step of an ongoing effort to applying machine learning algorithms towards classifying RR. Since we considered four OSS projects, representativeness of these projects is a threat to the external validity of our observations. To reduce this threat, our selected projects conform to the four propositions on case selection by Verner et al. [14]: i) we can measure RR attributes at any time, ii) we can identify the overall RR, iii) collected metrics and their collection process is clearly defined, and iv) collected metrics are relevant for answering the RQs. We also consider a relatively long observation period for each project.

Selection of classification technique and RR attributes may introduce threats to construct validity for the RC* method. To reduce this threat, we selected RF as the classification technique, which was shown as the best predictor for RR classification in a comparative analysis among nine classifiers [1]. Prior to reporting the results, we also performed parameter tuning and identified the best configuration by comparing them applying the balanced accuracy measure. However, parameters are continuous and can take infinite number of possible values. Therefore, the constructed models might not necessarily be the best models for the given datasets. To reduce the threat related to RR attributes selection, we consider key dimensions of RR (e.g. implementation, testing). Selected RR attributes were shown as influential on RR in an explorative study [2] and represent 50% of RR attributes known from comprehensive industry guidelines.

7. SUMMARY AND FUTURE RESEARCH

Release readiness evaluation is of critical importance for release engineering. While inherently as difficult as predicting project success, we considered a simplified formulation of RR and applied RC* in an explorative case study set-up. The two main contributions of this paper are i) proposing and comparing two approaches for varying training dataset in RC* and ii) empirically evaluate the applicability of these approaches.

As a follow up, investigating varying project characteristics reports significant influence of three project characteristics on the predictive performance. We consider this research as the initial phase of a more comprehensive analysis with focus on:

- Incorporate prediction for different levels of readiness of a release, thus enhancing the current Boolean formulation of the problem.
- Analysis of the robustness of the results in dependence on the varying weights of RR.
- Tuning project characteristics weight factors to achieve better prediction results.
- Broadening the project scope to proprietary projects and comparison of results with observations from other OSS projects.
- Provide guidelines for which predictive technique is better suited for which type of release readiness prediction problem.

8. ACKNOWLEDGEMENTS

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada, NSERC Discovery Grant 250343-12, Alberta Innovates Technology Futures and by the institutional research grant IUT20-55 of the Estonian Research Council.

9. REFERENCES

[1] Alam, S. et al. 2016. Comparative Analysis of Predictive Techniques for Release Readiness Classification. *RAISE 2016* (2016).

[2] Alam, S. et al. 2015. Monitoring and Controlling Release Readiness by Learning across Projects. *Managing Software Process Evolution*. Springer.

[3] Cichosz, P. 2015. *Data Mining Algorithms: Explained Using R*. John Wiley and Sons.

[4] Mcconnell, S. 1997. Gauging software readiness with defect tracking. *IEEE Software*. 14, 3 (1997), 135–136.

[5] Minku, L.L. and Yao, X. 2013. Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology*. 55, 8 (2013), 1512–1528.

[6] Misirli, A.T. et al. 2011. An industrial case study of classifier ensembles for locating software defects. *Software Quality Journal*. 19, 3 (2011), 515–536.

[7] Motulsky, H. 2013. *Intuitive biostatistics: a nonmathematical guide to statistical thinking*. Oxford Univ. Press.

[8] Pearse, T. et al. 1999. Using Metrics to Manage the End-Game of a Software Project. *Proceedings of the Sixth International Software Metrics Symposium* (1999), 207–215.

[9] Quah, J.T.S. and Liew, S.W. 2008. Gauging Software Readiness Using Metrics. *SMCia* (2008), 426–431.

[10] Quah, T.-S. 2009. Estimating software readiness using predictive models. *Information Sciences*. 179, 4 (Feb. 2009), 430–445.

[11] R. Brettschneider 1989. Zero-Failure model -Is your software ready for release? *IEEE Software*. 6, 4 (1989).

[12] Shahnewaz, S. and Ruhe, G. 2014. RELREA - An Analytical Approach for Evaluating Release Readiness. *Proc. SEKE* (2014).

[13] Staron, M. et al. 2012. Release Readiness Indicator for Mature Agile and Lean Software Development Projects. *Agile Processes in Software Engineering and Extreme Programming*. (2012), 93–107.

[14] Verner, J.M. et al. 2009. Guidelines for industrially-based multiple case studies in software engineering. *RCIS* (2009), 313–324.

[15] Wild, R. and Brune, P. 2012. Determining Software Product Release Readiness by the Change-Error Correlation Function: On the Importance of the Change-Error Time Lag. *HICSS* (2012), 5360–5367.