Are We Ready to Release?
Analyzing Factors that Limit Release Readiness

S. M. Didar Al Alam, Jason Ho, Dietmar Pfahl, Guenther Ruhe, S. M. Shahnewaz

REPORT NUMBER 111/2014

SOFTWARE ENGINEERING DECISION SUPPORT LABORATORY
UNIVERSITY OF CALGARY

November, 2014
Are We Ready to Release? 
Analyzing Factors that Limit Release Readiness

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

Jason Ho  
Dept. of CPSC  
University of Calgary  
Calgary, Alberta, Canada  
hott@ucalgary.ca

Guenther Ruhe  
Dept. of CPSC & ECE  
University of Calgary  
Calgary, Alberta, Canada  
ruhe@ucalgary.ca

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

S. M. Shahnewaz  
Department of ECE  
University of Calgary  
Calgary, Alberta, Canada  
smshahne@ucalgary.ca

1. INTRODUCTION

Software releasing is the process of delivering the product into the operational environment for usage by its end users [1]. The product, or a major version of the product, can only be released when it is ready. This decision cannot be made ad hoc, but rather needs analytical evidence. Proactively, during the whole release cycle, it is important to know which factors are not performing sufficiently well (e.g. related to test performance) and likely to limit release readiness.

This paper discusses identifying, monitoring and analyzing attributes limiting release readiness. For that purpose, we present a three stages process that can be applied for any ongoing product release development project. Extending a former investigation [2], we have analyzed 34 open source projects from the open source systems GitHub repository. The projects were taken from two domains, i.e., desktop-based and web-based projects. From monitoring the performance of six established release readiness attributes, we focused on the lowest performing attributes when compared to plan. We call these attributes limiting or bottleneck attributes.

Early identification of bottleneck attributes is expected to help release engineers in proactively addressing potential resources limitations or to initiate process changes. As a form of learning across projects, we are interested in answering the following questions:

- How to monitor and analyze bottleneck attributes for release readiness?
- Are there any patterns in the occurrence frequency of bottleneck attributes?
- Do occurrence frequency patterns vary across domains and with regards to project-specific characteristics such as (i) project size, (ii) number of contributors, (c) release development phase?

Release decisions need to balance between release readiness and timing, to avoid releasing high quality software too late, or on-time delivery with low quality. In a survey by the National Institute of Standards and Technology, software errors cost the U.S $59.2 billion annually, from where $22 billion can potentially be removed by testing activities [3]. In an explorative study at NASA’s Jet Propulsion Laboratory (JPL) Port et al. [1] illustrated the value of monitoring RR for final project success.

2. MODELING OF RELEASE READINESS

In order to perform an analytical process for release readiness monitoring, analysis and improvement, we introduce the necessary concepts.

2.1 Release Readiness Attributes

Release readiness (RR) in general describes the status of a product release that is about to be shipped to customers. The term comprises various dimensions including status of implementation, build, status and scope of testing, documentation, risk analysis, and packaging. Each dimension is further described by a set of possible attributes. We will call them release readiness attributes (RR attributes).

Each attribute is quantitatively described by one or more metrics. For example, the attribute called feature implementation at week k can be measured (on a weekly base) by a metric called Metric(\text{feat\_impl.}, k) defined as follows:

\[
\text{Metric}(\text{feat\_impl.}, k) = (\text{# of features implemented up to week } k) / (\text{# of all features to be implemented in the release})
\]

While this metric might not measure the progress of implementation most accurately, primarily because it only refers to the completed features, it is good enough to support an analytical approach for monitoring, controlling and improving the release status.

2.2 Selection of Release Readiness Attributes

In practice, selection of attributes considered for deciding about release readiness varies among projects and companies. For the selection of RR attributes, different approaches have been applied. These approaches can be broadly classified into four categories: (i) checklist based approach, (ii) testing metrics based approach, (iii) defect prediction model based approach, and (iv) multi-dimensional metrics aggregation-based approach.

The most frequently applied approach to evaluate release readiness in practice is based on company-specific checklists [4]. Typically, checklists contain questions that are answered by either yes or no. While this is often hard to decide, the even more difficult question is how to aggregate the individual performance measures and how to initiate changes in the case that some risks are detected early. The majority of the former approaches [5][6][7][8] evaluated RR towards the end of the project and exclusively emphasized trends of the metrics related to defect
tracking and testing processes. Studying documented practices applied in both industry and academia, we elicited four key dimensions: Implementation of functionality, Testing, Source code quality, and Documentation.

From analyzing the frequency distribution of RR attributes from 16 related studies [4] we found that the majority of studies (14 out of 16) focused on testing scope and status dimension followed by implementation status (9 out of 16) dimension. Comparatively less number of studies reported source code quality (5 out of 16) and documentation scope and status (4 out of 16) related RR attributes.

The ultimate decision of RR attributes selection is upon the user. It is typically based on the organizational goals and customer expectations. We applied the Goal-Question-Metric (GQM) paradigm [9] to design an effective measurement program that fulfills the goal of overall RR evaluation.

Our selection of attributes and subsequent metrics for this paper was influenced by their (i) Acceptance in real world, (ii) Availability of corresponding data in the GitHub repository and (iii) Ease of calculation. Table 1 presents the results of applying the GQM approach for defining RR metrics corresponding to the RR attributes considered in our study. The metrics are called release readiness metrics.

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Attributes</th>
<th>Questions</th>
<th>Metric definitions</th>
<th>Acronyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation Status</td>
<td>Status of feature implementation</td>
<td>To what extent requested features are completed?</td>
<td># of features closed since the beginning of the release / # of features since the beginning of the release</td>
<td>FCR</td>
</tr>
<tr>
<td></td>
<td>Status of pull request completion</td>
<td>To what extent pull-requests are completed?</td>
<td># of pull-requests closed current week / # of pull-requests closed since the beginning of the release</td>
<td>PCR</td>
</tr>
<tr>
<td></td>
<td>Status of improvement completion</td>
<td>To what extent requested improvements are completed?</td>
<td># of improvements closed since the beginning of the release / # of improvements since the beginning of the release</td>
<td>ICR</td>
</tr>
<tr>
<td>Testing Status</td>
<td>Status of defect finding</td>
<td>To what extent the testing activity reducing the defects?</td>
<td># of defects found current week / # of defects found from the beginning of the release</td>
<td>DFR</td>
</tr>
<tr>
<td></td>
<td>Status of bug fixing</td>
<td>To what extent detected bugs are fixed?</td>
<td># of bugs closed since the beginning of the release / # of bugs from the beginning of the release</td>
<td>BFR</td>
</tr>
<tr>
<td></td>
<td>Status of source code stability</td>
<td>To what extent the source code is becoming stable?</td>
<td># of code churn in current week / # of code churn from the beginning of the release [7]</td>
<td>CCR</td>
</tr>
</tbody>
</table>

2.3 Local Release Readiness

We call an RR metric defined for an RR attribute a local RR metric. To analyze RR, we first calculate the local RR metrics. Then we combine the local RR metrics into a global RR metric. At any point in time $t^*$, for any of the selected release readiness attributes, the local release readiness status is defined as the deviation from plan in terms of its local RR metric. The measure is normalized to the [0,1] interval, where level 0 and 1 means that the status expected at this point in time is not at all respectively fully achieved. In general, the status will be somewhere between these two extreme points.

We illustrate our terminology via an example. For that purpose, we consider a project called $P$. For $P$, we consider one release of duration 20 weeks. To keep the example simple, we consider only two release readiness attributes: (a) Cumulative bug fix rate (BFR), and (b) Cumulative feature completion rate (FCR). We measure the local RR of BFR (on a weekly time interval) and assume we have a planned RR performance based on experience from successful former releases.

![Figure 1: Local RR evaluation for (a) BFR (left) and (b) FCR (right).](image-url)
As illustrated in Figure 1a, the planned (dotted line) rate at week 10 is higher than the computed actual rate (solid line). The relative performance degree is 0.88 for week 10. Similarly, we measure the local RR of FCR (on a weekly time interval) as defined in Table 1. The performance measure indicates that at week 10, the actual performance is only 48% of the planned performance.

2.4 Global Release Readiness

For any point in time (week) during the release, the global release readiness GRR(t) metric is defined as the Weighted Arithmetic Mean (WAM) of all local RR metrics. For the above example, GRR(10) is calculated as the arithmetic mean of local RR value of BFR(10) and FCR(10). This is illustrated in Figure 2. For equally weighted attributes, GRR(10) equals 0.33.

![Figure 2: Global RR evaluation for Release](image)

2.5 Bottleneck Attributes

For any given project, at any point in time during the release, the attribute with the lowest performing metric is called the bottleneck attribute. In our example, FCR represents the bottleneck at week 10 as its relative performance is smaller than that of BFR. In order to better understand which RR attributes become bottleneck attributes more or less frequently or whether there are certain conditions or contexts under which RR attributes become a bottleneck a broader analysis is required.

3. ANALYSIS OF RELEASE READINESS

To learn from the analysis of release readiness of individual projects in a systematic manner, we developed a method consisting of the following phases:

- **Phase 1 – Set-up:**
  First one needs to have available a set of projects that will serve as the basis for learning so that potential bottlenecks in similar new projects can be anticipated and effort can be invested to proactively counteract. If projects are of different nature, e.g., from different domains, it makes sense to look at the various domains separately. Then the RR attributes that are believed to influence release readiness as well as the relevant observation period have to be set. RR attributes and observation period must be chosen in such a way that all selected projects can provide the required data for each RR attribute over the whole observation period.

- **Phase 2 – Data collection and pre-processing:**
  For each selected project the required raw data per RR attribute must be collected. Then for each RR attribute in each project the number of occurrences the RR attribute became a bottleneck must be calculated as described in Section 2.

- **Phase 3 – Cross-project analysis of bottleneck attributes:**
  Now, to identify (cluster-specific) patterns of bottleneck occurrence, per domain analyses can be conducted answering the following questions:

  (a) What are the most frequently occurring bottleneck attributes (overall and per cluster)?

  (b) What project characteristics influence the occurrence of bottleneck attributes (overall and per cluster)?

Example characteristics are:
- Project size, i.e., to distinguish between bottleneck attribute occurrence in large vs. small projects,
- Project team, i.e., to distinguish between bottleneck attribute occurrence in projects with many contributors of commits vs. those with few contributors, and
- Project phase, i.e., to distinguish between bottleneck attribute occurrence in early vs. late phases of a release development.

Of course, other criteria that may be more relevant for a certain context (company) could be defined. Once all data has been collected and processed, analyses can be conducted to answer the questions of interest. Based on the results, conclusions can be drawn about the bottleneck attributes that are most important to be monitored and counter-acted, per cluster, per project type, and per project stage.

4. LEARNING ACROSS PROJECTS FOR BOTTLENECK ANALYSIS

In order to initially validate our method for learning across projects and project clusters, we collected data for six RR attributes from selected OSS projects of two different domains over a period of 104 weeks (two years).

4.1 Set-Up

We identified the domains ‘web-based’ (W) and ‘desktop-based’ (D) projects, as these are project domains that are often relevant for companies. Table 2 shows per domain the 34 projects we selected (specifying id and name). Each project is characterized by
- Project id and name (C1),
- Number of commits (C2),
- Number of releases (C3),
- Number of different contributors (C4) and
- Project duration in calendar days (C5).

The data for characteristics was collected for the total lifespan of each project (i.e., from project start to observation time).

The next step in the Set-Up Phase is the identification of RR attributes. We selected six RR attributes as shown in Table 1. In order to cover both functional and non-functional aspects (release readiness dimensions), we were interested in assessing RR attributes that satisfy certain goals related to the implementation and test status in each project.
4.2 Data Collection and Pre-Processing
Once the domains, projects and RR attributes had been identified, we collected the raw data for each RR attribute. Then we analyzed the raw data and calculated for each RR attributes how often it happened to be a bottleneck.

Figure 3 shows the frequencies of bottleneck occurrences for all RR attributes as box plots. For each RR attribute, the ranges of frequencies are given for both D and W-type projects.

For example, we can see that the mean value for RR attribute PCR being a bottleneck across all projects over the whole period of time is above 70 for D-type projects and above 80 for W-type projects.

4.3 Cross-Project Analysis of Bottleneck Attributes
Once we knew for each project how often a RR attribute occurred as a bottleneck attribute we conducted further analyses. First we wanted to know whether some RR attributes occur more frequently than others and whether there is a difference between the rankings of RR attributes between the two domains we selected.

Figure 4 shows the Pareto charts for both domains. Clearly, the frequency patterns and rankings are very similar. In particular, in both domains, the same three RR attributes (PCR, FCR, and BFR) account for more than 80% of all bottleneck occurrences. In addition, the ranking of most frequent three RR attributes is equal in both domains. This implies that an engineer or manager can focus on controlling the top-most RR attribute(s) if resources are scarce.

Next, we were interested whether the occurrence frequencies of bottleneck attributes differ depending on certain project characteristics. In our case, we looked at three project characteristics, i.e., size (measured as # of commits), team dispersion (measured as # of different contributors), and release development phase. In order to determine the early and late phase of a release development of a project, we split each release development into two equal parts, early and late phase. For example, if 16 releases were observed for a project (cf. column # of Releases in Table 2), we have 16 early phases and 16 late phases and we can count how often a certain RR attribute happened to be a bottleneck attribute during each of these phases.

We applied non-parametric statistical testing (Mann-Whitney U test) to check whether frequency occurrences were significantly different per RR attributes in a given domain.

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1: Berkshelf/ Berkshelf</td>
<td>3609</td>
<td>97</td>
<td>81</td>
<td>812</td>
<td></td>
</tr>
<tr>
<td>D2: Ranyak/ Can can</td>
<td>419</td>
<td>29</td>
<td>62</td>
<td>1710</td>
<td></td>
</tr>
<tr>
<td>D3: Celluloid/ Celluloid</td>
<td>1415</td>
<td>45</td>
<td>74</td>
<td>1169</td>
<td></td>
</tr>
<tr>
<td>D4: Clinton-hall/ NzbToMedia</td>
<td>1472</td>
<td>0</td>
<td>15</td>
<td>581</td>
<td></td>
</tr>
<tr>
<td>D5: Fastly/ Epoch</td>
<td>345</td>
<td>14</td>
<td>6</td>
<td>385</td>
<td></td>
</tr>
<tr>
<td>D6: GoldenCheetah/ GoldenCheetah</td>
<td>4016</td>
<td>22</td>
<td>41</td>
<td>1319</td>
<td></td>
</tr>
<tr>
<td>D7: Grafana/ Ggrafana</td>
<td>1541</td>
<td>16</td>
<td>70</td>
<td>188</td>
<td></td>
</tr>
<tr>
<td>D8: Intridea/ Grape</td>
<td>1802</td>
<td>25</td>
<td>151</td>
<td>1450</td>
<td></td>
</tr>
<tr>
<td>D9: Joyei711/ Phyloseq</td>
<td>581</td>
<td>0</td>
<td>5</td>
<td>1046</td>
<td></td>
</tr>
<tr>
<td>D10: Mybb/ Mybb</td>
<td>1234</td>
<td>37</td>
<td>24</td>
<td>760</td>
<td></td>
</tr>
<tr>
<td>D11: Orientecnetologies/ Orientdb</td>
<td>7731</td>
<td>21</td>
<td>49</td>
<td>590</td>
<td></td>
</tr>
<tr>
<td>D12: Owncloud/ Mirall</td>
<td>5865</td>
<td>43</td>
<td>42</td>
<td>681</td>
<td></td>
</tr>
<tr>
<td>D13: Python-pillow/ Pillow</td>
<td>2336</td>
<td>15</td>
<td>95</td>
<td>729</td>
<td></td>
</tr>
<tr>
<td>D14: Resque/ Resque</td>
<td>1910</td>
<td>70</td>
<td>229</td>
<td>1723</td>
<td></td>
</tr>
<tr>
<td>D15: Scikit-learn/ Scikit-learn</td>
<td>16816</td>
<td>58</td>
<td>282</td>
<td>1422</td>
<td></td>
</tr>
<tr>
<td>D16: SynoCommunity/ Spksrc</td>
<td>1754</td>
<td>0</td>
<td>41</td>
<td>1011</td>
<td></td>
</tr>
<tr>
<td>D17: Zfsolinux/ Zfs</td>
<td>1408</td>
<td>30</td>
<td>104</td>
<td>1531</td>
<td></td>
</tr>
<tr>
<td>W1: Adobe/ Adobe</td>
<td>13887</td>
<td>53</td>
<td>225</td>
<td>958</td>
<td></td>
</tr>
<tr>
<td>W2: Att/ Reloud</td>
<td>2842</td>
<td>12</td>
<td>11</td>
<td>712</td>
<td></td>
</tr>
<tr>
<td>W3: Automattic/ Socket.io</td>
<td>1293</td>
<td>89</td>
<td>68</td>
<td>1589</td>
<td></td>
</tr>
<tr>
<td>W4: Locomotivecems/ Engine</td>
<td>2209</td>
<td>36</td>
<td>80</td>
<td>1429</td>
<td></td>
</tr>
<tr>
<td>W5: FortAwesome/ Font-Awesome</td>
<td>573</td>
<td>14</td>
<td>28</td>
<td>869</td>
<td></td>
</tr>
<tr>
<td>W6: Gravitystorm/ Openstreetmap-carto</td>
<td>595</td>
<td>29</td>
<td>29</td>
<td>598</td>
<td></td>
</tr>
<tr>
<td>W7: HSbhtml5/ Boilerplate</td>
<td>1340</td>
<td>24</td>
<td>175</td>
<td>1641</td>
<td></td>
</tr>
<tr>
<td>W8: Hawtio/ Hawtio</td>
<td>5920</td>
<td>51</td>
<td>45</td>
<td>594</td>
<td></td>
</tr>
<tr>
<td>W9: Highslide-software/ Highcharts.com</td>
<td>4109</td>
<td>71</td>
<td>31</td>
<td>1498</td>
<td></td>
</tr>
<tr>
<td>W10: Hypothesis/ H</td>
<td>3851</td>
<td>9</td>
<td>18</td>
<td>831</td>
<td></td>
</tr>
<tr>
<td>W11: Jashkenas/ Backbone</td>
<td>2629</td>
<td>21</td>
<td>228</td>
<td>1379</td>
<td></td>
</tr>
<tr>
<td>W12: MayhemYG/ 4chan-x</td>
<td>5151</td>
<td>192</td>
<td>35</td>
<td>1017</td>
<td></td>
</tr>
<tr>
<td>W13: Mbostock/ D3</td>
<td>3207</td>
<td>173</td>
<td>78</td>
<td>1393</td>
<td></td>
</tr>
<tr>
<td>W14: Moment/ Moment</td>
<td>2050</td>
<td>36</td>
<td>204</td>
<td>1160</td>
<td></td>
</tr>
<tr>
<td>W15: Imathis/ Octopress</td>
<td>808</td>
<td>1</td>
<td>103</td>
<td>1683</td>
<td></td>
</tr>
<tr>
<td>W16: Travis-ci/ Travis-ci</td>
<td>3602</td>
<td>232</td>
<td>94</td>
<td>1241</td>
<td></td>
</tr>
<tr>
<td>W17: Webbuikki/ Dynmap</td>
<td>1738</td>
<td>67</td>
<td>14</td>
<td>1295</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5 shows per domain the split of occurrence frequencies for all six RR attributes regarding size, team dispersion, and release development phase. The symbol ‘***’ next to the RR attribute name indicates that the occurrence frequency of the respective attribute is significantly different at an alpha-level of 5%.

We found three different patterns: 1) Distinguishing projects with regards to size does not show any significant difference in the occurrence frequency of bottleneck attributes in both domains. 2)
Distinguishing projects with regards to phase does always show significant difference in the occurrence frequency of bottleneck attributes for all RR attributes in both domains. 3) Distinguishing projects with regards to team does not show any significant difference for desktop-based projects but does show significant difference for RR attribute PCR for web-based projects. In other words, it makes a difference whether one monitors the occurrence frequency of bottleneck attributes for different domains and different project related criteria.

In our example, all RR attributes show an almost uniform behavior depending on domain and characteristic. This might not always be the case. Then it might be most interesting to focus on the RR attribute that occurs most frequently as a bottleneck overall, i.e., PCR.

The benefit from doing such a kind of analysis is that a company with large portfolios of projects (and related repositories) can use this kind of information to focus their effort spent on monitoring RR attributes where it really matters.

5. APPLICABILITY OF FINDINGS

Based on the key findings analyzed in the previous chapter, the following results can potentially be further studied and applied in specific projects, by both engineers working in industry and scientists at research institutions alike:

i) a method to identify and monitor bottleneck attributes for individual project development and release decisions

ii) a method to uncover universal properties of bottleneck attributes (i.e. the Pareto-charts);

iii) a method to find context-specific properties of bottleneck attributes (those that may vary from company to company or domain to domain).

Early identification of bottleneck attributes is expected to help release engineers in proactively addressing potential resources limitations and recommend the course of action. However, this information is not easily available. The proposed method introduces a systematic learning approach from self/external experience of former releases. As a validation approach, we retrospectively investigated 34 OSS projects over a two year period with respect to six industry proven essential RR attributes.

Findings of our investigation can be applied in release engineering in meaningful ways to:

i) Identify influential bottleneck attributes that could cause issues in different phases of the releases,

ii) Pro-actively utilize information regarding variation in bottleneck attributes due to project characteristics,

iii) Guide project teams to better allocate potential resources conflicts based on volatility of the bottleneck attributes.

Due to unavailability of proprietary software, we validated our method based on OSS projects. OSS represents the real world software development and are accepted in both academia and industry as a good source for software investigation. In consideration of 104 weeks of observation on 34 projects, the investigation results are expected to be useful in real world proprietary projects as well. Universal rules and claims cannot be made based on our study. The study is meant to be a validation of how industry organizations and projects can systematically learn from related project experiences. Selection of project domains can be further classified to sub-categories to increase opportunities for more generalized results. However, considered domains are most prominent in software industry. RR attribute selection is dependent on organization goal and user demands. This is a highly context-specific decision. We selected a comprehensive set of RR attributes based on industry guidelines. Corresponding metrics also followed industry standard. These attempts make investigation results readily useful for industry projects.
Figure 5: Occurrence frequency of bottleneck attributes by project size (top), project team (middle), project phase (bottom) in desktop-based (left) and web-based projects (right).

6. CONCLUSIONS AND FUTURE WORK

Release management is a decision-centric process with a number of criteria, stakeholders and constraints involved. The impact of releasing a product too early or too late can be catastrophic. In the trade-off between quality, release time and delivery of functionality as requested by customers, ad-hoc decisions potentially cause significant risks to projects and organizations. With special emphasis on release readiness, we propose an analytical approach to monitor, analyze and improve release readiness. Even though some of the required data are uncertain and challenging to acquire, the results presented in this paper indicate that there are substantial differences in the occurrence of bottleneck attributes in achieving release readiness. The differences are between the attributes that have been studied in this paper, as well as between two domains where OSS projects studied were taken from.

We see the proposed work as a starting point to establish release decisions on objective data and analytical precision. As a form of decision support, the full value of the analytical result can only be achieved from a proper process of data collection and analysis,
combined with the proper involvement of the release engineers. The proposed method needs further analysis and evaluation of its applicability and usefulness. There are more advance models to integrate multiple attributes such as the Ordered Weighted Averaging (OWA) aggregation operator introduced by Yager [10]. In addition, the importance of the local RR attributes is changing in the course of a release, with emphasis on testing towards the end [7], which needs to be considered as well. Practical guidelines are needed in terms of the requested attribute performances to facilitate release delivery in-time and in quality. Instead of just counting the frequencies of occurrence, future work will also include the estimates on effort needed to reduce the gap between expected and actual performance.

7. REFERENCES