Chapter 1
Monitoring and Controlling Release Readiness by Learning across Projects

S M Didar Al Alam, Dietmar Pfahl, Guenther Ruhe

1.1 Introduction

In the competitive commercial software market, software companies strive to incorporate business and technology changes. Industrial innovation and ability to confront emerging technologies help companies to sustain a competitive advantage. Software in operational use is continuously evolving due to new changes and requirements. To accommodate rapid changes, companies require improvement in software processes. Project managers demand processes that support fast results and flexible feature delivery. The ultimate goal is to achieve competitiveness and business success. The key to survive is to develop and market quality products, on time, and within budget [2]. Companies feel compelled to quickly adapt and change the ways they develop products and services and release software the moment it is ready.

Software releasing is the process of delivering the product into the operational environment for usage by its end users [15]. A slip in release causes millions of dollars in lost revenue. On the other hand, if delivered early, these projects display excessive error quantities and low levels of reliability. A product, or a major version of the product, can only be released when it is ready. The decision dilemma project managers often face is whether to prioritize further functionality or further test/re-work/ process improvement, popularly known as the "stopping rule" problem [9]. This decision cannot be made ad hoc, but rather needs analytical evidence. Release readiness (RR) can provide this evidence and facilitate release decisions. RR is a time-dependent attribute of the product release. It aggregates a portfolio of release process and product measures to quantify status of the release. 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 book chapter is a part of an ongoing effort towards achieving improved RR through monitoring and analysis of RR and software process evolution. It presents the state-of-the art concepts related to software RR (e.g. existing RR approaches, metrics, tools, empirical evaluation). Emphasis is given to the approaches of RR evaluation. Based on relevant studies a comprehensive systematic mapping study is performed and reported. Comprehensive analysis of related works reveals existing gaps, illustrates the motivation of our research and our contributions towards the body of knowledge. This chapter also discusses identifying, monitoring and analyzing attributes limiting release readiness (called bottleneck factors or BFs). Our goal in this chapter is to understand frequencies and pattern of occurrence of attributes affecting project success by restricting the status of release readiness, at different stages of release cycle.

Our former explorative case study research [1]on open source software (OSS) projects provided an initial understanding of the frequent bottleneck factors for release readiness and their likelihood of subsequent occurrence. It revealed the importance of systematically studying and analyzing bottleneck factors across individual projects to understand common patterns and their frequency of occurrence. To achieve this goal, we propose a five-phased method for monitoring and controlling RR by learning BFs across projects. It can be applied for any ongoing product release development project. Initial validation of proposed method is performed based on 34 open source projects from the GitHub repository. The projects were taken from two domains, i.e., desktop-based and web-based projects. We monitored the performance of six established release readiness attributes, and primarily focused on the lowest performing attributes i.e. bottleneck factors.

Early identification of bottleneck factors and understanding their common patterns and frequency helps release engineers in proactively addressing potential resource limitations and initiate process changes. As a form of learning across projects, we are interested in answering the following questions:

- How to monitor and analyze bottleneck factors limiting software release readiness?
- Are there any common patterns in the occurrence frequency of bottleneck factors?
- Do occurrence frequencies of BFs vary across domains with regards to project-characteristics such as (i) project size, (ii) number of contributors, (iii) release development phase?

Proposed method also combines the strengths of analytical methods with the intuition of human experts. Inspired by the idea of software process control, the approach assumes continuous monitoring of the bottleneck factors identified in the learning process. The actual performance is compared with the planned one. In case of above-threshold deviation, an out-of-control situation is identified and handled by human experts. Learning BFs and their characteristics facilitate handling out-of-control situations. The rest of the chapter

is divided in four major sections. Section 2 presents the background and literature review. It also reports results from a semi-systematic mapping performed on related works. In Section 3, methodology of the proposed method is illustrated. Section 4 empirically validates the proposed method with respect to open source software. Section 5 presents summary of the chapter and future research.

1.2 Background and Related Works

1.2.1 Background

1.2.1.1 Software product management

Software Product Management, is a key success factor for companies [19] that facilitates timely production and faster product acceptance in market [8]. Authors in [6] defined software product management as

> "the discipline and role, which governs the software product (or solution or service) from its inception to the market/customer delivery in order to generate biggest possible value to the business".

In software product management practices, a product manager is responsible to decide product release content, timeframe, price and implements the business case in consideration of technical aspects [7].

1.2.1.2 Software release in iterative software development

Iterative software development is one of the most widely adopted techniques in software development. It supports incremental software development in small iterative cycles and allows incorporation of stakeholders' feedbacks in ongoing development cycle. In iterative context, software release consists of multiple iteration and delivers the product into the operational environment for usage by the end users [12]. Individual iteration might have iteration releases, where a partially completed, stable version is released internally [11].

1.2.1.3 Release readiness attributes and Degree of satisfaction

Release Readiness attributes are attributes of the candidate system that can define and judge the RR of the system. Satisfaction of Defect find rate (DFR), and Bug fix rate (BFR) are two examples of RR attributes. Approaches like Goal-Question-Metric (GQM) [4]paradigm can be applied to guide the

selection of RR attributes. Degree of satisfaction refers to project manager's satisfaction towards the performance of an RR attribute. In [21][12], the degree of satisfaction of the RR attributes is evaluated using the concept of membership function from fuzzy set theory [25]. A membership function μ_F (x) quantifies the degree of membership of element x in a fuzzy set F. The project manager select appropriate shapes and corresponding parameters of the membership function associated with each individual RR attributes to evaluate the degree of membership (i.e. the degree of satisfaction) based on its value. Further details can be found in [21].

1.2.1.4 Release readiness

At any point in time during a release cycle [0,T], the measurement of RR attributes helps project managers assess the status of the next release. RR is defined as the aggregation of various RR attributes that are considered to be essential to judge whether a product release is ready for shipping. Project managers based on successful legacy release and personal experience provide the relative weights of RR attributes. Further details are discussed in later sections and also in [21].

1.2.1.5 Bottleneck factors

A bottleneck in general is a factor that limits the performance of an entire system. Resource bottlenecks in project management are a well-understood phenomenon. We transfer this idea from project management to the study of RR. For a given project bottleneck factor refers to the RR attributes that is lowest satisfied and thus limits RR.

1.2.2 Related Works

Former literature [22, 17] considered defect tracking and test related metrics, e.g., number of defects, defect removal rate, test execution rate, test pass rate etc. in RR evaluation. Wild et al. [24] proposed to consider metrics from multiple dimensions (e.g., requirements, functionality, reliability etc.). Industry tools, e.g. Borland Team Inspector and PTC Integrity , visualize and verify functionality, code and test related metrics before releasing a piece of software. Most of these approaches concentrated on evaluating RR at the end of the release. Continuous awareness of the status of the product release is important [21] to ensure project success. To accurately model the release process and reveal RR, multiple attributes and their underlying relationship should be considered [10]. Significant uncertainties are associated with release

process [15]. Without considering these uncertainties and adapting towards changes, RR evaluation can be misleading and critically risky. While analyzing our related works in a semi-systematic mapping, we focused on these issues. From a significant number of related publications, we identified 22 papers most relevant to our research and investigated following questions:

- Q1: What are the types of approaches proposed for determining RR over time?
- Q2: What are the types of contributions offered in determining RR over time?
- Q3: What are the RR attributes and metrics selected in RR evaluation over time?

1.2.2.1 Q1: What are the types of approaches proposed for determining RR over time

In literature and industry, RR is determined in different ways. Broadly, they can be categorized into four categories as discussed below. Fig. 1.1, presents a mapping of related works with respect to type of approaches proposed and year of publication.

Checklist based approach: These approaches check a set of RR criteria at the end of release cycles before releasing a software [19, 13]. They extensively rely on subjective questions that are hard to judge. In addition, these approaches do not support any proactive analysis of RR. Around 22% related works fall under this category

Testing metrics based approach: These approaches [12, 14, 23] consider testing related metrics (e.g. test passing rate, defect find rate) and build various RR indicators. For example, authors in [22] calculated time to release based on defect data, authors in [14] compared objective metrics with past project data using spider chart to indicate RR. These approaches exclusively focus on the testing phase and cannot guarantee continuous monitoring. Around 43% of related articles are under this category

Defect prediction model based approach: These approaches assume that remaining defects in software is the major indicator of the readiness. This approach primarily focuses on creating a prediction model for remaining defects. Multiple techniques e.g. Neural Network [18], tracing code changes[24] are applied to identify remaining defects. Due to exclusive focus on remaining defects only, these approaches measure RR partially, while leaving other RR influencing dimensions uncaptured. 22% of related works applied approaches of this category.

Multi-dimensional metrics aggregation based approach: These approaches evaluate a portfolio of product, process related metrics and aggregate them into a single measure of release readiness. We found three studies [21, 3, 20] applying this approach. These approaches provide a broad overview

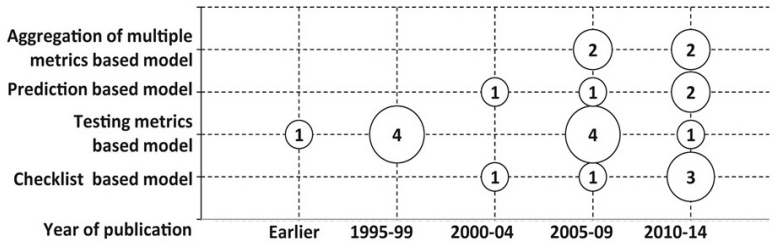


Fig. 1.1: Mapping related works with respect to type of approach applied and year of publication

of RR and enables proactive analysis of RR. However, adhoc selection of metrics may generate misleading RR evaluation.

1.2.2.2 Q2: What type of contribution these approaches focused over time

In Q2, we analyzed the type of contributions proposed over time in related works. Fig 1.2, denoted contributions using keywords certification, measurement, evaluation, prediction and recommendation. Certification means consideration of individual metrics and comparison with their expected values to certify their individual status e.g. checklist based approaches [19, 12]. Approximately 30% of related works certified metrics to evaluate RR. Measurement means isolated evaluation of multiple metrics. For example, measuring defect free hours before a release [5]. In this approach, metrics are not aggregated but measured in isolation to take the release decision. Evaluation means evaluation of multiple metrics along with aggregating them into a meaningful single measure, which reflects the status of release readiness. 30% of related works applied this approach. In [16], author built a neural network to estimate defects and predict release readiness. Authors in [21] applied time series extrapolation techniques to predict future RR. In all these approaches individual attributes are measured and then compared or aggregated to identify RR. However, the selection of attributes is crucial, while mostly performed adhoc. This chapter will address this problem in next sections.

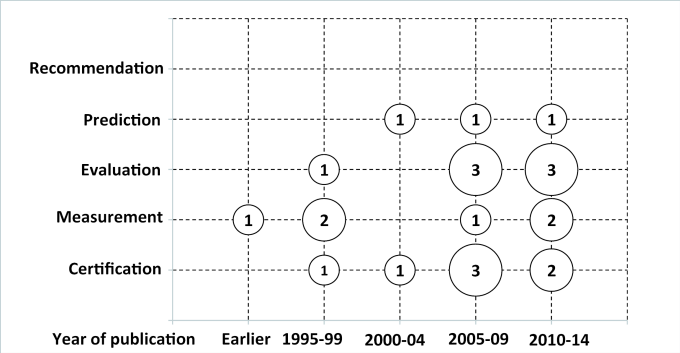


Fig. 1.2: Mapping related works with respect to type of contribution and year of publication

1.2.2.3 Q3: What are the RR attributes and metrics selected in RR evaluation over time

By reviewing recent literature and industry articles, a significant number of RR attributes are extracted. Broadly all these can be categorized into four RR dimensions that includes implementation status, testing scope and status, source code quality and documentation scope and status. Example of RR attributes related to each dimensions are presented in Table 1.1 below.

Table 1.1: RR dimensions and related RR attributes

RR dimensions	Overview of related RR attributes
Implementation status	Attributes related to feature implementation, change request implementation, coding effort, continuous integration, build trends, etc.
Testing scope and status	Attributes related to defect finding, defect fixing, test coverage, test effort, etc.
Source code quality	Attributes related to code review, coding style, code smells, refactoring, code complexity, etc.
Documentation scope and status	Attributes related to user manual, design documents, test specification, test case documentation, etc.

In our third investigation, we focused towards distribution of these RR attributes in related works over time. We extracted the list of attributes from each paper and categorized them in four categories stated earlier. Fig.1.3 presents a mapping of related studies with respect to RR metrics selected

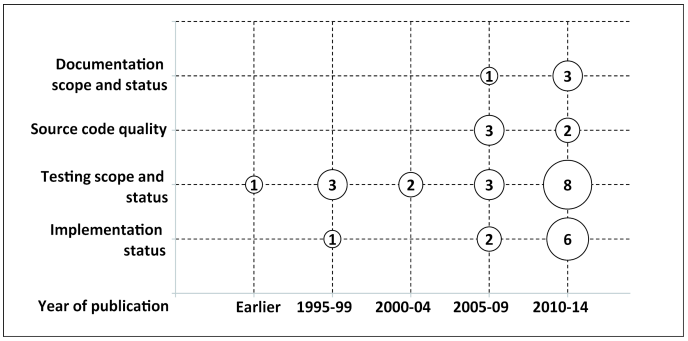


Fig. 1.3: Mapping related works with respect to RR dimensions addressed and year of publication.

and publication years. From this mapping, the importance of Implementation status and Testing scope and status dimensions becomes obvious compared to other dimensions. Therefore, in our research, we narrow our scope by considering these two RR dimensions only.

1.3 Methodology

In our former publication [1], we identified evidence for existence of common patterns in BF occurrences. It is worthwhile to study these patterns in understanding bottleneck factors. In this section, we propose a five-phased approach, where each phase presents guidelines to systematically identify and analyze BFs through individual projects, learning across this analysis to facilitate monitoring and controlling process changes. The approach is briefly discussed below.

1.3.1 Phase 1 – Selection of projects and RR attributes

A set of projects will serve as the basis for learning. This facilitates anticipating potential bottlenecks in similar new projects and investing effort to proactively counteract. If projects are of different nature, e.g., from different domains, it makes sense to look at the various domains separately. RR attributes that are believed to influence release readiness as well as the relevant observation period have to be set. Selection of attributes considered for deciding about release readiness varies among projects and companies. The

ultimate decision of RR attributes selection is upon the user. It is typically based on the organizational goals and customer expectations. 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.

For each selected project, the required raw data per RR attribute must be collected. As identified earlier in (see Table 1.1) there are four major dimensions for RR attributes. Each dimension is further described by a set of possible RR attributes. These are different aspects of software product by which release readiness can be evaluated. Each attribute is quantitatively described by one or more metrics. For example, the attribute satisfaction of feature implementation at week k can be measured (on a weekly base) by a metric called feature completion rate (FCR) defined as follows in Eq. (1.1):

$$FCR\,(k) = \frac{(num\,of\,features\,implemented\,up\,to\,week\,k)}{(num\,of\,features\,requested\,up\,to\,week\,k)} \tag{1.1}$$

As FCR only refers to the completed features, this metric might not measure the progress of implementation most accurately, however it is good enough to support an analytical approach for monitoring, controlling the release status and improve the process and end product.

Approaches like Goal-Question-Metric (GQM) paradigm is effective to design an measurement program that fulfills the goal of overall RR evaluation. RR attributes corresponding to questions further refine the measurement goal. Available data associated with each question quantitatively answers them. For simplicity we scope our self within two major RR dimensions i.e. implementation status and testing scope and status. Importance of these two dimensions in RR evaluation already revealed in Section 2. We selected six RR attributes from these two dimensions using the GQM approach as listed in Table 1.2. The specific attributes and measures taken in this example are highly context specific. They represent 50% of RR attributes known from comprehensive industry guidelines available

1.3.2 Phase 2- Identifying Local and Global release readiness

For each individual RR attribute, initially local RR metric is calculated. Subsequently, local RR metrics are combined into a global RR metric representing release readiness of the product. At any point in time t=t^* for any of the selected release readiness attributes, the local release readiness (LRR) status is a degree of satisfaction of individual attributes while applying piecewise linear membership function. LRR is defined as the deviation from plan in

Table 1.2: Defining RR metrics using the top-down GQM approach

RR dimensions	Attributes	Questions	Metric Definitions	Acronyms
Implementation status	Status of feature implementation	To what extent feature requests are completed	# of features implemented up to week k / # of features requested up to week k	FCR
Implementation status	Status of continuous integration	To what extent continuous integration (CI) requests are completed	# of CI requests completed in kth week/ # of CI requests completed up to week k	PCR
Implementation status	Status of improvement completion	To what extent improvement requests are completed	# of improvements implemented up to week k / # of improvements requested up to week k	ICR
Testing status	Status of defect finding	To what extent the testing activity reducing defects	# of defects found in kth week / # of defects found up to week k	DFR
Testing status	Status of bug fixing	To what extent detected bugs are fixed	# of bugs solved up to week k / # of bugs identified up to week k	BFR
Testing status	Status of source code stability	To what extent the source code is becoming stable	# of code churn in kth week / # of code churn up to week k	CCR

terms of its local RR metric. The measure is normalized to the [0,1] interval, where level 0 and 1 means that the degree of satisfaction expected at time t is null or 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 one release of duration 20 weeks in a project called P. For simplification, we consider only two RR attributes: (a) status of bug fixing, and (b) status of feature completion, defined as bellow

$$bug\,fix\,rate\,(k) = \frac{(num\,of\,bugs\,solved\,up\,to\,week\,k)}{(num\,of\,bugs\,identified\,up\,to\,week\,k)} \tag{1.2}$$

$$feature\,completion\,rate\,(k) = \frac{(num\,of\,features\,implemented\,up\,to\,week\,k)}{(num\,of\,features\,requested\,up\,to\,week\,k)} \tag{1.3}$$

These are considered as objective metrics, which can be used to evaluate the RR attributes. We measure the local RR of bug fix rate (BFR) & feature completion rate (FCR) (on a weekly time interval) following we have a planned RR performance based on experience from successful former releases.

For example, for BFR the minimum and maximum expectation are 1 and 10 bugs fixed per day respectively. In that case local RR of BFR for any value x is calculated following the Eq. (1.4) below.

$$LRR(x) = \begin{cases} 0 & x < 1 \\ \frac{x-1}{10-1} & 1 \leq x \leq 10 \\ 1 & x > 10 \end{cases} \tag{1.4}$$

Definition (Local RR): We assume that project P with duration $[0, T]$ at given week $t = t_0 \in [0, T]$ have

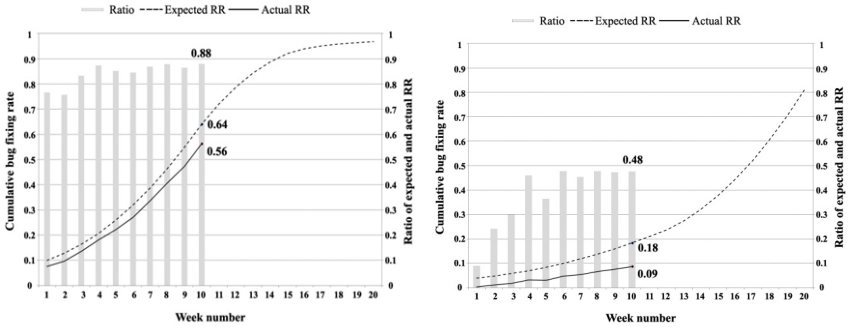
- a given set of RR attributes $A = \{a_1, a_2, ..., a_n\}$;
- for each RR attribute a corresponding minimum and maximum expected set of values are $A_{min}(a_i, t)$ and $A_{max}(a_i, t)$
- corresponding actual values of RR attributes given by the n-dimensional vector $A_{actual}(a_i, t)$
- Then, $LRR(a_i, t_0) \in [0, 1]$ is the local release readiness of attribute $a_i$at week $t = t_0$. It is calculated based on the corresponding value in vector $A_{actual}(a_i, t)$and expected values of $A_{min}(a_i, t)$ and $A_{max}(a_i, t)$ following Eq. (1.5)

$$LRR(a_i, t_0) = \frac{A_{actual}(a_i, t) - A_{min}(a_i, t)}{A_{max}(a_i, t) - A_{min}(a_i, t)} \tag{1.5}$$

As illustrated in Fig. 1.4, the planned (dotted line) rate at week t=10 is higher (0.64) than the computed actual (solid line) rate (0.56). Here, BFR 0.64 represents fully achieved Local RR (i.e. 1) and BFR 0 represents not achieved Local RR (i.e. 0). Bars represent the relative performance degree, which is 0.88 for week 10. It means we achieved a local RR of 0.88 for BFR. Similarly, we measure the local RR of FCR (on a weekly time interval) as defined earlier. The performance measure indicates that at week 10, the actual performance is only 48% of the planned performance, which represents local RR of 0.48 for FCR.

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.

Definition (Global RR): For a given set of RR attributes $A = \{a_1, a_2, ..., a_n\}$; the local RR of any attribute $a_i$at week $t = t_0 \in [0, T]$ is represented by $LRR(a_i, t_0)$, where $LRR(a_i, t_0) \in [0, 1]$. If $\{w_1, w_2, ..., w_n\}$represents corresponding weights of attributes, the global RR is calculated as follows:



(a) For bug fixing rate (b) For feature completion rate

Fig. 1.4: Local RR evaluaiton

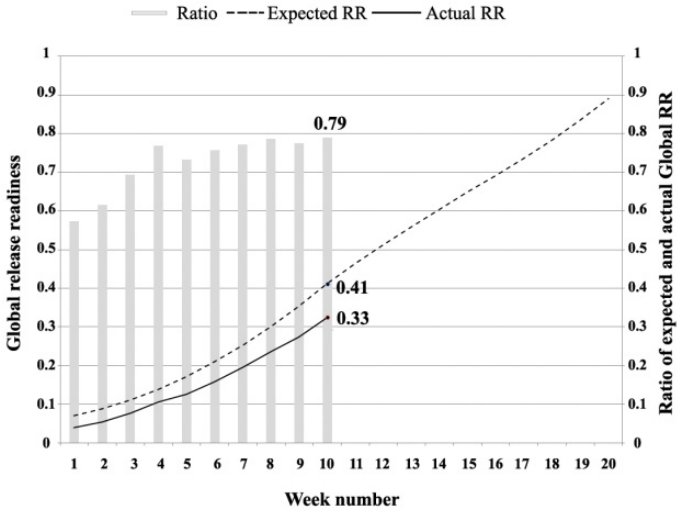


Fig. 1.5: Global Release Readiness evaluation

$$GRR(t_0) = WAM(LRR(a_1, t_0), LRR(a_2, t_0), ..., LRR(a_n, t_0)) = \sum_{i=1}^{n} w_i \times \times LRR(a_i, t_0) \tag{1.6}$$

For the above example, both RR attributes are considered equally weighted. Therefore, the global RR GRR (10) is calculated as the arithmetic mean of local RR value of LRR(BFR,10) and LRR (FCR,10) which is 0.33 and illustrated in Fig. 1.5 below.

1.3.3 Phase 3 – Identifying bottleneck factors

Our selection of attributes and subsequent metrics for this chapter was primarily influenced by their (i) acceptance in real world, (ii) availability of corresponding data in the given repository and (iii) Ease of calculation. Table 1.2 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 RR metrics. For each RR attribute in each project the number of occurrences the RR attribute became a bottleneck must be calculated as described earlier. As defined earlier Bottleneck factor (BF) limits the performance of the entire system. We define BF as follows:

Definition (Bottleneck factor): For a given project P and a given week $t = t_0 \in [0, T]$, the bottleneck factor $BF(t_0)$ is a RR attribute a_i that has the lowest Local RR value $LRR(a_i, t_0)$ among all RR attributes and thus limits global RR (Eq. (1.6)) value $GRR(t_0)$. at week $t = t_0$.

$$BF(t_0) = arg\,min_i(LRR(a_1, t_0), LRR(a_2, t_0), ..., LRR(a_n, t_0)) \quad (1.7)$$

Based on Eq. (1.7), we calculate BF for our project. Eq. (1.7) returns the index of the bottleneck factor. In our example, FCR represents the bottleneck at week 10, as its relative performance is smaller than that of BFR Multiple RR attributes can become BF in a single observation. To avoid bias in BF identification, we applied equal weights for RR attributes. Therefore, WAM from Eq. (1.6) becomes equivalent to arithmetic average, which allows a simpler BF identification approach using Eq. (1.7).

In order to better understand which RR attributes become bottleneck factors more or less frequently or whether there are certain conditions or contexts under which RR attributes become a bottleneck, we plan to compare the bottleneck frequency of each RR attributes

Definition (Bottleneck Frequency): For a given project P and a given time interval $[0, T]$, the bottleneck frequency of a RR attribute a_i (denoted by $BNF(a_i, t_0)$) is defined as the weekly frequency of $a_i$becoming bottleneck in achieving RR within T.

1.3.4 Phase 4 – Cross-project analysis of bottleneck attributes

To identify (cluster-specific) patterns of bottleneck occurrence, per domain analyses can be conducted answering the questions: i) What are the most frequently occurring bottleneck attributes (overall and per cluster)? ii) What project characteristics influence the occurrence of bottleneck attributes (overall and per cluster)? Example characteristics considered 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,
- Project phase, i.e., to distinguish between bottleneck attribute occurrence in early vs. late phases of a release development

Of course, other characteristics can become more relevant for a certain context (company), and should be defined and analyzed. 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.

1.3.5 Phase 5-Monitoring and controlling the process:

Controlling and evolution of process is supported by continuously monitoring actual plan implementation process. Based on accumulated knowledge of bottleneck factors and their characteristics, corresponding metrics are identified for continuous monitoring. For example, if BFR and FCR are identified for continuous monitoring, $\triangle BFR(t)$, $\triangle FCR(t)$ refer to the deviation between actual process and predicted process performance related to bug fix rate and feature completion rate, respectively at any time t. As soon as one of the factors indicates an out-of-control situation, required changes are initiated in process towards controlling the situation based on human expertise and analytical guidelines. 'Out-of-control' situations are considered those situations that fall outside the expected/planned values (after considering common variation) and indicate existence of an assignable cause of variation.

The release plan represents a schedule and an assignment to features for the development process of the current release period $[0, T]$. Process monitoring is performed related to the feature completion rate. A continuous monitoring process checks whether the LRR of FCR is above the tolerance level β_1, β_2. Otherwise, an alarm for Out-of-control situation is triggered at $t = t*$when

$$\triangle LRR(FCR, t^*) < \beta_1 \tag{1.8}$$

Process monitoring is performed related to the bug fixing rate. If a reliable initial defect estimation measure exists and bug fixing is monitored continuously, an Out-of-control situation is triggered at time $t = t*$ if LRR of BFR is below tolerance level β_2

$$\triangle LRR(BFR, t^*) < \beta_2 \tag{1.9}$$

Out-of-control situations are identified with respect to bottleneck factors identified earlier. Learning these BFs across projects provide guidelines regarding their characteristics in different environment and help project managers to control these factors through process evolution.

1.4 Empirical validation

1.4.1 Setup

To validate our proposed approach, we collected data for six RR attributes (see Table 1.2) from a set of selected 34 different OSS projects of two different domains over a period of 104 weeks (two years). We selected projects from two domains i.e. 'web-based' (W) and 'desktop-based' (D), which are often relevant for companies. Table 3 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). We selected six RR attributes (shown in Table 1.2) based on comprehensive industry guidelines. 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.

1.4.2 Data collection and pre-processing

Once the domains, projects and RR attributes are identified, we collected the raw data for each RR attribute. We analyzed the raw data and calculated for each RR attributes how often it happened to be a bottleneck. Fig. 1.6 shows the frequencies of bottleneck occurrences (i.e. bottleneck frequency) for all RR attributes as box plots.

	C1	C2	C3	C4	C5
Desktop based projects	D1: Berkshelf/ Berkshelf	3609	97	81	812
	D2: Ryanb/ Can can	419	29	62	1710
	D3: Celluloid/ Celluloid	1415	45	74	1169
	D4: Clinton-hall/ NzbToMedia	1472	0	15	581
	D5: Fastly/ Epoch	345	14	6	385
	D6: GoldenCheetah/ GoldenCheetah	4016	22	41	1319
	D7: Grafana/ Ggrafana	1541	16	70	188
	D8: Intridea/ Grape	1802	25	151	1450
	D9: Joey711/ Phyloseq	581	0	5	1046
	D10: Mybb/ Mybb	1234	37	24	760
	D11: Orientechnologies/ Orientdb	7731	21	49	590
	D12: Owncloud/ Mirall	5865	43	42	681
	D13: Python-pillow/ Pillow	2336	15	95	729
	D14: Resque/ Resque	1910	70	229	1723
	D15: Scikit-learn/ Scikit-learn	16816	58	282	1422
	D16: SynoCommunity/ Spksrc	1754	0	41	1011
	D17: Zfsonlinux/ Zfs	1408	30	104	1531
Web Projects	W1: Adobe/ Adobe	13887	53	225	958
	W2: Att/ Rcloud	2842	12	11	712
	W3: Automattic/ Socket.io	1293	89	68	1589
	W4: Locomotivecms/ Engine	2209	36	80	1429
	W5: FortAwesome/ Font-Awesome	573	14	28	869
	W6: Gravitystorm/ Openstreetmap-carto	595	29	29	598
	W7: H5bphtml5/ Boilerplate	1340	24	175	1641
	W8: Hawtio/ Hawtio	5920	51	45	594
	W9: Highslide-software/ Highcharts.com	4109	71	31	1498
	W10: Hypothesis/ H	3851	9	18	831
	W11: Jashkenas/ Backbone	2629	21	228	1379
	W12: MayhemYDG/ 4chan-x	5151	192	35	1017
	W13: Mbostock/ D3	3207	173	78	1393
	W14: Moment/ Moment	2050	36	204	1160
	W15: Imathis/ Octopress	808	1	103	1683
	W16: Travis-ci/ Travis-ci	3602	232	94	1241
	W17: Webbukkit/ Dynmap	1738	67	14	1295

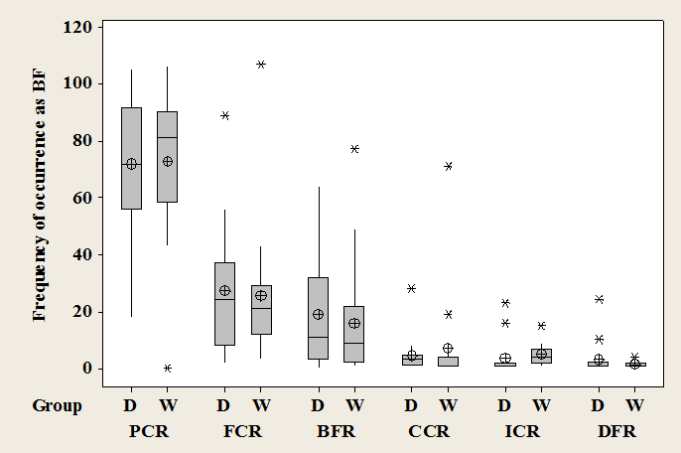


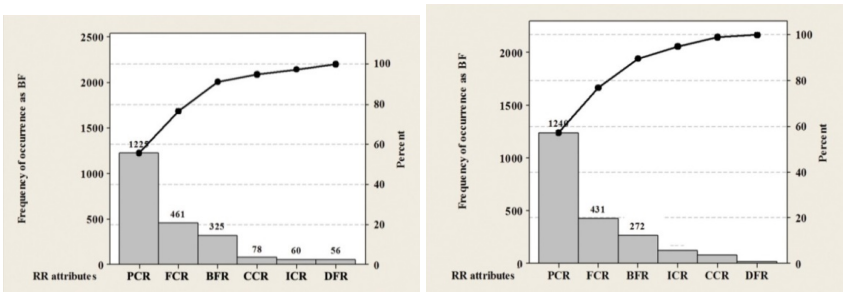
Fig. 1.6: Bottleneck frequency of all six RR attributes for both D and W-type projects

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 is above 70 for D-type projects and above 80 for W-type projects

1.4.3 Cross-project analysis of BF

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 common patterns of BN frequency i.e. whether some RR attributes occur more frequently than others do or whether there is a difference between the rankings of RR attributes between the two domains we selected. Fig. 1.7 shows the Pareto charts for both domains. Clearly, the bottleneck frequency patterns and rankings are very similar among domains. 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 bottleneck factors is similar 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



(a) Desktop-based projects (b) web-based projects

Fig. 1.7: Pareto charts for bottleneck factor occurrence frequencies

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 (see column # of Releases in Error! Reference source not found.), 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. Fig 1.8 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: i) Distinguishing projects with regards to size does not show any significant difference in the occurrence frequency of bottleneck attributes in both domains, ii) 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.

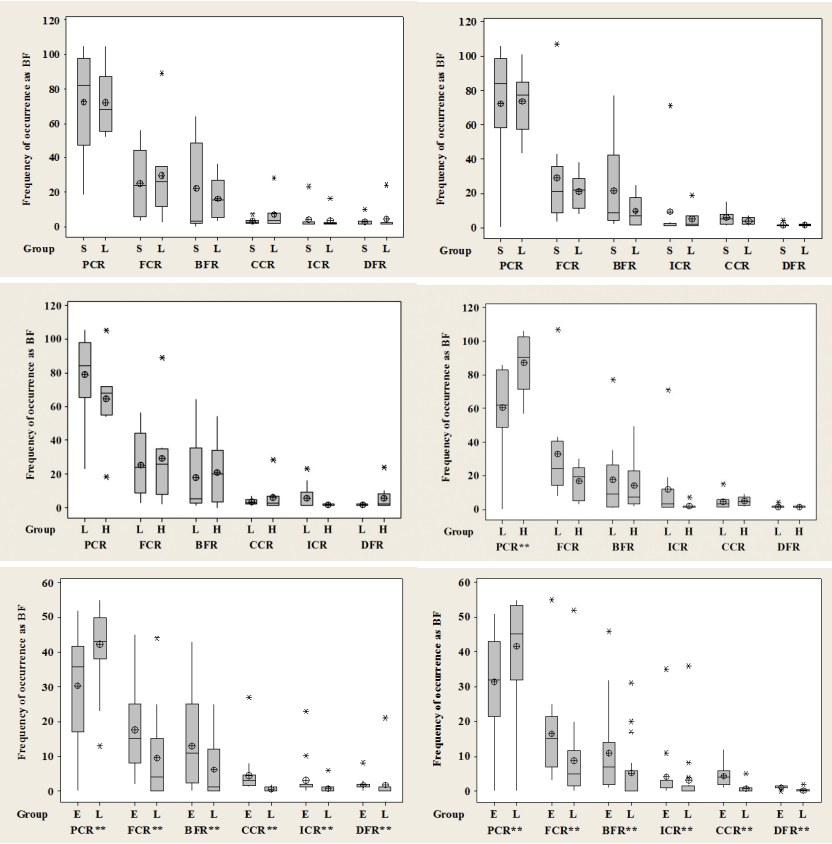


Fig. 1.8: Frequency of occurrence of bottleneck factors by project size (top), project team (middle), project phase (bottom) in desktop-based (left) and web-based (right) projects.

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.

1.4.4 Applicability of findings

Knowledge regarding bottleneck factors helps to identify RR attributes responsible for limiting RR. Early identification of bottleneck factors is ex-

pected 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. A comprehensive initial validation is conducted with respect to OSS projects. Proposed method and findings from initial investigation can be applied in RR evaluation and corresponding recommendation in meaningful ways:

- Identify influential bottleneck factors that can limit RR in different phases of the releases
- Pro-actively utilize information regarding project characteristics influence on variation of bottleneck factors
- Recommend project teams in better resource allocation based on volatility of the bottleneck factors

1.5 Summary and Future research

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 bottleneck factors and improve release readiness. Even though some of the required data are uncertain and challenging to acquire, the results presented in this chapter 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 chapter, as well as between two domains where OSS projects studied were taken from.

This study is considered as an initial phase of more comprehensive analysis with focus on evaluation and optimization of software release readiness. Due to unavailability of proprietary software, we validated our method based on OSS projects. We plan to broaden the project scope to proprietary projects and comparison of results with observations from other OSS projects. This work started 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. In addition, the importance of the local RR attributes is changing in the course of a release, with emphasis on testing towards the end, 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.

Acknowledgement

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

References

1. S. Alam, S. Shahnewaz, D. Pfahl, and G. Ruhe. Monitoring Bottlenecks in Achieving Release Readiness A Retrospective Case Study across Ten OSS Projects. In *ESEM 2014*, 2014.
2. N. Ashrafi. The impact of software process improvement on quality: in theory and practice. *Information & Management*, 40(7):677–690, Aug. 2003.
3. A. Asthana and J. Olivieri. Quantifying software reliability and readiness. In *Communications Quality and Reliability, 2009. CQR 2009. IEEE International Workshop Technical Committee on*, pages 1–6. IEEE, May 2009.
4. V. R. Basili, G. Caldiera, H. D. Rombach, and R. V. Solingen. The Goal Question Metric Approach. *Encyclopedia of Software Engineering*, 1(1):578–83, 2000.
5. R. Brettschneider. Is your software ready for release? *IEEE Software*, 6(4):100, 1989.
6. C. Ebert. The impacts of software product management. *Journal of Systems and Software*, 80:850—-861, 2007.
7. C. Ebert. Software Product Management. *IEEE Software*, 31:21–24, 2014.
8. C. Ebert and S. Brinkkemper. Software Product Management An industry evaluation. *Journal of Systems and Software*, 95:10–18, 2014.
9. S. Gokhale. Optimal Software Release Time Incorporating Fault Correction. In *Proceedings of the 28th Annual NASA Goddard Workshop on Software Engineering*, pages 175–184, 2003.
10. D. Greer and G. Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 46(4):243—-253, 2004.
11. C. Larman. Software Development: Iterative & Evolutionary, 2014.
12. S. Mcconnell. Gauging software readiness with defect tracking. *IEEE Software*, 14(3):135–136, 1997.
13. Microsoft. Plan the Release Readiness Review Meeting.
14. T. Pearse, T. Freeman, and P. Oman. Using Metrics to Manage the End-Game of a Software Project. In *Proceedings of the Sixth International Software Metrics Symposium*, pages 207–215, 1999.
15. D. Port and J. Wilf. The Value of Certifying Software Release Readiness: An Exploratory Study of Certification for a Critical System at JPL. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 373–382. IEEE, Oct. 2013.
16. J. T. Quah and M. M. T. Thwin. Prediction of Software Readiness Using Neural Network. In *Proceedings of 1st international conference on information technology & applications*, pages 307–312, 2002.

17. J. T. S. Quah and S. W. Liew. Gauging Software Readiness Using Metrics. In *SMCia*, pages 426–431, 2008.
18. T.-S. Quah. Estimating software readiness using predictive models. *Information Sciences*, 179(4):430–445, Feb. 2009.
19. J. Rothman. Measurements to Reduce Risk in Product Ship Decisions, 2014.
20. P. R. Satapathy. Evaluation of Software Release Readiness Metric [0,1] across the software development life cycle. *Retrieved on March*, 5, 2010.
21. S. Shahnewaz and G. Ruhe. RELREA - An Analytical Approach for Evaluating Release Readiness. In *proceedings of Software Engineering and Knowledge Engineering (SEKE)*, 2014.
22. M. Staron, W. Meding, and K. Palm. Release Readiness Indicator for Mature Agile and Lean Software Development Projects. *Agile Processes in Software Engineering and Extreme Programming*, pages 93—-107, 2012.
23. M. Ware, F. Wilkie, and M. Shapcott. The Use of Intra-Release Product Measures in Predicting Release Readiness. *2008 International Conference on Software Testing, Verification, and Validation*, pages 230–237, Apr. 2008.
24. R. Wild and P. Brune. Determining Software Product Release Readiness by the Change-Error Correlation Function: On the Importance of the Change-Error Time Lag. In *HICSS*, pages 5360–5367, 2012.
25. L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.