# A Lightweight Approach for Estimating Probability in Risk-Based Software Testing

Rudolf Ramler[1](✉), Michael Felderer[2], and Matthias Leitner[2]

[1] Software Competence Center Hagenberg GmbH,
Softwarepark 21, 4232 Hagenberg, Austria
rudolf.ramler@scch.at

[2] Department of Computer Science, University of Innsbruck,
Technikerstrasse 21a, 6020 Innsbruck, Austria
{michael.felderer,matthias.leitner}@uibk.ac.at

**Abstract.** Using risk information in testing is requested in many testing strategies and recommended by international standards. The resulting, widespread awareness creates an increasing demand for concrete implementation guidelines and for methodological support on risk-based testing. In practice, however, many companies still perform risk-based testing in an informal way, based only on expert opinion or intuition. In this paper we address the task of quantifying risks by proposing a lightweight approach for estimating risk probabilities. The approach follows the "yesterday's weather" principle used for planning in Extreme Programming. Probability estimates are based on the number of defects in the previous version. This simple heuristic can easily be implemented as part of risk-based testing without specific prerequisites. It suits the need of small and medium enterprises as well as agile environments which have neither time nor resources for establishing elaborated approaches and procedures for data collection and analysis. To investigate the feasibility of the approach we used historical defect data from a popular open-source application. Our estimates for three consecutive versions achieved an accuracy of 73% to 78% and showed a low number of critical overestimates (<4%) and few underestimates (<1%). For practical risk-based testing such estimates provide a reliable quantitative basis that can be easily augmented with the expert knowledge of human decision-makers. Furthermore, these results also define a baseline for future research on improving probability estimation approaches.

**Keywords:** Risk-based testing · Risk assessment · Probability estimation · Defect prediction · Test management · Software testing

## 1 Introduction

Risk-based testing is a testing approach which considers risks related to a software product as the guiding factor to support decisions in all phases of the test process [1]. As the recently published international standard for software testing, ISO/IEC/IEEE 29119 [2] explicitly involves risks as an integral part of the testing process, there is increasing demand for methodological support on risk-based testing.

J. Großmann et al. (Eds.): RISK 2016, LNCS 10224, pp. 115–128, 2017.
DOI: 10.1007/978-3-319-57858-3_9

In general, a *risk* is an event that may possibly occur and, if it occurs, it has negative consequences. Risks are determined by the two factors *probability* and *impact*. The factor probability describes the likelihood that the negative event, e.g. a software failure, occurs and impact characterizes the cost if the failure occurs in operation. Assessing the risk exposure of a software feature or component requires estimating both factors. Impact can usually be derived from the business value associated to the feature defined in the software requirements specification. Probability is influenced by the implementation characteristics of the feature or component as well as the usage context in which the software system is applied.

Identifying and estimating risks is a core activity in risk-based testing. Nevertheless, we observed that many companies do not follow established approaches. In a study on risk-based testing we investigated the daily practice of software testing in several large and small companies [3, 4]. Especially small and medium enterprises do not systematically estimate risks and, if they do, they mostly rely on expert opinion. While expert opinion is a valuable source for risk information, experts seem to underestimate the probability of risks and may produce contradicting as well as misleading estimates [5].

In this paper we present a lightweight approach to estimating the risk probability in risk-based software testing and its evaluation. The application context mainly considered in this paper is functional testing, although the approach may as well be applied for testing a wide spectrum of functional and non-functional properties of a software system. The emphasis of the approach is on being lightweight, i.e., simple in its design and simple in its application. The aim of this paper is to explore “the simplest thing that could possibly work” [6]. Hence, the approach can easily be implemented as part of risk-based test strategy development in small and medium enterprises as well as in agile environments without specific prerequisites.

The remainder of this paper is structured as follows. Section 2 summarizes the underlying approach to risk-based test strategy development. Section 3 presents the lightweight approach to risk probability estimation. Section 4 provides an initial evaluation of the probability estimation approach. Finally, Sect. 5 concludes the paper and highlights directions of future research.

## 2 Background: Risk-Based Testing and Test Strategy Development

Many testing processes (e.g., [7, 8]) as well as standards (e.g., [2]) recommend the use of risk information in software testing. Several risk-based approaches for software testing have been proposed such as by Bach [9], Amland [10], and van Veenendaal [11]. Furthermore, comprehensive frameworks and guidelines for risk management as well as for risk-based testing have been developed in context of software security; prominent examples are the Risk Management Framework (RMF) [12] and the OWASP Testing Guide [13]. These approaches implement or are accompanied by various different ways for assessing risks [14–17].

In the remainder of this section we describe a previously presented, empirically evaluated process for risk-based test strategy development [18] as one example of how probability and impact values may be determined and used in practice. It is an essential first step when introducing risk-based testing in an organization [19] to establish a risk-based test strategy that anchors risk-orientation as basis for all testing activities in the entire software lifecycle. The process has been evaluated as part of a research transfer project for introducing risk-based testing in five small and medium software development companies.

In general, a *test strategy* describes how testing is organized and performed on the different test levels [20]. The usually rather generic strategy has to be refined for its implementation in context of a specific project or product iteration. The refinement results in a concrete test approach that defines the different types of testing that need to be performed, the test and quality assurance techniques to be applied, and the coverage and exit criteria used for tacking the progress and determining test completion.

Figure 1 provides an overview of the overall process for risk-based test strategy development. It consists of different steps, which are either directly related to the risk-based test strategy development (shown in bold font) or which are used to establish the preconditions (shown in normal font) for the process by linking test strategy development to the related processes (drawn with dashed lines) of defect management, requirements management and quality management.

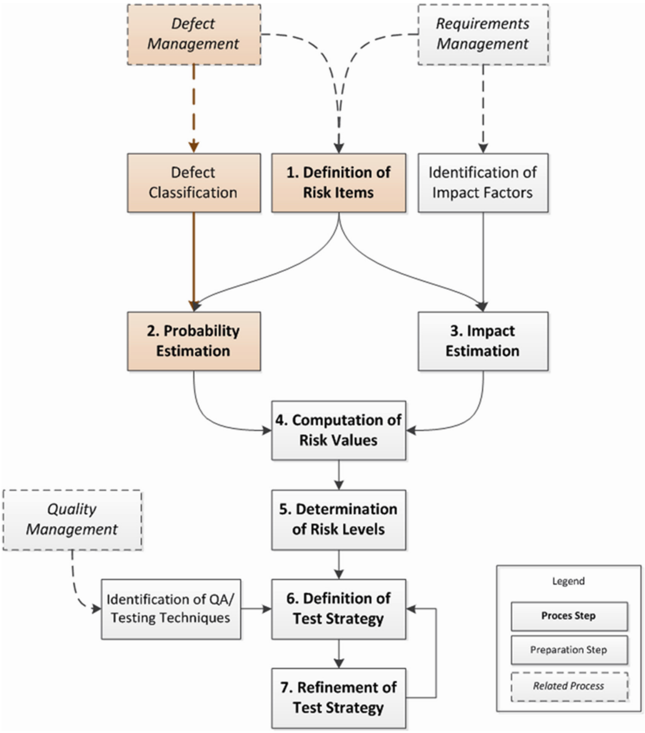

Defect Management
Requirements Management
Defect Classification
1. Definition of Risk Items
Identification of Impact Factors
2. Probability Estimation
3. Impact Estimation
4. Computation of Risk Values
5. Determination of Risk Levels
Quality Management
Identification of QA/ Testing Techniques
6. Definition of Test Strategy
7. Refinement of Test Strategy
Legend
Proces Step
Preparation Step
Related Process

**Fig. 1.** Probability estimation in risk-based test strategy development [18]. The highlighted steps relate to risk probability estimation as explored in this paper.

The process for risk-based test strategy development comprises seven core steps, which are as follows. In the first step, *risk items*, which are the basic elements associated with risks and mapped to test objects, are defined. They can be derived from established structures used in defect and requirements management. In the second step, for each risk item *probability values* are estimated which express their likelihood of defectiveness. For probability estimation, one can use data from defect classification [21] that captures and enhances the relevant data obtained from defect management. In the third step, for each risk item *impact values* are estimated which express the consequences of risk items being defective [22]. As the impact is closely related to the expected value of the components for the user or customer, requirements management is a main source of data for impact estimation. In the fourth step, *risk values* are computed from the estimated probability and impact values. The computed risk values can be used to group risk items, for example, according high, medium and low risk. In the fifth step, the spectrum of risk values is partitioned into *risk levels*, which comprise a further level of aggregation. The purpose of distinguishing different risk levels is to define classes of risks such that all risk items associated to a particular class are considered equally risky and as a consequence are subject to the same intensity of quality assurance and test measures. In the sixth step, the *test strategy is defined* on the basis of the different risk levels. For each risk level the test strategy describes how testing is organized and performed. Distinguishing different levels allows testing with different rigorousness in order to adequately address the expected risks. In the seventh step, the *test strategy is refined* to match the characteristics of the individual components of the software system (i.e., risk items).

The highlighted steps are related to risk probability estimation relevant for the approach further explored in this paper. Probability estimation is a core step in the risk-based test strategy development process to estimate risk values. As mentioned before, experts seem to underestimate the probability of risks and may produce contradicting as well as misleading estimates [5]. However, risk probabilities can be estimated based on historical defect data collected from previous releases or related projects. To support also less mature enterprises in applying defect data-based risk probability estimation, a lightweight approach is required.

## 3 Approach for Risk Probability Estimation

To improve risk estimation in context of small and medium enterprises, we recommend combining expert opinion with quantitative data from the systems' development history [18]. Figure 1 illustrates the different steps of a risk-based testing approach. It includes, first, the estimation of the factors probability and impact for each risk item.

In the context of testing the probability value expresses the likelihood of defectiveness of a risk item, i.e., the likelihood that a fault exists in a specific module that may lead to a failure. Most companies maintain a defect management system for reporting failures, tracing failures to faults and documenting their resolution. These systems capture the defect history of a software system and can serve as basis for deriving data for estimating future risk probabilities [18].

Modeling the usually complex relationship between software faults and resulting failures [20] requires considerable effort and a consistent data set that may not be available in practice. In contrast, projects in small and medium organizations are often following an on-demand, agile approach. Our lightweight approach has been inspired by the "yesterday's weather" principle used for planning in Extreme Programming [23]. It is a simple rule used in effort estimation, e.g., for estimating the amount of work a team can complete in a sprint. Instead of a complex estimation approach the rule suggests to use the amount of work completed in the previous sprint as estimate for the next sprint. This estimate is not meant to replace human judgement but to provide a quantitative basis that can be easily adjusted by including the knowledge and experience of human decision-makers. The benefit of this rule-based approach is its simplicity and widespread applicability.

In our approach we follow the same principle. Probability estimates are derived from defect counts of the previous version. In short, components with a high defect count in the last version are estimated to be likely defective in the next version and, vice versa, components that were already free of defects in the last version are still considered to be defect-free. Estimates based on defect counts are usually mapped to probability levels (e.g., high, medium and low), which are used to construct risk matrices that are the basis for the subsequent testing activities as shown in Fig. 2.

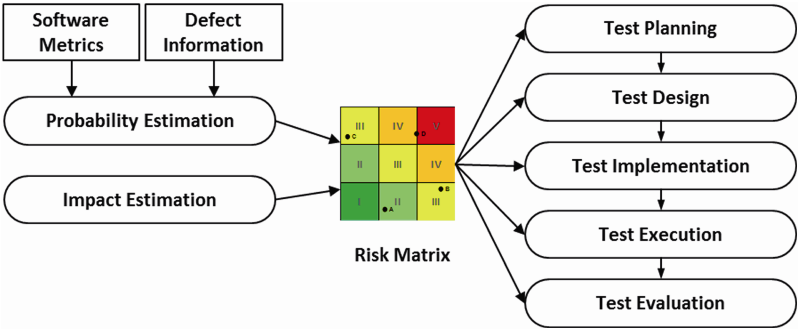


**Fig. 2.** Risk-based testing approach.

Extrapolation from defect counts provides a fast and easy way to estimate risk probabilities. However, the approach is based on the assumption of a continuous process and environment that keeps influence factors on risks stable over consecutive versions. Therefore, the estimates are adjusted by expert opinion to include knowledge about disruptive events in the development process, in testing or in the usage of the software system [24]. Human judgement is also used to decide about boundary cases and new components where no historical records are available.

# 4 Evaluation

We demonstrate the feasibility and explore the limits of the proposed lightweight estimation approach by applying it in context of the open-source project *jEdit*[1]. The tool jEdit is a widely used and mature programmer's text editor with – according to the project's description – hundreds of person-years of development behind it. It is written in Java and has been released as free software with full source code. The project jEdit has been subject to a previous study on defect prediction by Jureczko and Madeyski [25]. As part of their study the data has been made publically available and can be obtained from the *OpenScience tera-PROMISE* repository[2].

The data is used to illustrate and evaluate the approach of estimating the risk probability for source code files. In particular, we use available data in terms of defect counts per file from a "known" version *n* to make estimates for the next version *n + 1*. As described above, the estimates express the risk probability of a particular file containing defects according to the categories high/medium/low. A detailed analysis is provided by the following sub-sections.

- In Subsect. 4.1 we explore the distribution of defects in each individual studied version. We show that in each version there are a small number of highly defective files, a moderate number of files with a few defects, and a large number of defect-free files. We exploit this Pareto-like distribution for classifying the files as *high*, *medium* or *low* defective.
- In Subsect. 4.2 we compare the distribution of defects between versions and show that files with a high/low number of defects in version *n* usually also have a high/low number of defects in the next version *n + 1*. This trend is observable over consecutive versions and builds the foundation for making reliable estimates.
- In Subsect. 4.3 we estimate the high/medium/low probability of files being defective based on defect counts obtained from their previous version. We evaluate the results by computing the *accuracy* of the estimates (classification) as well as the number of *overestimates* and *underestimates*. Accuracy ranges from 73% to 78%, while critical underestimates are less than 1% and critical overestimates remain below 4%.
- In Subsect. 4.4 we discuss the threats to validity of our evaluation.

## 4.1 Versions and Defect Distributions

In our study we analyze four versions of jEdit (3.2.1, 4.0, 4.1, and 4.2), which are related to a continuous period of development of about three years. In this time interval the code base has steadily grown, from 129 KLOC (272 Java files) in version 3.2.1 to 171 KLOC (367 Java files) in version 4.2. In the same time the number of defects has been reduced from 382 to 106 defects (Table 1).

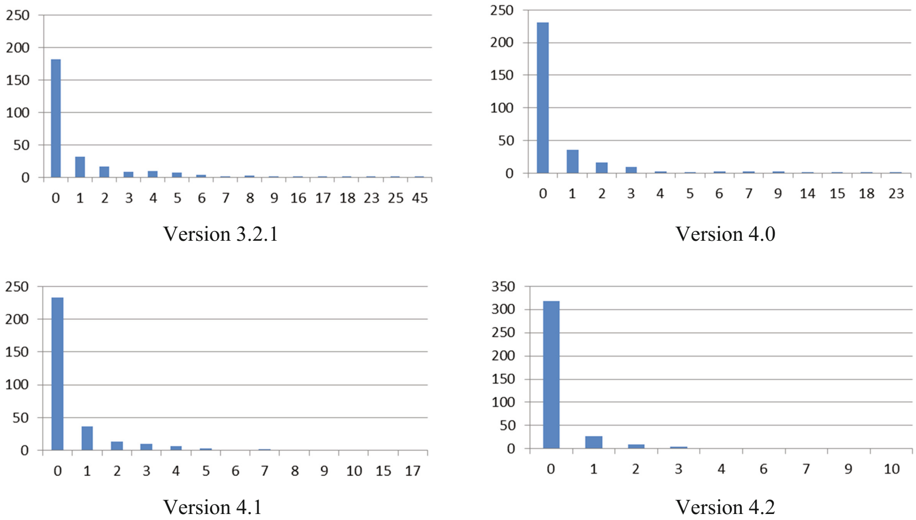
[1] http://www.jedit.org/

[2] http://openscience.us/repo/defect/ck/jedit

**Table 1.** Key measures of the studied versions of jEdit.

| Version | LOC | Files | Defects | Avg. defects/file | Max defects/file | Defect-free files |
|---|---|---|---|---|---|---|
| 3.2.1 | 128,883 | 272 | 382 | 1.40 | 45 | 67% |
| 4.0 | 144,803 | 306 | 226 | 0.74 | 23 | 75% |
| 4.1 | 153,087 | 312 | 217 | 0.70 | 17 | 75% |
| 4.2 | 170,683 | 367 | 106 | 0.29 | 10 | 87% |

In each version a Pareto-like distribution of defects to files can be observed. The top 10% of defective files contain 71% / 77% / 74% / 90% of the defects in each of the studied versions. The histograms in Fig. 3 show the distribution of files per number of defects. In each version there are a large number of defect-free files (0 defects), a moderate number of files with only a few defects (1 to 3 defects) and a small number of files with many defects (4 or more defects). In the following, we can exploit this information for classifying files as *low*, *medium* or *high* defective.



Version 3.2.1

Version 4.0

Version 4.1

Version 4.2

**Fig. 3.** Number of files per defect count for each version.

### 4.2 Defective Files in Consecutive Versions

Although the studied application is growing over time and undergoes many modifications, a large share of the files (75% to 89%) can be traced from one version to the next. These files are present in version *n* as well as in version *n* + *1*. In this section we explore if we can take advantage of the relationship these files share over consecutive versions to make reliable estimates.

The underlying assumption is that files that have a large number of defects in one version will also have many defects in the next version and, vice versa, files that do not have any defects will stay defect-free. The overall number of defects changes over time and so does the number of defects per file. However, we are mainly interested if the overall relationship in terms of high/medium/low number of defects stays the same. We first investigate our assumption by charting the cumulative gain in terms of defects over all files in version *n* + *1* when ordering them according to their defectiveness in version *n*. Figures 4, 5 and 6 show the respective gain charts (lift plot [26]).

The x-axis of the gain chart depicts the number of files subject to testing. The y-axis shows the cumulative percentage of total defects that can be found in testing when a particular ordering of the files is applied. The *optimal* ordering (green curve) is what one gets when sorting the files according to their actual number of defects in version *n* + *1*. It represents the best way of ordering the files for testing. However, the actual numbers are unknown at the time of testing and this ordering can only be determined from an ex-post view on the data. At the time of testing one has to rely on estimates. In the worst case such estimates are equal to guessing, which would correspond to a *random* ordering of the files (gray dotted 45-degree diagonal line). In our approach the *estimation* is based on the defect numbers of the previous version *n*. These numbers are already known when testing for version *n* + *1* is going to start and can therefore be used for prioritization, i.e., ordering the files (blue line) accordingly. If several files in version *n* have the same number of defects, their ordering for version *n* + *1* cannot be determined. Without any further information for making estimates one has to assume a random ordering for these files. The different possible combinations result in range defined by the curves *best estimate* and *worst estimate* (green/red dashed lines).

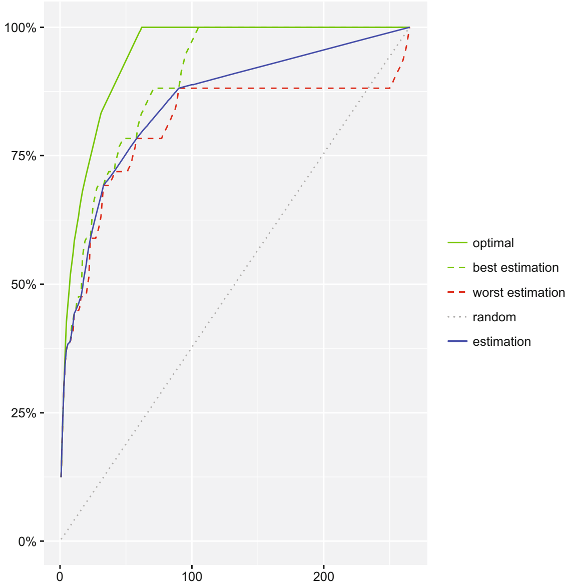


**Fig. 4.** Cumulative gain in version 4.0 based on the number of defects in 3.2.1. (Color figure online)

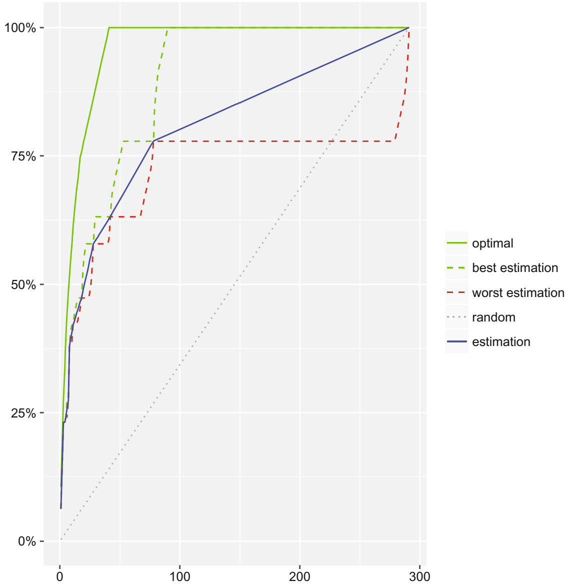


**Fig. 6.** Cumulative gain in version 4.2 based on the number of defects in 4.1. (Color figure online)

Concerning our assumption, we can make the following observations. First, the steep initial growth of the estimate curve shows that files with a high number of defects in version *n* are usually also containing a high number of defects in version *n + 1*. Second,

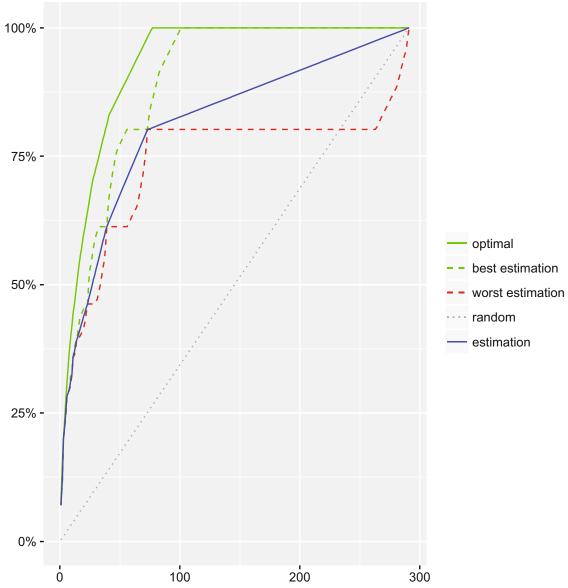


**Fig. 5.** Cumulative gain in version 4.1 based on the number of defects in 4.0. (Color figure online)

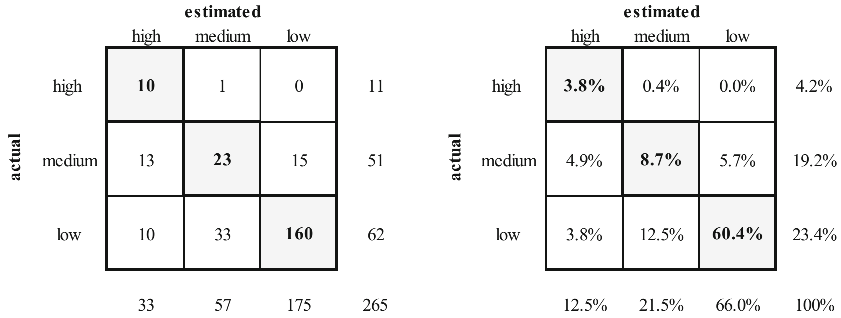
the long tails of the curves are due to the many defect-free files in all versions. Third, the gap between the tails of the curves *best estimate* and *worst estimate* indicates that some defects have been introduced in version *n* + *1* to previously defect-free files of version *n*.

In general, the ordering based on our *estimation* provides a substantial improvement over guessing. For version 4.0 (Fig. 4) the resulting ordering would allow finding about 75% of the defects after testing only 20% of the files, and more than 90% defects can be found after testing 50% of all files. Similar findings can be derived from all three gain charts (Figs. 4, 5 and 6). Hence, these findings provide a useful basis for making estimates and they confirm the feasibility of the proposed estimation approach.

### 4.3 Estimating Probability Classes

An exact ordering of the files is not required for developing a risk-based test strategy as initially described in Sect. 2. It is usually sufficient to associate the different files or parts of the system to risk probability classes such as high/medium/low probability of being defective. In this section we evaluate the feasibility of estimating probability classes based on the number of defects associated with a file as investigated in the previous Sect. 4.2. The classification used in the following is based on the findings from Sect. 4.1, where we explored the defectiveness of the files per version as *high* = ≥4 defects, *medium* = 3 to 1 defects, *low* = 0 defects.

Figures 7, 8 and 9 show the confusion matrix that result from estimating probability classes for the three versions 4.0, 4.1 and 4.2. Estimated numbers are shown on the x-axis of the confusion matrix and actual numbers are shown on the y-axis. Thus, the matrix for version 4.0 can be read as follows. The first row shows that out of the 11 files (4%) in version 4.0 with an actual high defectiveness, 10 were correctly estimated to have a high probability of being defective and 1 was underestimated as having a medium probability of being defective although being highly defective. In contrast, the first column shows that in total 33 files (12%) were estimated to have a high probability of being defective. Out of these 10 files (4%) are actually highly defective, 13 (5%) have a medium and 10 (4%) a low actual defectiveness. Thus, 10 files were classified correctly and 23 were overestimated.



| | | estimated | | | |
|---|---|---|---|---|---|
| | | high | medium | low | |
| actual | high | **10** | 1 | 0 | 11 |
| | medium | 13 | **23** | 15 | 51 |
| | low | 10 | 33 | **160** | 62 |
| | | 33 | 57 | 175 | 265 |

| | | estimated | | | |
|---|---|---|---|---|---|
| | | high | medium | low | |
| actual | high | **3.8%** | 0.4% | 0.0% | 4.2% |
| | medium | 4.9% | **8.7%** | 5.7% | 19.2% |
| | low | 3.8% | 12.5% | **60.4%** | 23.4% |
| | | 12.5% | 21.5% | 66.0% | 100% |

**Fig. 7.** Confusion matrix for estimates of version 4.0.

| | | estimated | | | |
|---|---|---|---|---|---|
| | | high | medium | low | |
| actual | high | **9** | 6 | 2 | 17 |
| | medium | 5 | **29** | 26 | 60 |
| | low | 0 | 24 | **190** | 77 |
| | | 14 | 59 | 218 | 291 |

| | | estimated | | | |
|---|---|---|---|---|---|
| | | high | medium | low | |
| actual | high | **3.1%** | 2.1% | 0.7% | 5.8% |
| | medium | 1.7% | **10.0%** | 8.9% | 20.6% |
| | low | 0.0% | 8.2% | **65.3%** | 26.5% |
| | | 4.8% | 20.3% | 74.9% | 100% |

**Fig. 8.** Confusion matrix for estimates of version 4.1.

| | | estimated | | | |
|---|---|---|---|---|---|
| | | high | medium | low | |
| actual | high | **4** | 2 | 1 | 7 |
| | medium | 8 | **15** | 11 | 34 |
| | low | 6 | 43 | **201** | 41 |
| | | 18 | 60 | 213 | 291 |

| | | estimated | | | |
|---|---|---|---|---|---|
| | | high | medium | low | |
| actual | high | **1.4%** | 0.7% | 0.3% | 2.4% |
| | medium | 2.7% | **5.2%** | 3.8% | 11.7% |
| | low | 2.1% | 14.8% | **69.1%** | 14.1% |
| | | 6.2% | 20.6% | 73.2% | 100% |

**Fig. 9.** Confusion matrix for estimates of version 4.2.

Various performance measures can be computed from the confusion matrix to evaluate the estimates. In the following we look at (1) *accuracy* as well as the number of (2) *overestimates* and (3) *underestimates*.

*Accuracy* is defined as the percentage of correct classifications over all classifications. In the confusion matrix the correct estimates can be found in the three diagonal cells from the top left (estimated high and actual high) to the bottom right (estimated low and actual low). The correct estimates in our study led to an accuracy of 72.8% in version 4.0, 78.4% in version 4.1, and 75.6% in version 4.2.

*Overestimate*s are defined by the percentage of classifications where the estimated classification is higher than the actual classification. In the confusion matrix the overestimates can be found in the three cells on the lower left (estimated high/medium and actual medium/low). For the three versions 4.0, 4.1 and 4.2 the overestimates are 21.1%, 10% and 19.6%. Overestimates ("false alarms") mean that files are subject to more rigorous testing than actually considered necessary. From the perspective of a risk-based approach, overestimation may lead to a waste of time and resources. Furthermore, in terms of "false alarms" they reduce the confidence in the estimates. These problems are particularly critical for files that were estimated to have a high probability of being defective yet they were found to actually have a low defectiveness. Only a small number of critical overestimates were produced: 3.8% for version 4.0, none (0%) for version 4.1, and 2.1% for version 4.2.

*Underestimates* are defined by the percentage of classifications where the estimated classification is lower than the actual classification. In the confusion matrix the underestimates can be found in the three cells on the top right (estimated low/medium and actual medium/high). In the three versions 6%, 11.7% and 4.8% of the misclassified files are underestimates. Underestimation means that defective files do not get enough attention and, thus, defects may be missed in testing. Again, we consider underestimates as especially critical if the files that actually have a high defectiveness were estimated as low defective. A very low number of files have been seriously underestimated: none (0%) in version 4.0, 0.7% in version 4.1, and 0.3% in version 4.2.

### 4.4 Threats to Validity

The prerequisite for applying the proposed approach is a complete and consistent record of defects mapped to files for each release over an extended period of time. Reliable, high-quality defect data is also a major factor for the validity of our evaluation. We therefore selected a publically available data set that has been used in a previous, rigorously reviewed empirical study on defect prediction [25].

Defect severity has not been considered in our study. The analyzed defect data does not include severity ratings of individual defects. In our initial approach [18] we suggested using severity ratings to weight defects, which provides the possibility to adjust the ordering if several files have the same number of defects. However, for the large share of files found to be defect-free the estimation will not change. One may even decide to ignore defect severities when estimating risk probabilities as this information is included in the impact side of risk that has to be included in a next step of the risk-based testing process (which is outside the scope of this study).

The approach relies on information derived from the files in the previous version. This information is not available for new files. In our initial approach we considered all new files relevant for testing, implicitly assuming a high probability of being defective. However, in our study we found that most new files are actually defect-free and the remaining ones only have few defects. Therefore our initial assumption seems to be too pessimistic and it produces overestimates.

The discussed threats affect the validity of the evaluation, in particular, its construct and internal validity. Concerning external validity it is clear that a generalization from only one studied case is limited as in any case study research [27]. The main goal of our study was to demonstrate the feasibility of the approach, which we were able to show in the selected case. Furthermore, the studied system can be considered representative for long-running projects developing desktop applications. However, further replications are necessary to validate our findings in different contexts.

## 5 Conclusion and Future Work

Estimating risk probabilities is an important step in risk-based testing. However, this step is often performed in an informal way, based on expert opinion and intuition rather than on quantitative data. One of the reasons is the lack of availability of such data in projects performed in small and medium enterprises or in agile environments. These projects neither have the time nor the resources for establishing additional procedures for data collection and analysis. Similarly, sophisticated risk estimation procedures are out of scope for these projects.

In this paper we therefore proposed a lightweight approach for estimating the risk probability in risk-based software testing following the "yesterday's weather" principle. Probability estimates are based on defect numbers from the preceding version. Simplicity of the estimation approach is of foremost concern. It is intended to be easily implemented as part of risk-based test strategy development in small and medium enterprises without specific prerequisites. The only required source of information is defect data from previous versions, which can usually be derived from existing defect databases.

To investigate the feasibility of the approach we performed an evaluation on the popular open-source application jEdit. We used historical defect data to estimate the defect probability of files for three consecutive versions. Our estimates achieve an accuracy of 73% to 78%. Furthermore, they resulted in a low number of critical overestimates (less than 4%) and only a few underestimates (less than 1%). The results show that the approach is capable to satisfy the requirements suggested for applying defect prediction as basis for risk-based testing [28].

In this paper our focus was on a lightweight estimation technique with the goal to find the simplest approach that could possibly work. As part of future work we will investigate strategies to improve the estimation approach and to increase its accuracy while still keeping it simple. Our aim is to include easy-to-compute process metrics and product metrics to augment the probability estimates based on defect data.

**Acknowledgments.** This work has been supported by the COMET Competence Center program of the Austrian Research Promotion Agency (FFG), and the project MOBSTECO (FWF P 26194-N15) funded by the Austrian Science Fund.

## References

1. Felderer, M., Schieferdecker, I.: A taxonomy of risk-based testing. Int. J. Softw. Tools Technol. Transf. **16**(5), 559–568 (2014)
2. ISO/IEC/IEEE 29119-2:2013 Software and systems engineering – Software testing – Part 2: Test processes. International Organization for Standardization, Geneva (2013)
3. Felderer, M., Ramler, R.: A multiple case study on risk-based testing in industry. Int. J. Softw. Tools Technol. Transf. **16**(5), 609–625 (2014)
4. Felderer, M., Ramler, R.: Risk orientation in software testing processes of small and medium enterprises: an exploratory and comparative study. Software Qual. J. **24**(3), 519–548 (2016)
5. Ramler, R., Felderer, M.: Experiences from an initial study on risk probability estimation based on expert opinion. In: Joint Conference of the 23rd International Workshop on Software Measurement and the Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), pp. 93–97. IEEE (2013)
6. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Boston (2000)

7. Spillner, A., Rossner, T., Winter, M., Linz, T.: Software Testing Practice: Test Management: A Study Guide for the Certified Tester Exam ISTQB Advanced Level. Rocky Nook, Santa Barbara (2007)
8. Black, R.: Advanced Software Testing. Guide to the ISTQB Advanced Certification as an Advanced Test Manager, vol. 2. Rocky Nook, Santa Barbara (2009)
9. Bach, J.: James Bach on risk-based testing: how to conduct heuristic risk analysis. Softw. Test. Qual. Eng. (STQE) Mag., 23–28, November/December 1999
10. Amland, S.: Risk-based testing: risk analysis fundamentals and metrics for software testing including a financial application case study. J. Syst. Softw. **53**(3), 287–295 (2000). Elsevier
11. van Veenendaal, E.: The PRISMA Approach. Uitgeverij Tutein Nolthenius, The Netherlands (2012)
12. CERT: Risk Management Framework (RMF). United States Computer Emergency Readiness Team, US-CERT, July 2013
13. OWASP: Testing Guide Ver. 4, Open Web Application Security Project, September 2014
14. Kontio, J.: Risk management in software development: a technology overview and the Riskit method. In: 21st International Conference on Software Engineering. ACM (1999)
15. Felderer, M., Haisjackl, C., Pekar, V., Breu, R.: A risk assessment framework for software testing. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014. LNCS, vol. 8803, pp. 292–308. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45231-8_21
16. Herrmann, A.: The quantitative estimation of IT-related risk probabilities. Risk Anal. **33**(8), 1510–1531 (2013)
17. Vose, D.: Risk Analysis: A Quantitative Guide. Wiley, Hoboken (2008)
18. Ramler, R., Felderer, M.: A process for risk-based test strategy development and its industrial evaluation. In: Abrahamsson, P., Corral, L., Oivo, M., Russo, B. (eds.) PROFES 2015. LNCS, vol. 9459, pp. 355–371. Springer, Cham (2015). doi:10.1007/978-3-319-26844-6_26
19. Felderer, M., Ramler, R.: Integrating risk-based testing in industrial test processes. Software Qual. J. **22**(3), 543–575 (2014)
20. ISTQB: Standard glossary of terms used in software testing. Version 2.1 (2010)
21. Felderer, M., Beer, A.: Using defect taxonomies for testing requirements. IEEE Softw. **32**(3), 94–101 (2015)
22. Gitzel, R., Krug, S., Brhel, M.: Towards a software failure cost impact model for the customer: an analysis of an open source product. In: 6th International Conference on Predictive Models in Software Engineering (PROMISE). ACM (2010)
23. Beck, K., Fowler, M.: Planning Extreme Programming. Addison-Wesley Professional, Boston (2001)
24. Felderer, M., Haisjackl, C., Breu, R., Motz, J.: Integrating manual and automatic risk assessment for risk-based testing. In: Biffl, S., Winkler, D., Bergsmann, J. (eds.) SWQD 2012. LNBIP, vol. 94, pp. 159–180. Springer, Heidelberg (2012). doi:10.1007/978-3-642-27213-4_11
25. Jureczko, M., Madeyski, L.: Towards identifying software project clusters with regard to defect prediction. In: 6th International Conference on Predictive Models in Software Engineering (PROMISE). ACM (2010)
26. Witten, I.H., Eibe, F.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Francisco (2005)
27. Runeson, P., Höst, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering: Guidelines and Examples. Wiley, Hoboken (2012)
28. Ramler, R., Felderer, M.: Requirements for integrating defect prediction and risk-based testing. In: 42nd Euromicro Conference on Software Engineering and Advanced Applications. IEEE (2016)