Comparison of Process Quality Characteristics Based on Change Request Data

Holger Schackmann and Horst Lichter

RWTH Aachen University, Research Group Software Construction
Ahornstr. 55, 52074 Aachen, Germany
{Schackmann,Lichter}@swc.rwth-aachen.de

Abstract. The evaluation of metrics on data available in change request management (CRM) systems offers valuable information for the assessment of process quality characteristics. The definition of appropriate metrics that consider the underlying change request workflow and address the information needs of an organization is an intricate task.

Furthermore CRM systems usually provide only a number of predefined metrics with limited adaptability. The tool BugzillaMetrics offers a more flexible approach that simplifies defining and adjusting new metrics. However a systematic approach for deriving an appropriate metric in a target-oriented way is needed. This paper describes a corresponding procedure on how to develop and validate metrics on CRM data applicable for the comparison of process quality characteristics.

Keywords: Process Metrics, Change Request Management, Metric Specification, Software Measurement Design, Measurement Tool.

1 Introduction

The management of a large software project portfolio raises several managerial challenges, like balancing resource allocation between different projects, and aligning development processes to the standards of the organization. Hence the project statuses and process quality characteristics, like planning precision or problem resolution speed, must be monitored continuously in order to identify development process weaknesses, and assess process improvements. Collecting the required data by regularly project status reporting can be expensive and intrusive, and furthermore ignores the past history of the process [1]. This motivates mining data from routinely collected repositories like change request management (**CRM**) systems.

The usage of this data for evaluating process quality characteristics imposes certain difficulties. Appropriate metrics will depend on the designated process and the improvement goals, as well as on the data available. It must be validated that the metrics are proper numerical characterizations of the qualities of interest, and that the measurements can be compared between different projects.

However existing CRM tools provide only a number of fixed metric evaluations and are limited in their adaptability [2]. Hence extraction and integration of the data

R. Dumke et al. (Eds.): IWSM / MetriKon / Mensura 2008, LNCS 5338, pp. 127–140, 2008.

typically require the development of custom scripts. Validation of the metrics most often necessitates adjusting the metrics definition and corresponding scripts, which is time-consuming and costly.

The open source tool BugzillaMetrics implements a more flexible approach for the evaluation of metrics on CRM data, based on declarative metric specifications [2]. The tool allows concentrating the main effort on the model of the metric, not on its implementation. Thus experimenting with metrics and adjusting them is faster and easier.

But, first experience with using the tool has revealed certain pitfalls in developing appropriate metric definitions [2]. This motivates the need for a structured approach for developing metrics on CRM data.

This paper presents a procedure to systematically develop metrics on CRM data used to compare process quality characteristics. This procedure includes validation steps as well as guidance for the interpretation of the metric results. First an overview of related work is given. Section 3 briefly describes the BugzillaMetrics tool.

2 Related Work

There exist numerous approaches to analyse CRM data as well as data from version control systems for several purposes (e.g. visualization of software evolution [4], or change impact analysis [5]). A survey is given by Kagdi et al [6]. Some of these approaches do also analyze specific aspects of the process. For example Sliwerski et al. present an approach to reconstruct links between the version-control system and resolved defect reports in the CRM database in order to analyse the frequency of fix-inducing changes [7]. Koponen presents a tool to analyse several aspects of maintenance processes of open source software, like typical defect-lifecycles, and origin of change analysis [8]. Gasser and Ripoche analyse CRM data of open source projects in order to extract their process models [9].

However none of these approaches is targeted at a general procedure for the assessment of process quality characteristics based on CRM data.

3 BugzillaMetrics

BugzillaMetrics is based on user defined metric specifications that abstract from the way the information is stored in the CRM database [3]. The basic building blocks for these specifications are **filters** for properties of a change request (e.g. its severity, status, or target milestone), and **events** that occur in the history of a change request (e.g. its creation, a change of the assignee, or the reopening of a resolved request). Filters and events can be combined with Boolean operators.

Each metric specification contains a **base filter** that defines which change requests are considered during the calculation (e.g. only change requests that belong to a certain product). Further on the **evaluation time period** and the **time granularity** (e.g. month or year) are defined.

Then one of several predefined **value calculators** can be applied to calculate a value for individual change requests in each time interval according to the given time

granularity. Examples of value calculators are the calculation of the length of a time interval between two specified events in the lifecycle of a change request, the calculation of the time a change request resides in a certain state, or the calculation of the number of occurrences of certain events during a time period. In the latter case an optional **weight** can be applied (e.g. a weighting by the severity of the change request, or by its estimated remaining workload). In terms of the ISO/IEC 15939 standard [10] the outcome of a value calculator can be denoted as **base measure** while a change request is the entity to be characterized by measuring.

The outcome of these value calculators can be combined with operations like sum, maximum, or mean value to calculate a result for a certain time interval. This outcome represents a **derived measure** related to the process in a certain time interval.

Thereby the tool offers a large flexibility for the specification of metrics. Furthermore the metric specification is separated from the way the required information is retrieved from the CRM database.

4 Developing Metrics on Change Request Data

This section describes a procedure on how to develop and validate metrics that target at the comparison of process quality characteristics. The approach is exemplified by developing metrics applicable to the CRM database of the Eclipse open source community.

4.1 Bidirectional Quality Models

In order to derive a metric we rely on the approach of bidirectional quality models [11]. This subsection briefly describes the related concepts (see Figure 1) and maps them to the terms of the ISO measurement information model contained in the ISO/IEC 15939 standard [10].

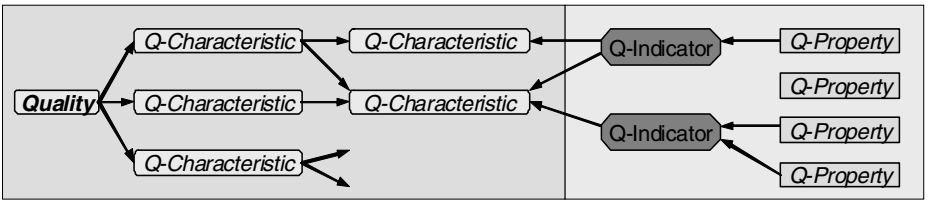


Fig. 1. Concepts of the bidirectional quality model

On the one side the **quality characteristics** reflect high-level requirements on the quality. In terms of the ISO/IEC 15939 standard these quality characteristics correspond with information needs derived from the business, organizational, regulatory, product or project objectives. An example of a quality characteristic is planning precision which can be subdivided into the quality characteristics: adherence to schedule, adherence to planned effort, and process transparency.

On the other side the **quality properties** denote objective characteristics of an entity (i.e. product, process, or system), that can be used to distinct between the considered entities. In terms of the ISO/IEC 15939 standard a quality property is an attribute of an entity that can be objectively and quantitatively distinguished by automated means. Examples for such quality properties are the total number of defects, the number of reopened change requests, or the frequency of assignee changes of a change request.

The quality properties will be used in a bottom-up fashion to form **quality indicators**. A quality indicator describes how a number of quality properties can be interpreted with respect to a quality characteristic. Hence the quality indicators bridge the gap between the technical view of quality properties and the abstract view of the quality characteristics. The notion of quality indicator complies with ISO/IEC 15939.

4.2 Identification of Quality Characteristics

The process quality characteristics of interest correspond to information needs that are in general derived from the objectives of the organization [12]. These characteristics can be refined stepwise.

For example the Eclipse community applies an agile development process based on several practices [13]. This process implicitly contains certain objectives, e.g. the planning precision of the scheduled milestones.

Related to the practice called “community involvement”, one can derive the process quality characteristic “**responsiveness to incoming defect reports**”, since the establishment of an active community requires timely reactions on observed problems. Note that this characteristic does not consider the resolution time of defects, but the duration to the first reaction on an incoming defect report. As most of the Eclipse projects are related to offering a general tools and integration platform the responsiveness on defect reports will have an impact on dependent projects based upon the Eclipse technology. In the following we will use this process quality characteristic as an example.

4.3 Identification of Quality Properties

In order to identify measurable quality properties it is necessary to analyze the way the CRM system is used, e.g. it must be examined what is the typical workflow of a change request, and which information is collected on a change request. Then quality properties need to be defined where some relation to the quality characteristics is conjectured.

Since each quality characteristic is related to some improvement goal, potential quality properties can be identified based on the Goal Question Metric approach [14]. However since the analysis is based on CRM data, it must be possible to determine a quantitative value for each quality property based on the data collected during the lifecycle of a change request. Naturally there will be some process quality characteristics where it is not possible to determine related quality properties, since not all quality characteristics can be evaluated based on the available CRM data.

For our ongoing example we need to find out which reaction can be considered as an appropriate acknowledgement for an incoming defect report. At first sight this will

be either adding an additional comment, changing the status of the defect report, or assigning the defect report to a specific assignee. These events can be specified in a metric of the BugzillaMetrics tool. The plausibility of the metric can then be validated by inspecting the results calculated for individual change requests and examining whether the history of a request conforms to the envisaged interpretation.

In our case it needs to be checked whether the metric considers all relevant reactions on defect reports. A detailed consideration shows that there are some more reactions, like changing the severity, priority, or the target milestone of the defect report, since this gives feedback about how the defect is rated by the project team. Hence the metric definition will be refined stepwise.

Until now the metric does only specify how the numbers for individual change requests are calculated. The next subsection will describe how these numbers will be aggregated in order to compare the process quality characteristics of different projects.

4.4 Definition of Quality Indicators

At first it needs to be defined how the values for individual change requests of a project can be aggregated. An appropriate metric must fulfil the following requirements:

- **Elimination of interfering factors:** It must be avoided that the aggregated value can be predominantly influenced by other factors, like size and age of the project. If other factors interfere with the original intention of the measurement the result will be difficult to interpret and to compare between projects.

  This would be the case in our example if we take the arithmetic mean of the individual first reaction values for each defect report. The result would potentially be distorted if a number of old but untreated defect reports is processed in a long-lived project. Hence the aggregation of the individual values will require using some kind of normalization or mean value that is stable against these kinds of outliers.
- **Timely relatedness to perceived problems in the process:** Each value calculated for a change request belongs to a specific time interval (e.g. month or year). It must be carefully considered that the assignment of the measurement values to time intervals stands in a temporal connection to potential causes in the process in order to prevent misleading interpretations.

  In our example this may happen if the values would be assigned to the time interval in which the first reaction occurred. Processing a number of old but untreated defect reports might then erroneously be interpreted as bad responsiveness in that time interval. Though the cause why these defect reports remained untouched dates back to the past.
- **Appropriate granularity of time intervals:** The granularity of the time intervals for which the change request values are aggregated must be similar to the release cycle of the project. Otherwise the resulting values will possibly be diverging due to the current phase in the release cycle (e.g. the endgame phase in the Eclipse development process [13]).

In order to derive an aggregated value that fulfils these requirements it is often better not to use the absolute values (in our case the time until first reaction on a change request), but to count the change requests where this value exceeds a certain threshold.

Hitting the threshold should be related to having a negative or positive impact on some quality characteristic. In our example it is reasonable to assume that defect reports with a severity equal or higher than normal that do not get any response within three days will probably retard or hamper dependent projects.

$$\text{ReactionEvents}(CR) = \{\textbf{commentAdded}(CR), \textbf{statusChange}(CR), \textbf{assigneeChange}(CR), \ \textbf{...}\} \quad (1)$$

$$\text{TimeUntilReaction}(CR) = \min\{\text{timeBetween}(\textbf{creation}(CR), e) \mid e \in \text{ReactionEvents}(CR)\} \quad (2)$$

$$\text{ThresholdHit}(CR) = \begin{cases} 0 & \text{TimeUntilReaction}(CR) \leq 3 \text{ days} \\ 1 & \text{otherwise} \end{cases} \quad (3)$$

$$\text{ThresholdHitDate}(CR) = \begin{cases} \textbf{creationDate}(CR) + 3 \text{ days} & \text{ThresholdHit}(CR) = 1 \\ \textbf{creationDate}(CR) + \text{TimeUntilReaction}(CR) & \text{ThresholdHit}(CR) = 0 \end{cases} \quad (4)$$

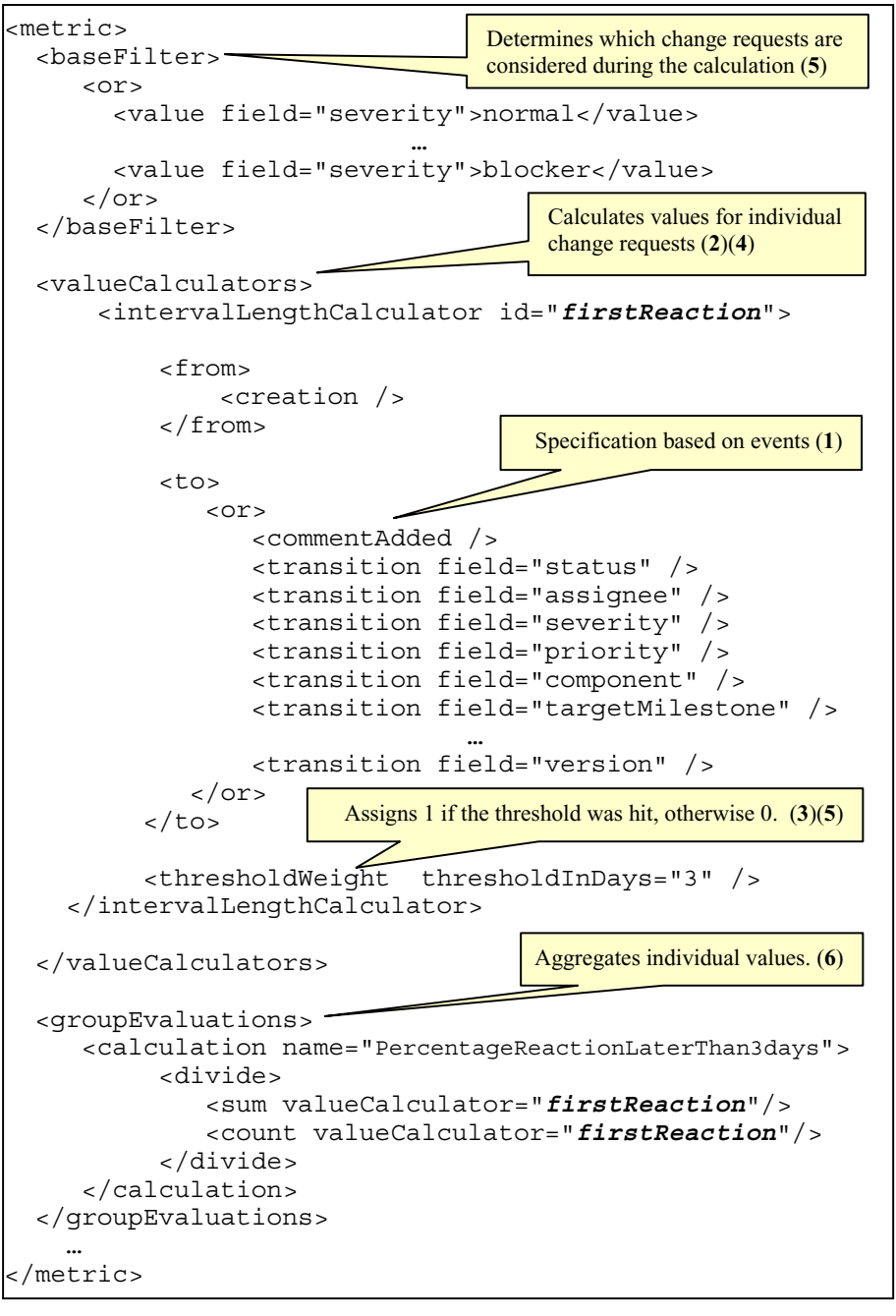
$$\text{ConsideredCRs}(\textit{timeInterval}) = \left\{ CR \;\middle|\; \begin{array}{c} \textbf{isDefectReport}(CR) \wedge \textbf{severity}(CR) \geq \textbf{\textit{normal}} \wedge \\ \text{ThresholdHitDate}(CR) \in \textit{timeInterval} \end{array} \right\} \quad (5)$$

$$\text{PercentageOfLateReactions}(\textit{timeInterval}) = \frac{\sum_{CR \in \text{ConsideredCRs}(\textit{timeInterval})} \text{ThresholdHit}(CR)}{\left|\text{ConsideredCRs}(\textit{timeInterval})\right|} \quad (6)$$

Fig. 2. Definition of PercentageOfLateReactions

Counting the change requests at the time when the threshold was hit ensures that there is a timely relation to perceived unresponsiveness. Additionally it enables to consider those defect reports in the calculation that did not yet get a response. Normalization of the results can be achieved by calculating the percentage of defect reports whose first response hits the threshold. The metric definition is sketched in Figure 2.

The calculation can again be specified with BugzillaMetrics (see Figure 3, numbers refer to the corresponding part of the metric definition). In order to use the aggregated result as quality indicator it requires defining some guidance how to interpret the results. This will be discussed in the following section.




```
<metric>
  <baseFilter>
     <or>
       <value field="severity">normal</value>
                              …
       <value field="severity">blocker</value>
     </or>
  </baseFilter>

  <valueCalculators>
      <intervalLengthCalculator id="firstReaction">

          <from>
              <creation />
          </from>

          <to>
             <or>
                <commentAdded />
                <transition field="status" />
                <transition field="assignee" />
                <transition field="severity" />
                <transition field="priority" />
                <transition field="component" />
                <transition field="targetMilestone" />
                              …
                <transition field="version" />
             </or>
          </to>

          <thresholdWeight  thresholdInDays="3" />
     </intervalLengthCalculator>

  </valueCalculators>

  <groupEvaluations>
     <calculation name="PercentageReactionLaterThan3days">
          <divide>
             <sum valueCalculator="firstReaction"/>
             <count valueCalculator="firstReaction"/>
          </divide>
     </calculation>
  </groupEvaluations>
     …
</metric>
```

Fig. 3. Metric Specification of "Percentage of Reaction later than 3 days". Numbers refer to the related formulas in Figure 2.

4.5 Interpretation Based on Empirical Data

The comparison within a peer group of projects offers a practical approach for the interpretation of the measurement values in order to decide whether a project is doing good or bad with respect to a certain quality characteristic.

The CRM system of the Eclipse project provides in our example the necessary empirical data. The resulting measurement values for a number of large projects are shown in Figure 4. Since the release dates of the major Eclipse projects are aligned in simultaneous release at the end of June, the measurement values have been calculated for the time periods between these releases.

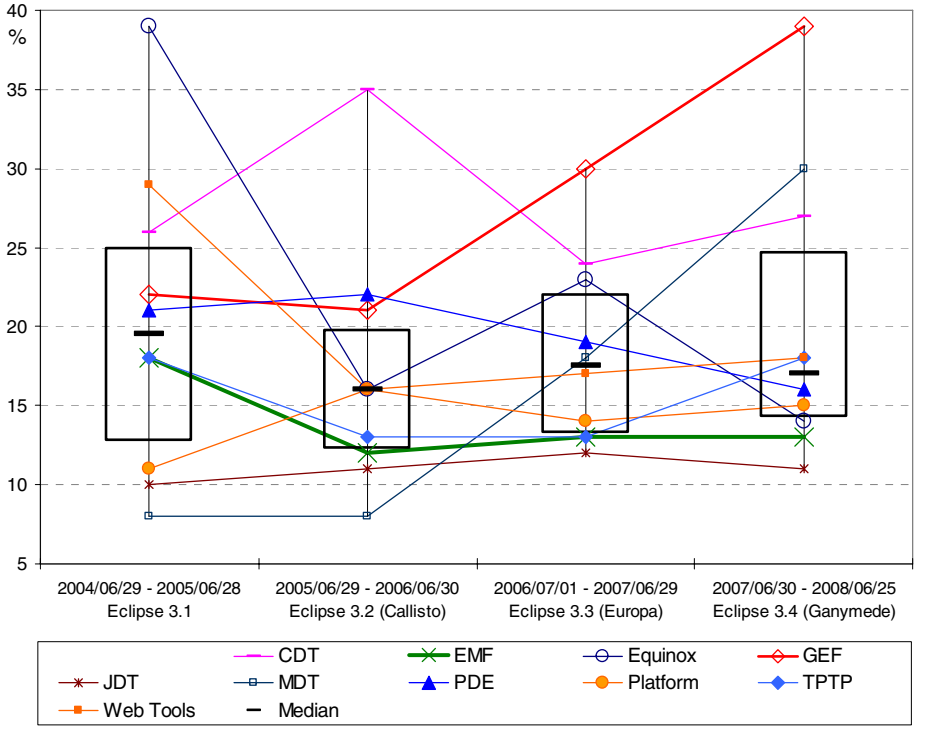


Fig. 4. Percentage of defect reports with the first reaction later than 3 days and a severity of "normal" or higher

It can be observed that the values for most of the projects tend to change only gradually between the years. This matches with the experience that discontinuous improvements of the process can rather seldom be achieved in large projects. If the values for most of the projects are volatile the underlying metric definition should be examined whether it really fulfils the requirements stated in the previous section.

A value for a project can now be interpreted in comparison to the value distribution in the time period related to the release. Naturally there can be slight differences of the interpretation dependent on the base for the comparison. The boxplot denotes the

second and third quartile of the data set. Projects within this range can be interpreted as having an around average responsiveness to incoming defect reports. So it can for example be stated that the EMF project had a good responsiveness for several release periods. The responsiveness of the GEF project is rather poor and declined in the last years. These results match with the experience gained during the development of a toolset at our research group that is called ViPER and is based on EMF and GEF [15].

4.6 Additional Example

In order to illustrate the procedure for metrics development an additional example will briefly be discussed in this section. At first we have to identify a **quality characteristic** of interest. A general objective in the development process is the efficient processing of the change requests. A related sub-goal is that the change requests should be resolved initially in an adequate way, since later rework often requires additional effort, and may be caused by insufficient coordination or misunderstandings related to the initial change request. Hence we can derive the quality characteristic **"frequency of rework"**.

In order to identify related measurable quality properties we have to analyse how rework is reflected in the available information about the lifecycle of a change request. Bugzilla has two related fields: *status* and *resolution*. If some action has been taken to resolve a change request, it is switched to the "Resolved" status. Some Eclipse projects also use the subsequent status values "Verified" and "Closed".

The resolution field indicates how the change request was resolved. Possible values are for example "Fixed" (some change to the software had been implemented), "Duplicate" (change request is already described in another existing change request), "WorksForMe" (described problem could not be reproduced), or "Not_Eclipse" (problem is related to a third-party package).

If the resolution is deemed to be incorrect the change request can be switched to the status "Reopened". A state transition to the status "Reopened" of a change request with resolution "Fixed" would indicate that a bug fix or new feature had not been implemented correctly. If the change request had a different resolution (e.g. "Duplicate" or "Not_Eclipse") this indicates that the decision to resolve the change request was based on some wrong assumptions.

Basically one can define the **quality property** "number of transitions to Reopened during the lifecycle of a change request". However we have to clarify whether we are only interested in transitions where some rework of previous changes in the software is required, or we are interested in all transitions where a change request is reexamined for some reason. The first interpretation would require taking the resolution field into account when identifying the respective state transitions. While both interpretations are reasonable, we choose the latter one here, since we are more interested in rework related to the overall change request process, instead of rework related to the quality of the implemented changes in the software.

Again, the plausibility of the previous assumptions can be validated by inspecting the lifecycle of individual change requests. By doing this we notice that the Eclipse CRM database allowed setting the resolution "Later" or "Remind" when a change request was switched to the status "Resolved". These resolution values are now deprecated since they do not indicate that the change request had really been resolved

```
<metric>
    ...
  <valueCalculators>
     <countEvents id="TransitionsToReopened">
        <event>
           <and>
              <transition field="resolution">
                 <from>FIXED</from>
                 <from>INVALID</from>
                 <from>WONTFIX</from>
                 <from>DUPLICATE</from>
                 <from>WORKSFORME</from>
                 <from>MOVED</from>
                 <from>NOT_ECLIPSE</from>
              </transition>
              <stateFilter>
                 <value field="status">REOPENED</value>
              </stateFilter>
           </and>
        </event>
     </countEvents>

     <countEvents id="TransitionsToResolved">
        <event>
           <and>
              <transition field="resolution">
                 <to>FIXED</to>
                       ...
                 <to>NOT_ECLIPSE</to>
              </transition>
              <stateFilter>
                 <value field="status">RESOLVED</value>
              </stateFilter>
           </and>
        </event>
     </countEvents>

  </valueCalculators>

  <groupEvaluations>
    <calculation name="ProportionOfRework">
      <divide>
        <count valueCalculator="TransitionsToReopened"/>
        <count valueCalculator="TransitionsToResolved"/>
      </divide>
    </calculation>
  </groupEvaluations>
    ...
</metric>
```

All possible resolutions, except „Later“ and „Remind“.

All possible resolutions, except „Later“ and „Remind“.

Fig. 5. Metric Specification of “Number of transitions to ‘Reopened’ divided by the number of transitions to ‘Resolved’ in a time period”

[16]. Instead such change requests should be marked either by setting a target milestone named "Future", by adding the "needinfo" keyword (which means asking more information from the reporter), or by decreasing their priority.

When counting status transitions to "Reopened" it must therefore be distinguished between change requests that had the resolution "Later" or "Remind", and those that had a proper resolution. Only transitions that had a proper resolution can be counted as reopened change requests. Otherwise the resulting values would be distorted for projects that once had used the "Later" and "Remind" resolution.

The **quality indicator** has to be defined in a way that the resulting values can be compared between different projects. The total number of transitions to the "Reopened" status in a certain time period will depend on the size of the project. In order to normalize the result we can divide by the total number of change requests resolved in that time period.

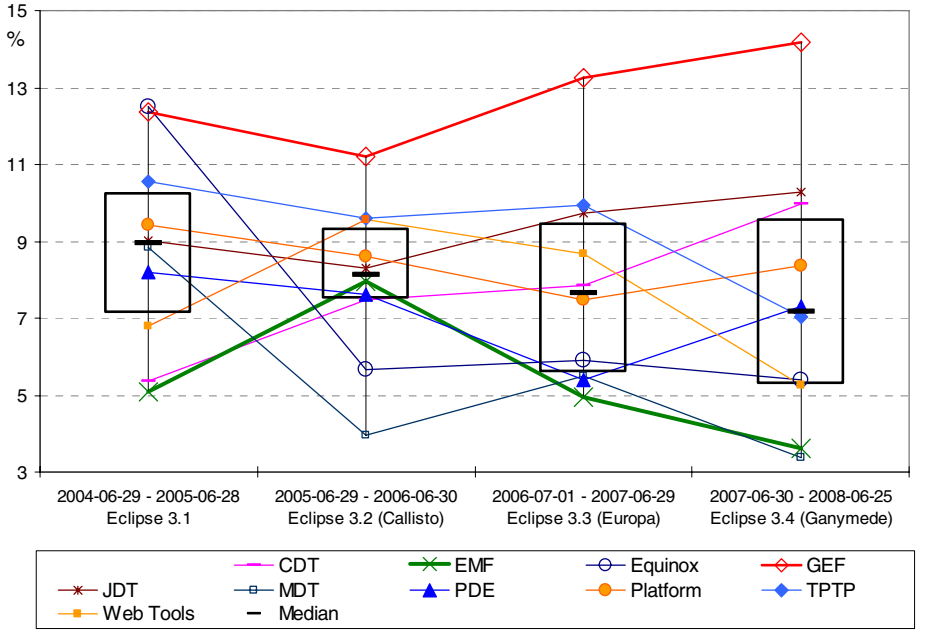


Fig. 6. Percentage of transitions to "Reopened" relative to the number of transitions to "Resolved" in a time period

More precisely we have to decide whether to count resolved change requests in that time period only once, or to count each state transition to "Resolved" of the same change request. Since the numerator (total number of transitions to the "Reopened" status) refers to all incorrect resolutions of a change request, we choose the second option for the denominator, since this corresponds to all resolutions of a change request. Again, state transitions to the "Resolved" status with the resolution "Later" or "Remind" should not be considered, since these change requests have not really been resolved. The corresponding metric specification is shown in figure 5.

Figure 6 shows the resulting measurement values of a number of large projects. Again it can be stated that the EMF project performs better than the average project, while GMF had a high proportion of reopened change requests. Additionally it can be stated from our experience that the GMF project has provided few major new features in the Europe and Ganymede release, and concentrated more on fixing defects.

5 Conclusion and Outlook

In this paper we presented a procedure for developing and validating metric definitions based on CRM data that can be used to evaluate quality characteristics of the development process. It is based on bidirectional quality models which provide an approach how to relate high-level quality characteristics to the technical view of measurable quality properties.

Summarizing, the following steps are performed for the development of a metric:

1. Deriving of process **quality characteristics** from the objectives of the organization.
2. Improvement goal based identification of corresponding **quality properties** and initial validation based on the inspection of individual change request lifecycles.
3. Definition of **quality indicators** that enable comparability between projects.
4. Interpretation based on empirical data.

The usage of metric specifications provided by the BugzillaMetrics tool facilitates an iterative refinement of the related metric definitions in steps 2 to 4. Further on the presented procedure guides the validation of underlying assumptions on different levels: by inspecting the lifecycle of individual change requests and by checking whether the results on the project and the project portfolio level match with experience. This enables to uncover wrong assumptions early during development of the metric.

In the presented examples is a one-to-one relation between the quality characteristic and the quality indicator. In general there can be several quality indicators that need to be weighted according to their influence on a quality characteristic.

It depends on the available CRM data which process characteristics can be evaluated. The Eclipse CRM database enables e.g. to consider characteristics like stability of the prospected target milestones, resolution speed of problem reports and enhancement requests, frequency of high-severity bugs, or the stability of the prioritization of change requests.

CRM systems with a more fine-grained workflow definition and more data collected like estimated and actual work time enable the evaluation of a wider range of quality characteristics.

Since BugzillaMetrics can automatically adapt to custom information collected for the change requests in the Bugzilla database, it can straightforwardly be used for related analyses. As an example the orthogonal defect classification (ODC) requires to classify each reported defect according to a *defect type* and a *defect trigger* in order to compare their distribution to an expected distribution in a certain phase of the process [17]. Given

that these classifications are collected in Bugzilla, the corresponding distributions and their change over time can directly be evaluated using BugzillaMetrics.

Furthermore it can be of interest to associate some kind of size metric to the change requests. By end of 2008 an extension of BugzillaMetrics will be released that enables collecting metrics from version control systems by considering the code changes related to a change request. This enables to consider size metrics like the size of a code change in the evaluations. A prerequisite for these evaluations will be the integration of Bugzilla with version control systems, such as CVS and Subversion, as provided by the Scmbug project [18].

Acknowledgements. We would like to thank Kisters AG, Aachen for supporting this work, and the Eclipse Foundation for providing access to the CRM database.

References

1. Cook, J.E., Votta, L.G., Wolf, A.L.: Cost-Effective Analysis of In-Place Software Processes. IEEE Trans. on Software Engineering 24(8) (1998)
2. Grammel, L., Schackmann, H., Lichter, H.: BugzillaMetrics - Design of an adaptable tool for evaluating user-defined metric specifications on change requests. In: Büren, Bundschuh, Dumke (eds.) Tagungsband des DASMA Software Metrik Kongresses MetriKon 2007. Shaker Verlag, Aachen (2007)
3. BugzillaMetrics, `http://www.bugzillametrics.org`
4. Gall, H.C., Lanza, M.: Software evolution: analysis and visualization. In: Proc.of the 28th international Conference on Software Engineering (ICSE 2006). ACM, New York (2006)
5. Canfora, G., Cerulo, L.: Impact Analysis by Mining Software and Change Request Repositories. In: Proc. of the 11th IEEE International Software Metrics Symposium – METRICS 2005, Como, Italy. IEEE CS Press, Los Alamitos (2005)
6. Kagdi, H.H., Collard, M.L., Maletic, J.I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. Journal of Software Maintenance 19(2), 77–131 (2007)
7. Sliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? In: 2nd International Workshop on Mining Software Repositories (MSR 2005). ACM Press, New York (2005)
8. Koponen, T.: RaSOSS - Remote Analysis System for Open Source Software. In: The International Conference on Software Engineering Advances (ICSEA 2006). IEEE Press, Los Alamitos (2006)
9. Gasser, L., Ripoche, G.: Distributed Collective Practices and F/OSS Problem Management: Perspective and Methods. In: Conference on Cooperation, Innovation & Technology (CITE 2003), Troyes, France (2003)
10. ISO/IEC 15939. Systems and software engineering – Measurement Process. International Organization for Standardization – ISO, Geneva (2007)
11. Simon, F., Seng, O., Mohaupt, T.: Code-Quality Management. Dpunkt-Verlag, Heidelberg (2006)
12. Ebert, C., Dumke, R.: Software Measurement: Establish - Extract - Evaluate - Execute. Springer, Berlin (2007)
13. Gamma, E.: Agile, Open Source, Distributed, and On-Time – Inside the Eclipse Development Process. Keynote Talk, ICSE, St. Louis, Missouri (2005)

14. Basili, V., Caldiera, G., Rombach, H.D.: The Goal Question Metric Paradigm. In: Encyclopedia of Software Engineering, John Wiley & Sons, Chichester (1994)
15. ViPER - Visual Tooling Platform for Model-Based Engineering, http://www.viper.sc
16. Eclipse Bugs – Remove LATER and REMIND resolutions, https://bugs.eclipse.org/178923
17. Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus, D.S., Ray, B.K., Wong, M.: Orthogonal Defect Classification - A Concept for In-Process Measurements. IEEE Trans. Software Eng. 18(11), 943–956 (1992)
18. Makris, K., Ryu, K.D.: Scmbug: policy-based integration of software configuration management with bug-tracking. In: USENIX Annual Technical Conference. USENIX Association, Berkeley (2005)