Evaluating Process Quality based on Change Request Data – An Empirical Study of the Eclipse Project

Holger Schackmann, Henning Schaefer, Horst Lichter

Research Group Software Construction, RWTH Aachen University
Ahornstr. 55, 52074 Aachen, Germany
{schackmann|lichter}@swc.rwth-aachen.de, henning.schaefer@rwth-aachen.de

Abstract. The information routinely collected in change request management systems contains valuable information for monitoring of the process quality. However this data is currently utilized in a very limited way. This paper presents an empirical study of the process quality in the product portfolio of the Eclipse project. It is based on a systematic approach for the evaluation of process quality characteristics using change request data. Results of the study offer insights into the development process of Eclipse. Moreover the study allows assessing applicability and limitations of the proposed approach for the evaluation of process quality.

Keywords: Process Metrics, Open Source Development, Change Request Management, Process Improvement, Metric Specification, Mining Software Repositories

1 Introduction

Managing a large portfolio of software products requires continuous monitoring of project status and process quality. Collecting the required data by regularly status reporting can be expensive and intrusive and furthermore ignores the past history of the process [1].

The routinely collected information during processing of enhancement requests and defect reports with a change request management system reflects many characteristics of process quality. However this data is currently only used in a very limited way for a systematic evaluation of the process. This is caused by methodological problems and lack of appropriate tool support. Existing change request management systems usually provide a set of fixed metric evaluations with limited adaptability, like specifying the considered product or time period [2]. Metrics of interest depend on the objectives of the organization [3]. In order to ensure that a metric is a correct numerical characterisation of the quality of interest, a metric definition must be validated and usually be refined. Thus metrics must typically be implemented in custom scripts [4], which is time-consuming and costly. On the methodological side, it is not clear how measurement values can be interpreted with respect to high-level quality characteristics of interest.

We have developed a new approach that is based on defining an organization-specific quality model representing a decomposition of the quality characteristics of interest, and their relations to metrics used as quality indicators [5]. Interpretation of the measurement values is guided by comparisons within a peer group of projects. Complementary tool support allows specifying metrics in a declarative language, which simplifies developing and validating new metrics [2].

This case study demonstrates the comprehensive application of the mentioned approach to a large portfolio of projects that follow a common development process. The objective of the exploratory case study is to quantitatively characterize the process quality, based on the analysis of change request data. We focus on the change request process of Eclipse as the object of study. The units of analysis are the projects under the Eclipse umbrella.

Related work will briefly be sketched in section 2. The development process of Eclipse is described in section 3. Questions regarding quality characteristics of the process are formulated in section 4. The evaluation approach is then illustrated in section 5, followed by the evaluation results for the Eclipse project in section 6. In section 7 we comment on threats to validity. Lessons learnt from this case study related to the applicability and limitations of the approach are discussed in section 8.

2 Related Work

There are numerous approaches that use routinely collected data available in version control systems or change request management systems in order to analyze different aspects of software evolution. A survey is given by Kagdi et al [6]. Since Eclipse is one of the most intensively studied open source development projects, many of these approaches are also applied in case studies related to Eclipse. The analyzed aspects encompass architectural evolution [7][8][9], communications patterns of developers and users via newsgroups [10], classification of developer contributions according to topics [11], and predictive models, e.g. for the number of changes [12], growth of defects [13], or bug lifetimes [14].

Some approaches target the analysis of specific aspects of the process, like developer contributions and defect density [15][16], performance characteristics of the bug fixing process [17][18], the frequency of fix-inducing changes [19], typical defect-lifecycles [20], or the usage of the bug reporting form [21][22].

Summarizing it can be stated that the mentioned approaches facilitate insights to isolated aspects of the process quality. Hence the related scripts and tools for data extraction are tailored for a specific change request management system and the research questions at hand. We do not know about a generalized approach for analyzing process quality characteristics based on change request data.

3 The Eclipse Development Process

In order to assess the process quality within Eclipse projects it must be clarified which are the underlying goals of the development process.

The Eclipse Foundation practices its own agile development approach that sets guidelines for the 105 projects that fall under the Eclipse umbrella[1]. Known as the "Eclipse Way" [23], it encompasses a set of complementary practices which target at achieving predictability and quality delivery on time. However most of the project culture is not documented in every detail, as the process is defined and evolved by the development teams of the various projects.

Community involvement should be fostered by timely feedback and reactions, and by a transparent development process. For example it must be published which features are available with a new milestone to create an incentive for users to move to this milestone build. The guidelines recommend a release cycle that is structured into milestones every 6 weeks, and the endgame - a stabilization phase before the final release.

Some more detailed advice is given on the usage of Bugzilla[2]. Incoming Change Requests (**CR**s) should be triaged at the start of each day. If further information on a CR is requested, and there is no response in a week, the CR should be closed with the status *Resolved/Invalid* or *Resolved/WontFix*. CRs that should get fixed in the current release should be marked with an appropriate target milestone. When a developer fixes a CR the status is set to *Resolved/Fixed*. A fixed CR should be assigned to another committer on the team to verify. When a project does a major release, the verified CRs are changed to *Closed*.

Additionally Eclipse has a process description[3] that defines the structure and organization of Eclipse projects, and the phases that projects undergo (*Proposal*, *Incubation*, *Mature*, and *Top-Level*). However this part of the process description does not give guidelines for the change request process.

4 Questions Addressed

As mentioned in the previous section, some of the main goals of the "Eclipse Way" of development are quality delivery on time, predictability, and promotion of community involvement. Based on these organization-specific goals we identified several **quality characteristics** related to steps in the change request process. They are illustrated in the following by one or several questions addressed.

A. Quality of the Reported CRs

Although the development team has only limited influence on the quality of CRs reported by general users, the quality of incoming CRs facilitates permits to draw conclusions on the competence and maturity of the user community of a product. So, we want to know:

- What is the quality of the reported CRs in terms of completeness, understandability, and redundancy freeness?

B. Quality of the CR Triage

---

1 *http://www.eclipse.org/projects/listofprojects.php*

2 *http://wiki.eclipse.org/Development_Resources/howto/Bugzilla_Use*

3 *http://www.eclipse.org/projects/dev_process/development_process.php*

Timely reactions and appropriate classification of incoming CRs influences the perception of the project by general users. Hence we derived these questions:

- How fast does the organization react on an incoming CR?
- Are the triaged CRs correctly classified?
- To which degree does the prioritization of defect reports take into account the perception of the severity by the user?

C. Quality of Planning

Predictability and transparency of the availability of new features are essential in order to motivate users to move to current milestone builds. On the level of change requests this manifests in assigning a target milestone for scheduled CRs:

- How many of the fixed CRs are planned with a specified target milestone?
- How often are target milestones changed? This usually means that a CR has been postponed to a later milestone.

D. Quality of CR Processing

Again a timely and correct resolution of CRs by the development team can foster community involvement and permits to draw conclusions on the internal quality of the product. Moreover we are interested in the frequency of problems during processing CRs, which can for example be indicated by many high priority defect reports, frequent assignee changes, or breaking the feature freeze during the endgame phase. All these problems cause unwanted friction or interruptions during development. Hence we consider these questions:

- How long does it take to fix a new CR?
- How often has a fixed CR to be reopened?
- How friction-less is the processing of CRs?

E. Quality of CR Verification

The process guidelines require fixed CRs to be verified by another committer. Hence we are interested in the quality of this verification process:

- How many of the CRs are explicitly verified?
- How often has a verified CR to be reopened?

5 Evaluation approach

Our approach is based on calculating metrics on the change request data that can be used as indicators for the quality characteristics of interest. To calculate metrics we applied the open source tool BugzillaMetrics [2]. It supports the specification of metrics in a declarative language. Thus metrics can be described precisely on a higher abstraction level, which simplifies the process of developing and validating metrics [5].

5.1 Metrics Used as Quality Indicators

Based on the questions formulated in section 4 we derived a number of corresponding metrics that are listed in table 1 with brief descriptions. Each metric is normalized

Table 1. Metrics used as quality indicators

Id	Metric	Description
A.1	Duplicated CRs	Number of CRs marked as *Duplicate* relative to the number of all resolved CRs in a time interval.
A.2	Invalid CRs	Number of CRs marked as *Invalid* relative to the number of resolved CRs in a time interval.
A.3	Defect reports without version	Number of reported defects with unspecified version number relative to the number of all reported defects in a time interval.
A.4	Comments before leaving status *New*	Average number of comments before a CR changes into status *Assigned* or *Resolved* for the first time.
B.1	CRs with no reaction within 2 days	Percentage of CRs created in a time interval where the first reaction takes longer than 2 days.
B.2	Reopened rate of rejected CRs	Number of triaged CRs with resolution *Duplicate*, *Invalid*, *NotEclipse, WontFix,* or *WorksForMe* that have been reopened, relative to the number of rejected CRs in a time interval.
B.3	Priority of severe bugs	Average priority of CRs with severity *critical* or *blocker* that had been resolved in a time interval.
C.1	Assigned without milestone	Number of CRs that change into status *Assigned* with no valid target milestone relative to the number of all CRs that change into *Assigned*.
C.2	Fixed without milestone	Number of CRs that are fixed and have no valid target milestone relative to the number of all CRs fixed in a time interval.
C.3	Frequency of milestone changes	Number of changes to defined target milestones relative to the number of CRs with a defined target milestone.
D.1	Time until fixed	Median age in days of CRs that change into the status *Resolved/Fixed*.
D.2	High Priority Lifetime Ratio	Average lifetime of fixed CRs with priority *P1* relative to the average lifetime of all fixed CRs.
D.3	Reopened Rate of fixed CRs	Number of fixed CRs that are reopened relative to the number of fixed CRs in a time interval.
D.4	High priority CRs	Number of fixed CRs with priority *P1* relative to the number of all CRs resolved in a time interval.
D.5	Average Assignee Changes	Number of assignee changes relative to the number of CRs assigned in a time interval.
D.6	Enhancements during Endgame	Number of enhancement requests fixed during the Endgame phase relative to the number of all enhancement requests fixed in the release cycle.
E.1	Closed/ Resolved Ratio	Number of closed CRs in a time interval relative to the number of resolved CRs in a time interval.
E.2	Closed without Verified	Number of closed CRs in a time interval that had not been in the status *Verified*.

such that its results are not directly dependent on factors like size or age of the product. Furthermore each metric is specified in a way such that minimal values are considered to be optimal. The precise and complete specification of each metric is made available on *www.bugzillametrics.org*.

These metrics can then be evaluated for a number of selected products and a given time interval. The value distribution of the results for each metric in a time interval gives an impression on how good the different products perform in general and how large are the differences between the products.

We will briefly delineate some of the underlying assumptions. Metric A.4 assumes that a CR with incomplete or vague information needs more comments until it can eventually be assigned or resolved.

The metrics B.1 – B.3 reflect that users expect timely and appropriate reactions on their change requests. Otherwise users will perceive that their feedback will not have any impact and stop providing valuable input. It was not possible to define a metric that counts false positive triage decisions, since a status like *Unconfirmed* is not used by the majority of the projects.

Metrics C.1 and C. assume that CRs with an undefined target milestone can be considered as lack of transparency. Metric D.4 reflects that the occurrence of high-priority CRs might interrupt work on other CRs. In D.5 it is assumed that problems like unclear responsibilities, overburdened developers, or vague requirements lead to frequent assignee changes of a CR.

5.2 Evaluation of Quality Characteristics

By means of these metrics we get basic data concerning the change request management process. Moreover we want to aggregate these raw results in order to assess the quality characteristics introduced in section 4. However defining thresholds for metric results in order to classify the quality would be an intricate task. The process description of the "Eclipse Way" of development does not impose absolute goals for the outcome of these metrics. Trying to define reasonable thresholds upfront is likely to be based on unrealistic assumptions. Thus we prefer to use the value distribution of the metric evaluated for a set of comparison projects as base for classifying metric results. The Eclipse projects itself in a selected time interval can for example be used as comparison data. This pragmatic approach facilitates to assess a quality characteristic of the development process of a certain project relative to other Eclipse projects. Moreover it can be analysed how quality characteristics evolved over time.

In order to specify the quality model, we used the QMetric quality model editor and evaluation tool [24]. The quality model defines how individual metric results are aggregated in order to assess a quality characteristic. The QMetric evaluation tool supports an automatic evaluation of the quality model based on results of a metric tool like BugzillaMetrics.

The evaluation of the quality characteristics is based on classifying each individual metric value according to the quartiles of the metric results for comparison products. Using quartiles also facilitates the application of the approach on a small number of comparison products. A linear equation is used to aggregate the results. In detail this can be defined as follows:

Let

C_m be a set of values for a given metric m measured for a number of products used as comparison data,

$Q_i(C_m)$ be the i-th quartile of C_m, $i = 1..3$.

The quartile classification q of a metric value v_m with respect to the corresponding comparison data C_m is defined as:

$$q(v_m, C_m) = \begin{cases} 1 & Q_3(C_m) < v_m \\ 2 & Q_2(C_m) < v_m \leq Q_3(C_m) \\ 3 & Q_1(C_m) < v_m \leq Q_2(C_m) \\ 4 & v_m \leq Q_1(C_m) \end{cases} \quad (1)$$

A quality characteristic QC with underlying metrics $m_1, \ldots, m_n$ can then be evaluated as:

$$e(QC) = \frac{1}{n} \sum_{i=1}^{n} q(v_{m_i}, C_{m_i}) \,. \quad (2)$$

Hence the evaluation of a quality characteristic is normalized to a number between 1 and 4 with the following interpretation:

- 4 indicates that the considered product performs better than 75% of the products used as comparison data for each of the underlying metrics
- 1 indicates that the quality is poorer than in 75% of the compared products
- 2.5 can be interpreted as average quality.

The complete quality model is shown in Figure 1. In general the quality model can be a DAG. The sink nodes and inner nodes of the DAG represent quality

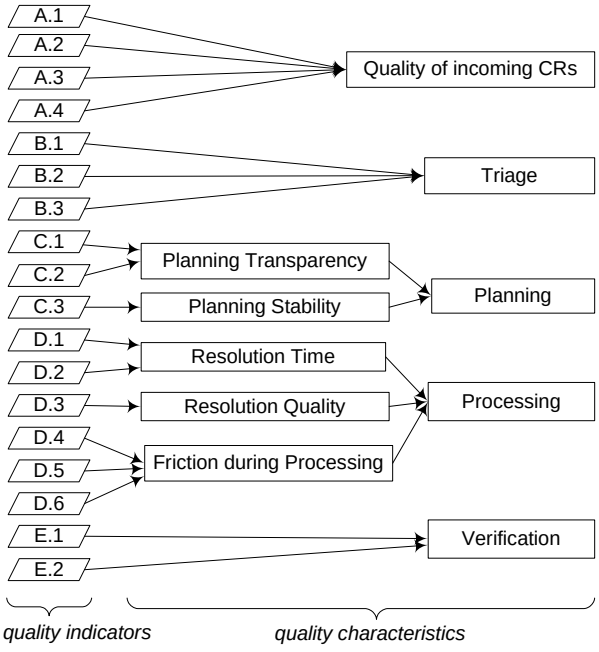

Figure 1. Eclipse process quality model

characteristics. Source nodes represent quality indicators. Each quality indicator includes a metric specification, and some guidance for the interpretation of the metric result. In the given model the metric results are classified according to the quartiles of the value distribution of comparison projects.

The quality characteristics introduced in section 4 are partly refined into sub characteristics, and finally led back to a number of quality indicators. The evaluation of the each quality characteristic is normalized to the interval 1-4. In the given model each incoming quality indicator is equally weighted, as the "Eclipse Way" development process does not impose a definitive precedence of the considered aspects. Of course the quality model editor supports to express more refined models, e.g. a weighting of the different metrics, or finer structuring of quality characteristics as tree or DAG.

Table 2. Value distribution of the metric results for the products of the Ganymede release (July 1, 2007 – June 30, 2008)

Metric Id	Unit	Minimum	Lower Quantile	Median	Upper Quantile	Maximum
A.1	%	1,01	4,27	9,17	13,07	19,64
A.2	%	0,53	3,03	4,64	6,03	9,17
A.3	%	0,00	7,71	21,94	53,31	100,00
A.4	comments/CR	0,58	1,27	2,51	2,99	4,98
B.1	%	11,41	20,97	30,28	36,73	48,30
B.2	%	0,00	2,70	7,46	11,76	26,67
B.3	priority P1-P5	1,15	2,54	2,92	3,00	3,00
C.1	%	3,70	55,40	73,97	90,41	100,00
C.2	%	1,26	23,04	41,17	69,34	100,00
C.3	milestone changes/CR	0,00	0,08	0,44	0,64	2,31
D.1	days	0,86	4,63	9,26	17,14	48,66
D.2	ratio	0,00	0,00	0,31	0,86	2,90
D.3	%	0,00	3,80	6,36	8,64	18,16
D.4	%	0,00	0,12	1,06	4,41	57,29
D.5	assignee changes/CR	1,00	1,06	1,16	1,27	1,55
D.6	%	0,00	0,00	0,00	0,01	0,07
E.1	%	0,34	2,66	5,40	23,12	78,23
E.2	%	50,00	82,19	92,98	98,49	100,00

6 Evaluation Results

Most of the mature Eclipse projects ship a major release each year and take part in a coordinated simultaneous release at the end of June. Thus we consider these release cycles as time intervals for the evaluation. As comparison group we selected all 29

products that took part in the latest simultaneous release[4] in June 2008. The corresponding value distribution is given in table 2. Metric Ids refer to the metrics given in table 1.

The value distribution already allows drawing conclusions on the projects. We will briefly point out some observations. A relatively high percentage of the incoming defect reports lack a version number (A.3). It takes typically around 2.5 comments until a CR leaves the status *New* (A.4). *TPTP* contributes the maximum value, thus usually more discussion is needed in this project.

In most of the projects the percentage of CRs with the first reaction later than two days is around 30% (B.1). Most of the projects have a reopened rate of rejected CRs higher than 7% (B.2). For half of the projects the average priority of severe CRs is near the default *P3* (B.3). Thus priority does not correlate with the severity specified by the user.

Setting a target milestone to inform about the availability of new features is rather neglected by most of the developers (C.1, C.2). CRs are relatively often postponed to another milestone (C.3).

The median time until a CR varies around 10 days for most of the projects (D.1). At first sight the minimum value of less than a day looks astonishing. Detailed analysis shows that in the related project *M2M* some committers use Bugzilla to keep log of their day-to-day work, such that a high number of CRs is reported and resolved by the same person just on the same day. The median value for metric D.2 indicates that it typically takes only one third of the time to fix a CR with priority *P1*, compared to the resolution time of all CRs. However there are some projects (e.g. *BIRT* and *Eclipse Platform*) where it takes more than twice as long to resolve *P1* CRs. Hence the high-priority CRs in these products seem to be rather intricate tasks. Most of the projects have a reopened rate of fixed CRs higher than 6% (D.3).

Most of the projects have a low percentage of CRs processed with a high priority (D.4). The maximum value for metric D.4 is contributed by *TPTP Profiling*. A detailed analysis shows that most of the CRs for *TPTP Profiling* had been escalated to P1 before eventually being fixed.

Assigning CRs to responsible assignees seems to run relatively smoothly (D.5). Adding enhancements after the feature freeze is very rare (D.6). Due to the uneven spread of the resulting values, it is inappropriate to use them as comparison data. Thus we dropped D.6 from the quality model.

None of the projects does consistently use the status values *Verified* and *Closed* (E.1 and E.2). If a CR enters the status *Closed* it has most often not been explicitly verified before. Since most of the projects obviously neglect using the *Verified* state, it is not possible to draw further conclusions about the verification process.

In the next step we will focus on the evolution of the quality characteristics over time. The aggregated results of the release cycles 2004 – 2008 are shown in Figure 2. For the sake of readability we concentrate on 9 projects that are developed since at least 2003.

The quality of the incoming CRs remains relatively stable for most of the projects (Figure 2 A). In some projects like *EMF* and *Equinox* the triage process was improved in the last years, while it worsened in the *GMF* project (Figure 2 B). Quality of the

4 *http://wiki.eclipse.org/Ganymede_Simultaneous_Release*

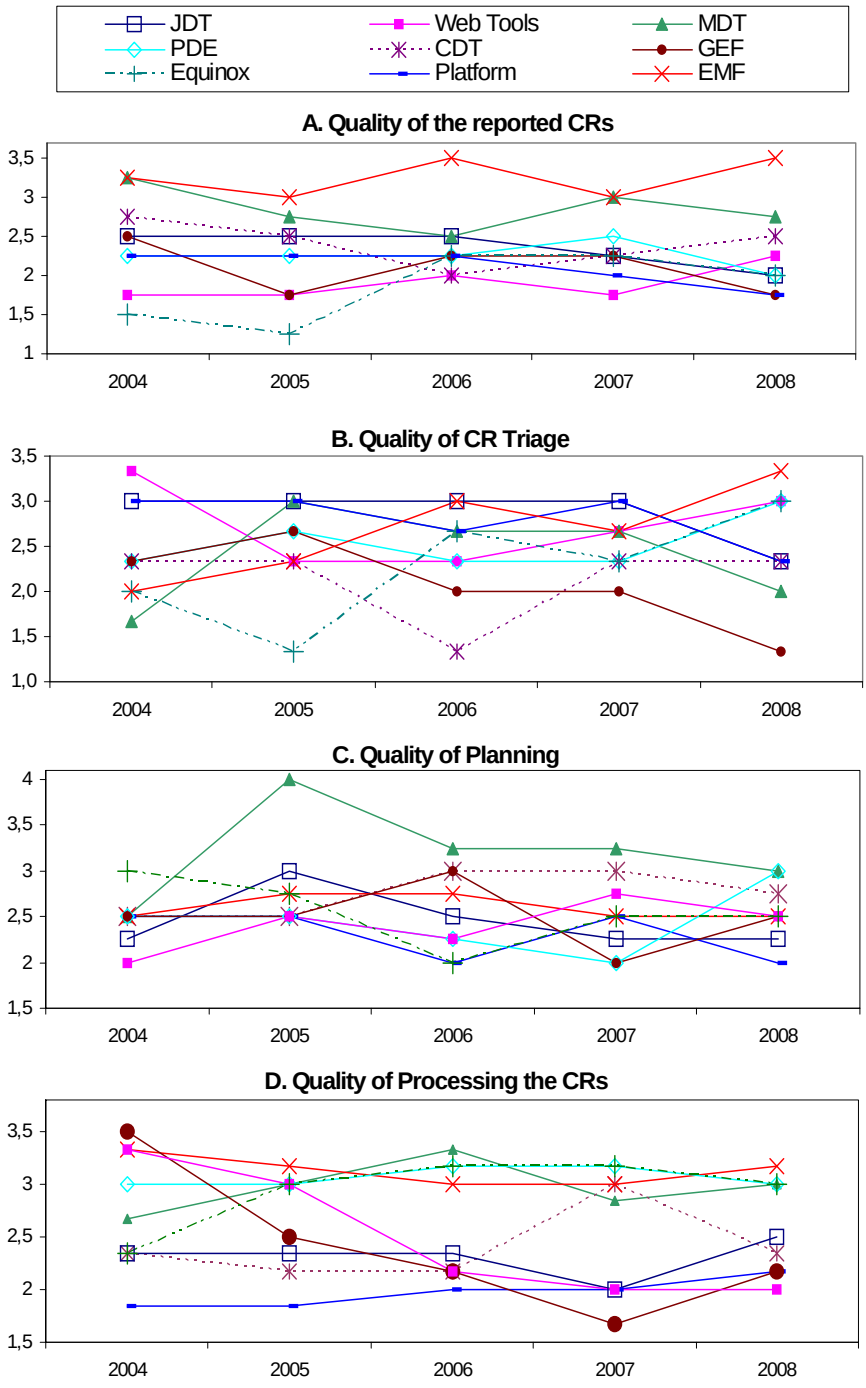

Figure 2. Evolution of quality characteristics in the major releases 2004-2008.

planning process is near the average value of 2.5 in most of the selected projects (Figure 2 C). Planning in *MDT* is notably better. Regarding the quality of processing the CRs, a group of projects (*MDT, EMF,* and *PDE*) maintains a good quality for all the considered releases. Processing quality has decreased gradually in *GEF* and *WebTools* (Figure 2 D). In order to validate these interpretations of the results of the quality model, they have to be compared to expert opinions on the considered projects.

Based on the experience gained during the development of an *EMF* and *GEF* based toolset at our research group[5], we can confirm the results for these projects. CRs filed for *EMF* had been quickly responded, and fixed timely. In *GEF* we could increasingly observe no, late or inappropriate reactions on reported CRs. Additionally it can be stated that *GEF* has provided few major new features in the Europa and Ganymede release, and concentrated more on fixing defects.

An exhaustive validation of the results would require in-depth experience with all considered projects.

The performance in the four categories A-D is quite heterogeneous between all projects. Thus it was not reasonable to aggregate the results to some general process quality, or to distinct between different maturity levels of the projects.

7 Threats to Validity

As this case study did not examine causal relationships we focus on threats to the construct validity. In general the following threats to construct validity exist for the described approach:

Data quality: The Bugzilla database can reveal some inconsistencies e.g. due to maintenance like renaming or restructuring of products, or importing data from other Bugzilla instances. This affects the product *Mylyn* which was renamed (previously *Mylar*), and *TPTP* which was restructured into several products within Bugzilla. In this case those CRs, which had been moved between products, can not always be correctly associated to the original product.

Validity of the underlying metrics: It must be carefully validated that each metric is a proper numerical characterization of the qualities of interest, and that the measurements can be compared between different products. To ensure this, we applied a systematic stepwise validation approach [5]. It includes inspecting measurement results for sample CRs in order to find out whether the intended interpretation of a measurement value matches with the interpretation of the events happened during the lifecycle of the CR.

Homogeneity of Bugzilla usage: The interpretation of different CR attributes can deviate between different products. We tried to base the metrics on CR attributes with a commonly accepted interpretation. Results can also be distorted if issues are reported on other channels, like mailing lists. However this situation should be rare, as users are explicitly requested to use Bugzilla[6].

---

5 ViPER: Visual Tooling Platform for Model-Based Engineering. *http://www.viper.sc*

6 *http://www.eclipse.org/mail/*

8 Discussion

In the following we first discuss observations related to the process quality of Eclipse projects and second, the lessons learnt related to the evaluation approach.

8.1 Process Quality in Eclipse

The analyzed quality characteristics remain relatively stable for most of the projects, or changes only gradually. This matches with the experience that discontinuous improvements can only seldom be achieved for large projects.

There is no statistically significant correlation between the different considered quality characteristics of the projects.

Further on one might ask the question whether projects in the *mature* phase perform better than those in the *incubation* phase. This is generally not the case. However some differences can be observed. Incubating projects have typically fewer duplicate or invalid CRs and less discussion until assigning a CR. Transparency with respect to of setting target milestones is lower than in mature projects. Reopened rate and the age of fixed CRs are lower for the incubating projects, while friction during processing is similar in both groups.

While existing analyses based on change request data typically focus on evaluating a selected aspect of the process quality (see section 2), the presented study offers a broader view on a range of quality characteristics.

As the assessment is based on comparisons to real projects, careful analysis of the evaluation results for a single project can give valuable advice on realistic potential for improvement.

8.2 Evaluation Approach

The case study confirms that quality models have to be organization- or even product-specific. Even if one relies on the default status workflow that comes with Bugzilla, there are different schemes of using Bugzilla, which have to be reflected in the quality model and its underlying metrics.

The specification of metrics in a declarative language enables to describe each metric in a compact form, and forces to deal with details of the metric definition that would often be overlooked when using a structured natural language description. A lesson from the case study is that the definition of the metrics is often an intricate task. When defining metrics, not only the current usage scheme of the change request workflow has to be considered, but also its evolution in the past. An example is that in the past it was possible to set a CR to the status *Resolved* with resolutions called *Later* and *Remind.* Since this does not indicate that the CR had really been resolved, these field values had been deprecated[7]. Instead such CRs should be marked either by

[7] *http://wiki.eclipse.org/Bug_Reporting_FAQ*

setting a target milestone named *future*, by adding the *needinfo* keyword, or by decreasing their priority. When counting status transitions to *Resolved* the CRs with resolutions *Later* and *Remind* must therefore be excluded.

Moreover the process documentation is not fully consistent and up to date, as the Bugzilla usage guide[8] still recommends setting the resolution *Later* for CRs that will not be fixed in the current release.

Another difficulty for the definition of appropriate metrics is the inconsistent use of Bugzilla fields by different developers. An example is the priority field. While the Bugzilla usage guide[7] recommends using the priority independent from the severity field, some developers use the priority to subclassify the severity field 21. If this usage pattern is applied for many CRs, an underlying assumption of metric D.4 is broken. Thus metric results are potentially distorted. It remains difficult to prevent such conditions when defining metrics, as these usage patterns can often not be differentiated in retrospective.

However a detailed analysis of the metric results can unveil the existence of such different usage schemes, and direct the attention to increase the uniformity of the usage scheme. Lack of uniformity is not only a problem when evaluating the process quality; it is probably also confusing if CRs are passed from one developer to another.

Generally the assessment of quality characteristics is limited to those aspects that are made visible in the CR database. An example is the verification of resolved CRs. As the corresponding status transition is not used by most of the projects, it is not possible to draw substantive conclusions on the quality of the verification process.

9 Summary and Outlook

This case study presents the derivation of a quality model for process characteristics of Eclipse projects. It is based on evaluating metrics on the change request data, and classification of metric results according to the value distribution of a set of comparison projects.

The usage of declarative metric specifications is essential for the practical application of the approach, as it simplifies development and validation of metrics. Major attention to the details of the underlying metric definitions is required in order to achieve interpretable results. A prerequisite is to have at least some degree of uniformity in the usage scheme of the change request system.

A generalized quality model can be complementary to approaches for quality evaluation of open source projects [25][26].

Change request systems in an industrial environment usually have more fine-grained workflow definitions and collect more detailed information, like estimated and actual effort, or information on scheduled deadlines. Thus the proposed approach can be applied for a wider range of quality characteristics. An evaluation based on an appropriate quality model can help to identify weaknesses in the development process, and improve transparency in order to support planning and resource allocation.

---

8 *http://wiki.eclipse.org/Development_Resources/howto/Bugzilla_Use*

References

1. Cook, J. E., Votta, L. G., Wolf, A. L.: Cost-Effective Analysis of In-Place Software Processes. IEEE Trans. Softw. Eng. 24, 8 (Aug. 1998), pp. 650-663 (1998)
2. Grammel, L., Schackmann, H., Lichter, H.: BugzillaMetrics: An Adaptable Tool for Evaluating Metric Specifications on Change Requests. In Ninth Intl. Workshop on Principles of Software Evolution (Dubrovnik, Croatia, Sep. 03 - 04, 2007). IWPSE '07, pp. 35-38. ACM, New York (2007)
3. Ebert, C., Dumke, R.: Software Measurement. Establish – Extract – Evaluate - Execute. Springer, Berlin, Heidelberg (2007)
4. Kanat-Alexander, M.: The Bugzilla Survey – August 2008. http://wiki.mozilla.org/Bugzilla:Survey (2008)
5. Schackmann, H., Lichter, H.: Comparison of Process Quality Characteristics Based on Change Request Data. In Proc. of the Intl. Conf. on Software Process and Product Measurement (Munich, Germany, Nov. 18 - 19, 2008). R. R. Dumke et al., Eds. LNCS 5338, pp. 127-140. Springer, Berlin, Heidelberg (2008)
6. Kagdi, H., Collard, M. L., Maletic, J. I.: A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution. J. Softw. Maint. Evol. 19, 2 (Mar. 2007), pp. 77-131. (2007)
7. Breu, S., Zimmermann, T., Lindig, C.: HAM: Cross-Cutting Concerns in Eclipse. In Proc. of the 2006 OOPSLA Workshop on Eclipse Technology Exchange (Portland, Oregon, Oct. 22-23, 2006), pp. 21-24. ACM, New York (2006)
8. Hou, D.: Studying the Evolution of the Eclipse Java Editor. In Proc. of the 2007 OOPSLA Workshop on Eclipse Technology Exchange (Montreal, Quebec, Canada, Oct. 21, 2007), pp. 65-69. ACM, New York (2007)
9. Wermelinger, M., Yu, Y.: Analyzing the Evolution of Eclipse Plugins. In Proc. of the 2008 Intl. Working Conf. on Mining Software Repositories (Leipzig, Germany, May 10-11, 2008), pp. 133-136. ACM, New York (2008)
10. Kidane, Y., Gloor, P.: Correlating Temporal Communication Patterns of the Eclipse Open Source Community with Performance and Creativity. In Proc. of NAACSOS 2005 (Notre Dame, Indiana, June 26-28, 2005) (2005)
11. Linstead, E., Rigor, P., Bajracharya, S., Lopes, C., Baldi, P.: Mining Eclipse Developer Contributions via Author-Topic Models. In Proc. of the Fourth Intl. Workshop on Mining Software Repositories (Minneapolis, MN, May 19-20, 2007), p. 30. IEEE, Washington, DC (2007)
12. Herraiz, I., Gonzalez-Barahona, J. M., Robles, G.: Forecasting the Number of Changes in Eclipse Using Time Series Analysis. In Proc. of the Fourth Intl. Workshop on Mining Software Repositories (Minneapolis, MN, May 19-20, 2007), p. 32. IEEE, Washington, DC (2007)
13. Zhang, H.: An Initial Study of the Growth of Eclipse Defects. In Proc. of the 2008 Intl. Working Conf. on Mining Software Repositories (Leipzig, Germany, May 10-11, 2008), pp. 141-144. ACM, New York (2008)
14. Panjer, L. D.: Predicting Eclipse Bug Lifetimes. In Proc. of the Fourth Intl. Workshop on Mining Software Repositories (Minneapolis, MN, May 19-20, 2007), p. 29. IEEE, Washington, DC (2007)
15. Schuler, D., Zimmermann, T.: Mining Usage Expertise from Version Archives. In Proc. of the 2008 Intl. Working Conf. on Mining Software Repositories (Leipzig, Germany, May 10-11, 2008), pp. 121-124. ACM, New York (2008)
16. Mockus, A., Fielding, R. T., Herbsleb, J. D.: Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Trans. Softw. Eng. Methodol. 11, 3 (Jul. 2002), 309-346. (2002)

17. Francalanci, C., Merlo, F.: Empirical Analysis of the Bug Fixing Process in Open Source Projects. Open Source Development, Communities and Quality, pp. 187-196. Springer, Boston (2008)
18. Michlmayr, M., Senyard, A.: A Statistical Analysis of Defects in Debian and Strategies for Improving Quality in Free Software Projects. In: Bitzer, J., Schröder, P. J. H. (Eds.), The Economics of Open Source Software Development. pp. 131–148. Elsevier, Amsterdam (2006)
19. Śliwerski, J., Zimmermann, T., Zeller, A.: When Do Changes Induce Fixes? In Proc. of the 2005 Intl. Workshop on Mining Software Repositories (St. Louis, Missouri, May 17, 2005), pp. 1-5. ACM, New York, NY (2005)
20. Koponen, T: RaSOSS - Remote Analysis System for Open Source Software. In Proc. of the Intl. Conf. on Software Eng. Advances (Papeete, Tahiti, French Polynesia, Oct. 29 – Nov. 03, 2006), pp. 54-59. IEEE, Washington, DC (2006)
21. Herraiz, I., German, D. M., Gonzalez-Barahona, J. M., Robles, G.: Towards a Simplification of the Bug Report Form in Eclipse. In Proc. of the 2008 Intl. Working Conf. on Mining Software Repositories (Leipzig, Germany, May 10-11, 2008), pp. 145-148. ACM, New York (2008)
22. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T.: What Makes a Good Bug Report? In Proc. of the 16th ACM SIGSOFT Intl. Symp. on Foundations of Software Engineering (Atlanta, Georgia, Nov. 09-14, 2008), pp. 308-318. ACM, New York (2008)
23. Gamma, E.: Agile, Open Source, Distributed, and On-Time – Inside the Eclipse Development Process. Keynote Talk, 27th Intl. Conf. on Software Engineering., 15-21 May 2005, St. Louis, Missouri (2005)
24. Schackmann, H., Jansen, M., Lischkowitz, C., Lichter, H.: QMetric - A Metric Tool Suite for the Evaluation of Software Process Data. In Companion Proc. of the 31th Intl. Conf. on Software Engineering (Vancouver, Canada, May 16-22, 2009), pp 415-416. ACM, New York (2009)
25. Samoladas, I., Gousios, G., Spinellis, D., Stamelos, I.: The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation. In Open Source Development, Communities and Quality (Milano, Italy, Sep. 7-10), IFIP vol. 275, pp. 237-248. Springer, Boston (2008)
26. Ciolkowski, M., Soto, M.: Towards a Comprehensive Approach for Assessing Open Source Projects. In Proc. of the Intl. Conf. on Software Process and Product Measurement (Munich, Germany, Nov. 18-19, 2008). R. R. Dumke et al. Eds., LNCS 5338, pp. 316-330. Springer, Berlin, Heidelberg (2008)