BugzillaMetrics - Design of an adaptable tool for evaluating user-defined metric specifications on change requests

Lars Grammel, Holger Schackmann, Horst Lichter

RWTH Aachen University – Research Group Software Construction

lars.grammel@googlemail.com, {schackmann, lichter}@cs.rwth-aachen.de

Abstract:

The evaluation of metrics on the data available in change request management (CRM) systems can give valuable information for the management of software development. It can for example be helpful in assessing the current workload, product quality or development process weaknesses.

Metrics and charts on change requests are already available in current CRM systems. They provide information about common metrics, but their adaptability is limited with respect to the specification of metrics customized to organization-specific needs.

This paper describes a more flexible approach for the evaluation of metrics on change requests. The core part of the presented tool is an event driven evaluation algorithm for the calculation of time series data. It is parametrized with user defined metric specifications. This enables a separation between metric specification and information retrieval. Further design decisions enable a transparent execution optimization and an abstraction from the data sources of the underlying CRM database.

Keywords

Process metrics, change request management, metrics specification

1 Introduction

To manage the evolution of software processes and products, it is essential to evaluate their current state and how it evolved. This information can be obtained by analyzing the data that is available in change request management (**CRM**) systems like Bugzilla.

The evaluation of metrics on this data can be used for several purposes:

- Evaluation how the CRM system is used. This means measuring the quality of the data collected in the CRM system.
- Improvement of awareness and monitoring of current project states. Typical measurement categories are the workload and the product quality.
- Identifying software development process weaknesses. This includes measuring the software development process quality and speed.

MetriKon 2007 - Praxis der Software-Messung, in Büren, Bundschuh, Dumke (Hrsg.)
Tagungsband des DASMA Software Metrik Kongresses MetriKon 2007, 15.-16. November 2007, Kaiserslautern, Shaker Verlag, Aachen, ISBN 978-3-8322-6703-2, pp 201-218

- Assessing whether development process changes improved the process or not. Measurements required to answer this question are the workload, the product quality and the software development process speed and quality.

It depends on organization-specific circumstances and goals which metrics are of interest. Furthermore CRM systems used in a commercial context must typically be adapted to and extended for the specific needs of the organization.

But the capabilities to define metrics in existing CRM tools are restricted. Typically only a fixed set of common metrics is offered. Adaptability is limited with respect to the definition of new metrics, and with respect to the adaption to customizations of the underlying database scheme. Thus, there is a need for a tool that supports a flexible definition of metrics and can easily be adapted to changes in the underlying CRM system.

The tool BugzillaMetrics presented in this paper should fulfil these requirements [16]. By analyzing a broad range of metrics on change requests, the variation points within the calculation of these metrics were identified. This served as a basis for the design of an evaluation algorithm that offers a flexible way to specify a metric by configuring given calculation elements. Thereby the metric specification is separated from the way the required information is retrieved. Further on an abstraction layer separates the information requirements for the evaluation algorithm from the way the data is stored in the CRM database.

The tool was developed and used in cooperation with Kisters AG [5]. This company offers a large portfolio of software products that are developed in software product lines [10]. Kisters AG uses a modified Bugzilla installation, for example realizing an extended state management.

The paper is structured as follows: First, a survey of existing CRM tools is presented. Then, an overview of the requirements and the architecture of the tool are given, and the evaluation algorithm is described in detail. Afterwards, an example is shown and results of using the tool are presented.

2 Survey of Existing Tools

Metrics and charts on Change Requests (**CRs**) are already available in current CRM systems. They mostly provide common metrics and are furthermore limited in their adaptability. The development of BugzillaMetrics was based on the evaluation and comparison of the following CRM tools:

- Bugzilla [1] is an open-source issue tracker. Version 2.18.6 was evaluated.
- JIRA [4] is a commercial issue tracker. Version 3.7 was evaluated.
- Polarion for Subversion [6] is a commercial application lifecycle management tool. It integrates requirements, change and project management tools

and provides real-time visibility of the development status. Version 2.6.0 of Polarion was evaluated.

- Code Beamer [3] is a commercial collaborative development platform with application life cycle management features. Code Beamer 4.2.1 was evaluated.

There exists also a number of approaches in research to analyze CRM data for several purposes (e.g. visualization of software evolution [15, 12], or change impact analysis [8, 9]). Some approaches try to integrate information from different sources like source code repository, change request database and mailing-lists (e.g. to retrieve related change requests [11] or to obtain feedback for software process improvement [18]). Moreover [19] describes an adaptable architecture for the mining of source code repositories. Nevertheless we do not know about any approach for flexible metric definition on change request data.

In the following subsections, the relevant results of the evaluation and analysis are presented.

2.1 Change Request Filtering

The search functionality provided by the different tools ranges from a simple keyword search in a fixed set of properties to the specification of the `WHERE` part of an SQL select statement.

Whereas a simple keyword search is not sufficient for selecting a specific set of CRs, SQL statement parts are on a level of abstraction that is too technical to be used by project or product managers. The most important intermediate level search features are:

- Text fields can be searched by simple keywords, wildcards and fuzzy search. Further options for text field searches are starts-with search and equality search, as well as search by regular expressions.
- Fields of an ordered type like text fields, date fields and number fields can be searched by range search, as well as by comparison with other values.
- Fields with a finite, fixed set of values can be searched by sets of these values.
- The basic search terms can be grouped and combined with the boolean operators `AND`, `OR` and `NOT`.
- CRs can be searched by their creation and modification dates especially by searching CRs with modification or creation dates that are contained in certain time periods.
- CRs can be searched by their field changes. The original or new value that is searched can be specified for such a filter.

2.2 Reports

An important distinction for reports is the distinction between snapshot and time series reports. Snapshot reports calculate values for a specific point of time, most often the current date. Time series reports calculate values at different time points in a time period. The different time points in the time period are usually calculated by splitting it into time intervals of a fixed duration and using the end points of the intervals as time points.

Because time series reports can be modelled as the aggregation of multiple snapshot reports with the same calculation that is cumulative or not, it is only necessary to evaluate snapshot reports, as they can be extended into time series reports.

The following types of snapshot reports can be created by the evaluated tools:

- Splitting. According to some criteria, the CRs are split into groups. The size of the groups is calculated and displayed in the snapshot report. Splitting includes counting the number of all CRs as a special case, namely splitting into one group.

  Examples for these kinds of reports are the number of created CRs, the number of closed CRs, the state distribution of open CRs, or the priority distribution of all CRs. The splitting can be multi-dimensional, for example splitting into priority and status.

- Age based reports. Age based reports like the resolution time of closed CRs or the average age of open CRs require the calculation of the length of a time period.

- Workload reports. The workload reports are based on the amount of time spent on CRs and the original and remaining estimations on how much time must be spent to resolve a CR.

- Derived reports. Reports that calculate derived values based on other calculations, like the original effort estimation accuracy.

Reports usually have a basic filter that determines the set of CRs that are evaluated. The basic filter can be just the selected project, but it can be an advanced filter, too.

Another important parameter is the time granularity. It determines the length of the time period for the snapshot evaluation, for example, that all created CRs on one day should be counted.

2.3 Limitations

There are several limitations in the evaluated tools. Users can only use a fixed set of evaluations with slight modifications like the evaluated timespan.

Metric developers can only change details through the user interfaces. If they want to create new metrics, they have to implement these by themselves, if the software is available in source code or offers an appropriate interface, or they have to rely on the CRM vendor. Thus, developing and experimenting with new metrics is difficult.

Another limitation is the problem of comparing metric results from different CRM systems. Slight differences in the metric implementations lead to difficulties in the comparison of results for similar metrics, if similar metrics are available, which is often not the case. The comparison problems can result in a loss of historical data when changing the CRM system.

3 Requirements and Architecture

In this section, we first introduce the basic terms, then the requirements for BugzillaMetrics are described and an overview of the architecture is given. In the next section, the evaluation algorithm is examined more in detail.

3.1 Basic Terms

A **change request** is a request to extend or modify a software system. CRs can be feature requests or bugs. This is determined by the **CR type**. The CR state is the property configuration of a CR at a certain point of time.

One property of a CR is the **CR status**, which models the processing state the CR is currently in. Commonly used CR status values are ‘new’, ‘assigned’, ‘resolved’, ‘closed’ and ‘reopened’. Other **CR properties** are for example the assignee, product, priority, severity and actual effort properties.

In this paper, an **event** is an occurrence of a change in the history of a CR, or a change of the evaluated time interval. Examples for such events are the creation of a CR, or a change of the assignee of a CR.

An **event filter** is a filter that accepts or denies events. For example, an event filter can be defined, that only accepts events which model assignee changes. A **state filter** is a filter that accepts or denies CRs based on their state. For example, a state filter that accepts all CRs which belong to a certain assignee can be defined.

A **CR value** is a numerical value that is assigned to a CR as the result of the evaluation of an event by a CR value calculator. An example for a **CR value calculator** is the incoming rate: all case creation events are counted in a CR value with the default weight of 1, whereas other events are not counted.

3.2 Requirements

The overall development goal of BugzillaMetrics was to overcome the limitations of the existing tools concerning metric definition and evaluation.

Metric developers should concentrate on the metric models and be able to test new metrics quickly. Therefore the tool should provide a mechanism for the flexible specification of a wide range of metrics. Furthermore, it should have a modular architecture that supports extensions and modifications.

Regarding the difficulties comparing data from different CRM systems, the tool should provide a mechanism that separates the metric evaluation from the data source access. By means of such an architecture, different CRM systems can be configured as data sources and the same metric can be evaluated on their data, provided the data required for the metric is available.

3.3 Variation Points

Several variation points where the evaluation algorithm is likely to be extended were identified during the iterative development of BugzillaMetrics. They are examined in the following, grouped by their likelihood to change.

The following aspects are very likely to change:

- **Weights** are used in the calculation of CR values on certain events. New weights are likely to be added when new calculations for CR values are required.
- **Data sources** are likely to be adapted to a changed database scheme when the tool is ported to a new version of the CRM system or a different CRM system.

The following aspects are somewhat likely to change:

- New **CR state filters**, for example to support range checks on number or date fields, might be added.
- **Events** and the corresponding **event filters** might be added when new data sources are added.
- **Group calculation operations** to support more complex statistical or mathematical operations might be added to support new metric calculations.

The following aspects might change, although they are expected to be stable:

- New **CR value calculators** (see subsection 4.3) might be added to support calculations that are out of the scope of the available calculators.
- **Groupings**. The way the results are grouped might be changed. This includes changing the order and the available group parameters.

BugzillaMetrics is designed in a way that allows these variation points to change without affecting the evaluation algorithm and the data structures. This was achieved by concentrating the complete algorithm configuration in a configuration

part and restricting the dependencies of the evaluation algorithm to the interfaces of the variation points, not their implementations.

3.4 General Architecture

The tool architecture defines three different components (see Figure 1):

- Core component. The core component contains the evaluation algorithm which calculates the metric result for a metric specification. Both the metric result and the metric specification are XML documents.
- Chart component. The chart component creates chart images from the metric results of the core component and an XML chart specification.
- Web-Frontend. The web frontend provides a user interface for interaction with the core and chart components.

BugzillaMetrics is integrated in an existing environment with a Bugzilla installation. The core component uses the Bugzilla database to retrieve the values for the CRs. The database is accessed for reading purposes only. The users access the web frontend to get the results for their metrics.

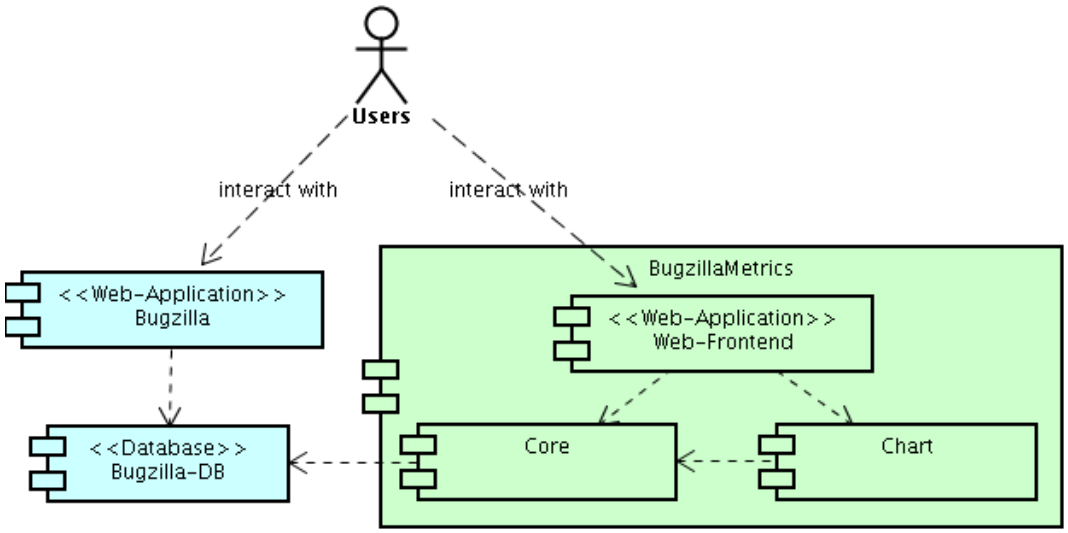

Figure 1: Integration of the architecture in the existing environment

4 The Evaluation Algorithm

In this section, the most important parts of the evaluation algorithm are outlined. The main characteristics of the algorithm are a flexible parametrization mechanism, an event driven design that calculates time series data, transparent execution optimization and an abstraction of the data sources.

4.1 Overview and Steps

The evaluation algorithm can be divided into a sequence of the following main steps:

1. The XML metric specification is parsed and the object structure of the metric calculation is configured. This step includes the configuration of the CR value calculators and the reading of the group parameters and basic filters.
2. It is calculated which information is required for the metric evaluation.
3. Objects for the CRs that are included in the basic filter are created and initialized with the current values for the required fields.
4. All CR values are calculated by processing the event sources and calling the configured CR value calculators (see Subsection 4.3) with the created events. The CR values are classified in a tree structure that is similar to the result structure.

   The event source processing creates an event stream that starts with the newest relevant event and goes back in time to the oldest relevant event. The CR states are updated accordingly, so each CR has the state it had when the event occurred.
5. The group values for the CR values created in the previous step are calculated by calling the group value calculators with the bottom layer of the tree structure that contains the CR values.
6. The XML result element for the group values that are stored in the CR value container tree is created.

4.2 Parametrization

The evaluation algorithm (see Figure 2) can be parametrized in several ways by the metric specification:

- The **basic filter** determines which CRs will be considered in the evaluation. The basic filter is a state filter.
- **State filters** provide a configurable filtering on CR states like the product, assignee, component and so on. They can be combined with `AND` and `OR` expressions.
- **Event filters** provide a configurable filtering of events. Event filters are used to configure how CR value calculators react on events.

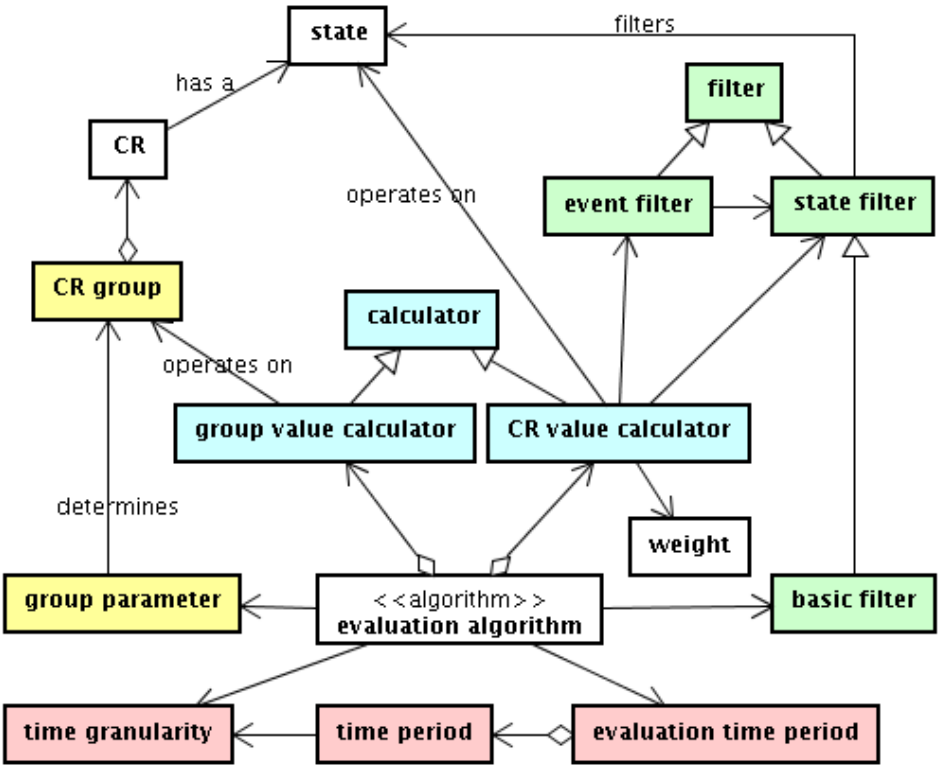

Figure 2: Parametrization of the evaluation algorithm

- The **group parameter** determines how the evaluated CRs will be partitioned into CR groups before the group themselves are evaluated. Examples are product, component or priority. A nested grouping is possible, for example group by products on the first level and by their components on the second level.
- A **CR value calculator** determines how the value for a CR on a certain event is calculated. The algorithm can be parametrized with more than one CR value calculator. There are different types of CR value calculators, which use state filters, event filters and weights. Weights are calculations based on the state of CRs on a certain event.
- **Group value calculations** determine how the result value for a group is calculated from the results of the CR value calculators for the CRs in that group. The results from the CR value calculators can be combined using mathematical operations, both operations that work on sets of CR values like sum and common operations like division. The algorithm can be parametrized with more than one group value calculation.
- **Evaluation time period** and **time granularity**, i. e. the time periods for which group values are calculated, for example weeks or months.

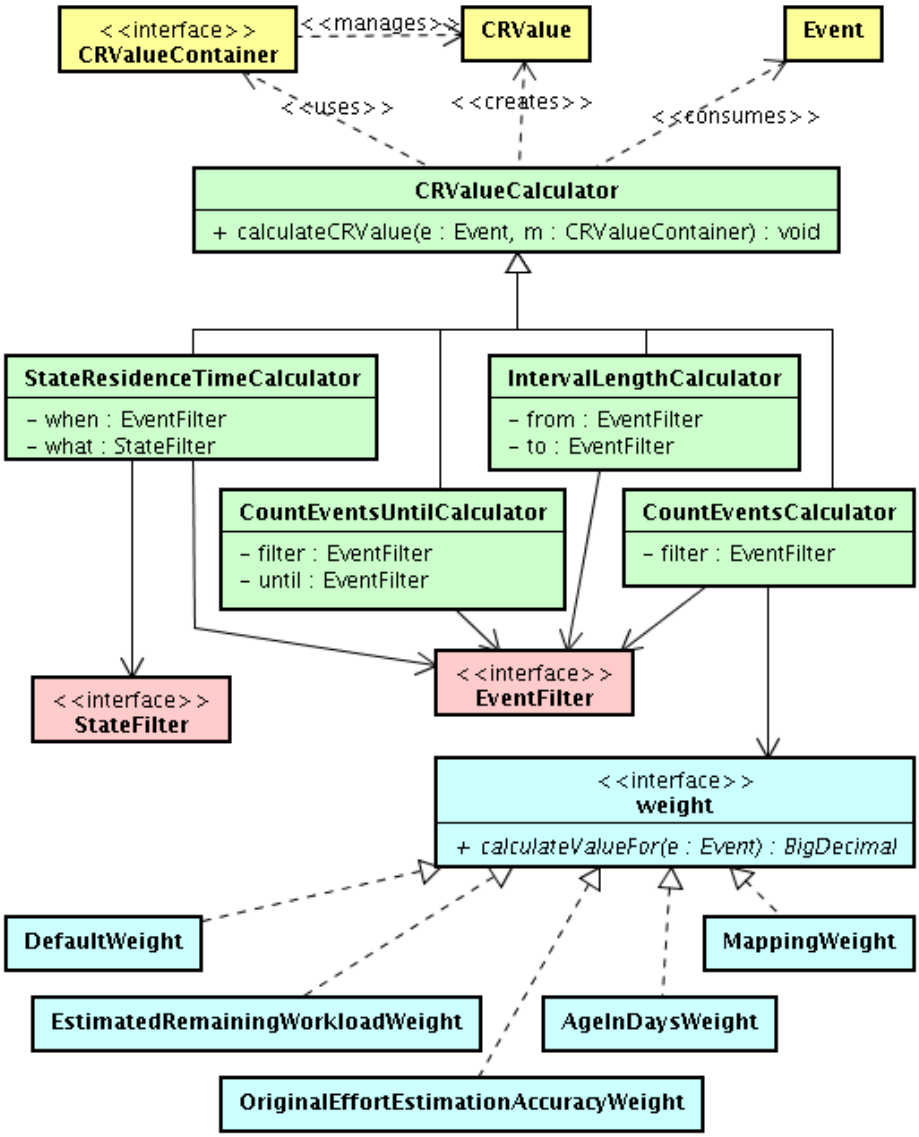

Figure 3: Design of the CR value calculators

4.3 CR Value Calculators

CR value calculators (see Figure 3) calculate CR values on certain events. They are a core part of the evaluation algorithm, filtering and transforming the event stream to a set of CR values. The following CR value calculators are predefined:

- **CountsEventsCalculator** is the most flexible calculator. It contains an event filter that selects the events for which CR values are calculated. The calculation of the numerical values of the CR values is delegated to the weight the calculator is parametrized with. Examples where this calculator is used are the incoming rate or the outgoing rate.
- **CountEventsUntilCalculator** calculates the number of times an event has occurred for a CR until another event happened. Both events are specified

by an event filter. An example where this calculator is used is the number of assignee changes before resolution metric.

- **IntervalLengthCalculator** calculates the length of the time interval in days between two events that happen on a CR. Both events are specified by an event filter. The interval length calculator can be used to calculate the age of a CR before it switches to the processing state for the first time, for example.
- **StateResidenceTimeCalculator** calculates the time in days a CR was in a certain state before the time point of a certain event. At such an event, a CR value is calculated. The event is specified by an event filter and the state is specified by a state filter. This calculator is used in the average processing time metric, for example.

Using event filters to configure the CR value calculators has two advantages:

- The different concerns of event filtering and calculating CR values are separated. This especially provides independent extensibility of the event filter variation point and the CR value calculator variation point.
- Flexible configuration of the CR value calculators with different event filters in the metric specifications.

Weights are used in the **CountEventsCalculator** to calculate a value for a CR state. This allows a flexible configuration with different weights and is in fact a separation of concerns between the decision whether to calculate a value or not and the calculation itself. Because it is likely that new calculations are added, the effort of introducing them is reduced by this design. Examples for available weights are the **DefaultWeight** that returns the value 1 for each CR, the **AgeInDaysWeight** that returns the age of a CR on the occurrence of the event in days, and the **EstimatedRemainingWorkloadWeight**.

4.4 Information Requirement Abstraction

The goal of the information requirement abstraction design is to achieve independence of the use of the CR fields from the way they are stored and retrieved. This is especially essential if the underlying CRM system changes. The different concerns of using CR fields and retrieving their values are separated by this design.

InformationConsumers (see Figure 4) are objects that require information about certain CR fields. The following classes are **InformationConsumers**:

- **CRValueCalculators**. They need information about the fields based on which they calculate values. Furthermore, they need the information required by their event filters.

- **StateFilters**. They require information about the fields their CR state filtering is based on.
- **EventFilters**. They may include state filters and require the information required by these state filters.
- **GroupParameters**. They require information about the field for which the grouping splits the CR values into different groups.

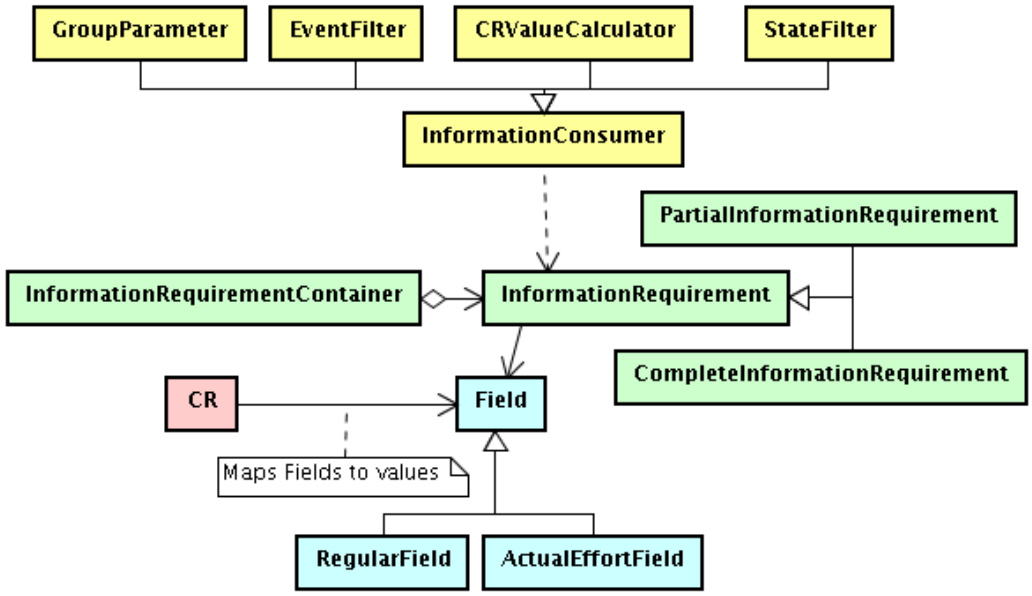

Figure 4: Design of the information requirements

InformationConsumers have **InformationRequirements**. There are two types of **InformationRequirements**: complete and partial information requirements.

- **PartialInformationRequirements** model the need for information about certain values of a given field. For example, for a state filter that filters on a certain product, not all information about product values in CRs is necessary, only information that affects that product.
- **CompleteInformationRequirements** model the need for information about a certain CR field. This is the case for group parameters, which do not know which groups there are in advance.

All **InformationRequirements** are directed on one specific CR field. For example, a group parameter that creates the groups according to the assignee has a **CompleteInformationRequirement** about the **assignee** field, which is a regular field stored in the Bugzilla bugs table.

For the different types of CR field representations in the database, there are different types of fields:

- Regular fields represent CR fields that are stored as columns in the Bugzilla bugs table.
- The actual effort field represents the actual effort value reconstructed from the CR history.
- In the original implementation of the tool for a customized Bugzilla, there were several other field types.

A CR object maps fields to the current values of these fields. It is independent from the field implementation and the data retrieval.

5 Metric Specification Example

To give an impression how the metric specifications which are evaluated by BugzillaMetrics look like, the backlog management index (BMI) [17] is used as example. It is calculated as follows:

$$\mathrm{BMI} = \frac{\mathrm{outgoingRate}}{\mathrm{incomingRate}}$$

The metric specification for the BMI is shown in Figure 5. It is divided into five different parts. We will focus on the basic filter, the group evaluation and the CR value calculator parts here. The other sections of the metric specification mostly deal with the evaluated time period and the grouping of the CR values.

The basic filter XML element contains the specification of a state filter. In the example, it is an `OR` filter which contains two filter specifications on the product field of the CRs. Therefore, all CRs which are or have been in the products `Product A' and 'Product B' are accepted by the basic filter. Only these CRs are selected from the database and further evaluated by the algorithm.

The CR value calculators XML element contains the specifications for the different CR value calculators that are evaluated by the algorithm. In the example, two CR value calculators that count events are defined, one for the incoming rate and one for the outgoing rate. Each CR value calculator has an identifier by which it can be referenced from the group evaluation specification.

The CR value calculator for the incoming rate counts the CR creations. Each CR creation is counted by the default weight 1.

The CR value calculator for the outgoing rate counts CR status transitions from the set of unfinished work CR states to the set of finished work CR states.

The group evaluation is a calculation that is named `BMI`. For each CR value calculator, the CR values are summed up. The sum of the outgoing rate CR value calculator is divided by the sum of the incoming rate CR value calculator.

```
<metric>
    <basicFilter>
        <or>
            <value field="product">Product A</value>
            <value field="product">Product B</value>
        </or>
    </basicFilter>
    <groupingParameters>
        <fieldGrouping>product</fieldGrouping>
    </groupingParameters>
    <groupEvaluations>
        <calculation name="BMI" >
        <divide>
            <sum crValueCalculator="outgoing" />
            <sum crValueCalculator="incoming" />
         </divide>
      </calculation>
    </groupEvaluations>
    <crValueCalculators>
        <countEvents id="incoming">
            <event>
                <create/>
            </event>
        </countEvents>
        <countEvents id="outgoing">
            <event>
                 <transition field="status">
                     <from>NEW</from>
                     <from>REOPENED</from>
                     <from>ASSIGNED</from>
                     <to>RESOLVED</to>
                     <to>CLOSED</to>
                 </transition>
            </event>
        </countEvents>
   </crValueCalculators>
   <evaluationTimePeriod>
       <start>2006-08-01</start>
       <end>2007-08-01</end>
   </evaluationTimePeriod>
</metric>
```

Figure 5: Backlog management index specification

6 Practical Results

BugzillaMetrics was used in practice on a company database containing about 20,000 CRs, aggregated over the past five years, with the most active product consisting of about 10,000 CRs.

Discussions of the metrics and charts created by the tool with the users gave a first impression of the advantages and problems of the tool and its usage.

The already known advantages of using metrics in general are, amongst others, that vague assumptions are supported by concrete figures, and that it can be controlled whether process changes lead to an improved process or not [7]. Besides these, another important advantage is the specification of new metrics. Metrics can be validated and adjusted without major programming, concentrating the main effort on the model of the metric, not its implementation.

Furthermore, because of its modular architecture and the separation of the data source, porting the tool from a modified Bugzilla to a standard Bugzilla was rather easy.

But it also showed that specifying metrics on CRM data bears some pitfalls in practice, in addition to the general pitfalls of metrics already observed in literature like manipulation of the data base [13] and the fact that interpretation is a must [14, 20]. Examples for problems that occur in practice are given in the following.

6.1 Comparing incoming and outgoing rate

There are some pitfalls comparing the incoming and outgoing rate, which then as consequence affect the BMI. If the incoming rate is defined as the number of CRs created in a time interval, and the outgoing rate is defined as the number of CRs resolved in a time interval, there are the following problems:

- CRs can be resolved multiple times if they are reopened.
- CRs that are created outside the scope of the basic filter and moved into its scope can be resolved and counted in the outgoing rate, but are not counted in the incoming rate. An example for this is the scenario when CRs are filtered for a given component, but some CRs were created in another component and then moved into the filtered component.
- CRs that are created inside the scope of the basic filter and moved outside its scope are counted in the incoming rate, but not in the outgoing rate.

This problem can be solved by changing the metric specification to take care of all these effects. For example, CRs moved outside the scope of the basic filter can be added to the outgoing rate.

6.2 Interpretation depends on CR selection

The defect rate can be defined as

$$\text{DefectRate} = \frac{\text{Bugs}}{\text{Bugs} + \text{Enhancements}}$$

Using it on different sets of CRs from the same product yields different results.

- The defect rate of all CRs, i.e. the open and closed CRs, shows how much defects there were in the complete product history compared to enhancements.
- The defect rate of the open CRs shows how much open defects there are in relation to open enhancements. This rate is often lower than the defect rate of all CRs, because defects are often smaller and fixed faster than enhancements. Furthermore, low-priority enhancements tend to accumulate over time.
- The defect rate of the CRs resolved in a time interval shows how much defects are fixed compared to completed enhancements in that time interval. This rate is usually higher than the defect rate of the open CRs for the same reasons as above.

This example shows that it is important in general to consider not only how a metric is calculated on a higher level, but which CRs are selected on which events, too. This is especially relevant when interpreting the metric results.

7 Summary and Future Work

In this paper, the main concepts of BugzillaMetrics, a tool for the flexible evaluation of metrics on CRM system databases, have been presented.

The starting point was the evaluation of existing CRM systems. Based on those results, a design approach for a metric evaluation tool has been outlined. The most important design concepts are the separation between the metric specification and the data retrieval, and the flexible configuration of metrics. The efficiency optimizations of the evaluation algorithm have been shown, as well as practical results from the usage of BugzillaMetrics.

Further work on the tool includes the development of a better user interface and algorithm improvements. The algorithm improvements are switching to a backward-forward calculation instead of a backward calculation to improve memory efficiency, and changes to the CR value classification to fix calculation problems with CRs that change their classification in their history.

BugzillaMetrics is published as an open-source project [2].

Acknowledgements

This work was supported by Kisters AG, Aachen.

References

1. Bugzilla 2.18.6. http://www.bugzilla.org.
2. BugzillaMetrics. http://www.bugzillametrics.org.
3. Code Beamer 4.2.1. http://www.intland.com.
4. JIRA 3.7. http://www.atlassian.com/software/jira/.
5. Kisters AG. http://www.kisters.de.
6. Polarion for Subversion 2.6. http://www.polarion.com.
7. V. Basili, G. Caldiera, and H. Rombach. The goal question metric approach. Encyclopedia of Software Engineering, pages 528–532, 1994.
8. G. Canfora and L. Cerulo. Impact analysis by mining software and change request repositories. In 11th IEEE International Symposium on Software Metrics (METRICS 2005), page 29. IEEE Computer Society, 2005.
9. G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In SAC '06: Proceedings of the 2006 ACM symposium on Applied computing, pages 1767–1772, New York, NY, USA, 2006. ACM Press.
10. P. C. Clements and L. Northrop. Software Product Lines: Practices and Patterns. Addison Wesley. 2001.
11. D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. IEEE Trans. Software Eng., 31(6):446–465, 2005.
12. M. D'Ambros, M. Lanza, and M. Pinzger. "A Bug's Life" - Visualizing a Bug Database. In Proceedings of VISSOFT 2007 (4th IEEE International Workshop on Visualizing Software For Understanding and Analysis), pages 113–120. IEEE CS Press, June 2007.
13. T. DeMarco. Why Does Software Cost So Much? Dorset House Publishing, New York, 1995.
14. N. E. Fenton and S. L. Pfleeger. Software Metrics. PWS Publishing Company, Boston, MA, 1996.
15. M. Fischer and H. Gall. Visualizing feature evolution of large-scale software based on problem and modification report data. Journal of Software Maintenance, 16(6):385–403, 2004.
16. L. Grammel. Development of a tool for the evaluation of change requests. Diploma thesis, RWTH Aachen University, 2007.
17. S. H. Kan. Metrics and Models in Software Quality Engineering. Addison-Wesley, Reading, MA, 1995.
18. M. Ohira, R. Yokomori, M. Sakai, K. Matsumoto, K. Inoue, and K. Torii. Empirical

project monitor: A tool for mining multiple project data. In Mining Software Repositories Workshop, 26th International Conference on Software Engineering (Edinburgh, Scotland), 2004.

19. G. Robles, J. Gonzalez-Barahona, and R. Ghosh. Gluetheos: Automating the retrieval and analysis of data from publicly available software repositories. In Mining Software Repositories Workshop, 26th International Conference on Software Engineering (Edinburgh, Scotland), 2004.

20. D. H. Rombach, L. C. Briand, and C. M. Differding. Practical guidelines for measurement-based process improvement. Software Process: Improvement and Practice, 2(4), 1997.