BugzillaMetrics - An adaptable tool for evaluating metric specifications on change requests

Lars Grammel, Holger Schackmann, Horst Lichter
RWTH Aachen University – Research Group Software Construction
Ahornstr. 55, 52074 Aachen, Germany
lars.grammel@googlemail.com, {schackmann, lichter}@cs.rwth-aachen.de

ABSTRACT

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 available in change request management (CRM) systems like Bugzilla.

Metrics and charts on change requests are already available in current CRM systems. They provide information about common metrics, but their adaptability is limited. This paper describes a more flexible approach for the evaluation of metrics on change requests.

The main characteristics of the tool presented in this paper are the separation between metric specification and data retrieval, an event driven algorithm that calculates time series data, and an abstraction of its data sources.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*process metrics, product metrics*; D.2.11 [**Software Engineering**]: Software Architectures—*domain-specific architectures, data abstraction*

Keywords

Evaluation of change requests, software measurement, process metrics, metrics specification, change request management

1. INTRODUCTION

Information on the evolution of software processes and products can be obtained by analyzing the data available in change request management **(CRM)** systems. The evaluation of metrics on this data can be used for several purposes, e.g. to improve the awareness and monitoring of current project states, identify software development process weaknesses, and assess the result of changes in the process.

But the report functionality of existing CRM tools typically contains only a fixed set of common metrics and is limited in its adaptability (see Section 2). 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 fulfill these requirements [10]. It was developed and used in cooperation with Kisters AG [5]. This company offers a large portfolio of software products developed as software product lines. Kisters AG uses a customized Bugzilla installation.

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

IWPSE'07, September 3-4, 2007, Dubrovnik, Croatia.

2. EXISTING TOOLS

Metrics and charts on Change Requests **(CRs)** are already available in current CRM tools. The development of BugzillaMetrics was based on the evaluation and comparison of the CRM tools **Bugzilla** [1], **JIRA** [4], **Polarion for Subversion** [6] and **Code Beamer** [3]. A brief overview of the relevant results is given in the following.

Reports usually have a basic filter that determines the set of CRs that are evaluated. The filter functionality provided by the different tools ranges from simple keyword search to the specification of the `WHERE` part of an SQL statement. Whereas a simple keyword search is not sufficient for selecting a specific set of CRs, SQL statements are on a level of abstraction that is too technical to be used by project or product managers. Intermediate level features are for example specifying a range or a set of values, filtering for field changes by specifying an original or new value, filtering by regular expressions, and combining filters with boolean operators.

Reports can be divided into snapshot reports that calculate values for a specific point of time, and time series reports that calculate values at certain intervals within a time period. The evaluated tools basically offer the following types of snapshot reports:

- 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, or the priority distribution of all CRs.

- 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 estimated remaining time.

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 that comparing metric results from different CRM systems is not easy due to slight differences in the metric implementations.

3. REQUIREMENTS AND ARCHITECTURE

In this section, we first introduce some basic terms. Then the requirements for BugzillaMetrics are described and an overview of the architecture is given.

3.1 Basic Terms

A **change request** is a request to extend or modify a software system. 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, e.g. 'new', or 'assigned'.

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. 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 CR state.

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 CR creation events are counted in a CR value with the default weight of 1.

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.

Several variations points of the evaluation algorithm were identified that describe the flexibility requirements for BugzillaMetrics. They are examined in the following, grouped by their likelihood to change.

- **Weights** are used in the calculation of CR values on certain events. New weights are likely to be added when new calculations 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.

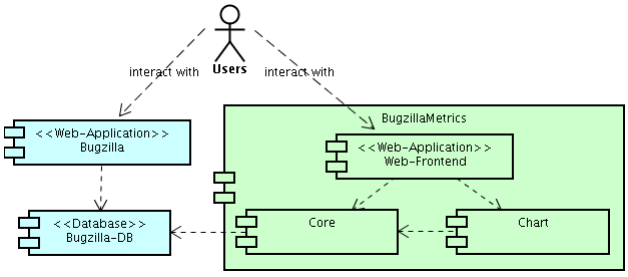

Figure 1: Integration of the architecture in the existing environment

The following aspects are somewhat likely to change:

- New **CR state filters**, for example to support range checks on number fields, might be added.
- **Events** and the corresponding **event filters** might be added to represent and filter new data sources.
- **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:

- **CR value calculators** that support calculations outside 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 its 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.

3.3 General Architecture

The tool architecture defines three components (see Figure 1). The **core component** contains the evaluation algorithm which calculates the metric result for a metric specification, and has read-only access to an existing Bugzilla database. Both the metric result and the metric specification are XML documents. The **chart component** creates chart images from a given metric result. The **web frontend** provides a user interface for specifying and using metrics.

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, and an abstraction of the data sources.

The algorithm can be divided into these main steps:

1. The XML metric specification is parsed and the object structure of the metric calculation is configured, e.g. the CR value calculators.

2. It is calculated which information is required for the metric evaluation. Then the objects for the CRs included in the basic filter are created and initialized with the current values for the required fields.
3. All CR values are calculated by processing the event sources and calling the configured CR value calculators (see Subsection 4.2) with the created events. The CR values are classified in a tree structure that is similar to the result structure. The event source processing goes back in time from the newest to the oldest relevant event. The CR states are updated accordingly, so each CR has the state it had when the event occurred.
4. 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.
5. The XML result document for the group values that are stored in the CR value container tree is created.

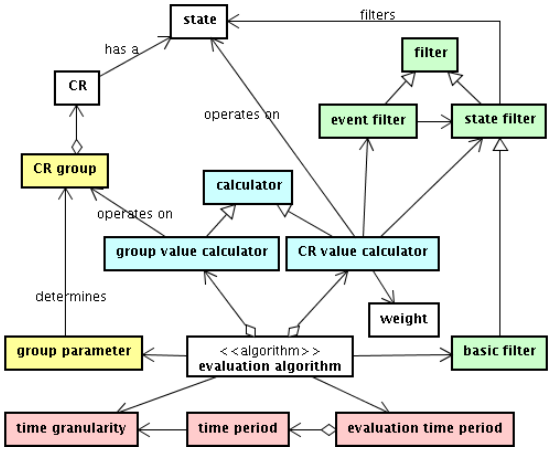

Figure 2: Parametrization of the algorithm

4.1 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. It 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.
- The **group parameter** determines how the evaluated CRs will be partitioned into CR groups before the group themselves are evaluated.
- **CR value calculators** determine 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.

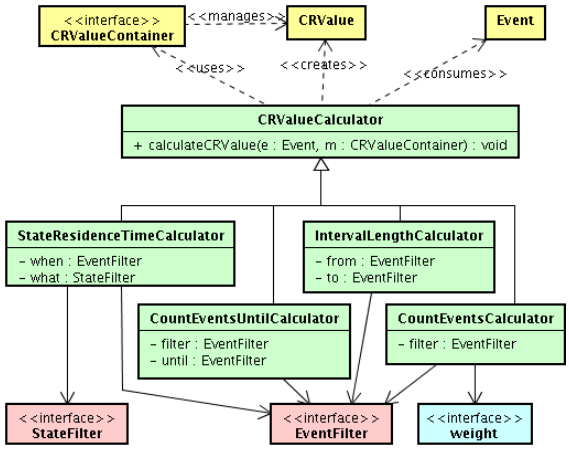

Figure 3: Design of the CR value calculators

- **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**.

4.2 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:

- `CountEventsCalculator` 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. An example for this calculator is the average processing time metric.

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.

```
<metric>
   <groupingParameters>
      <fieldGrouping>product</fieldGrouping>
   </groupingParameters>
   <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>
   <groupEvaluations>
      <calculation name="BMI" >
         <divide>
           <sum crValueCalculator="outgoing" />
           <sum crValueCalculator="incoming" />
         </divide>
      </calculation>
   </groupEvaluations>
   <evaluationTimePeriod>
       <start> 2006-08-14 </start>
       <end> 2006-08-27 </end>
   </evaluationTimePeriod>
</metric>
```

Figure 4: Backlog management index specification

5. METRIC SPECIFICATION EXAMPLE

Figure 5 shows as an example of a metric specification a definition of the backlog management index (BMI) [11].

It is calculated as: $\text{BMI} = \frac{\text{outgoingRate}}{\text{incomingRate}}$

No basic filter is given in the specification, so all CRs in the database are evaluated by the algorithm. The grouping parameters XML element determines that the CRs are split into groups according to their product. So the BMI for each product will be represented by a line in the resulting chart.

In the example, two CR value calculators that count events are defined, one for the incoming rate that counts the CR creations with the default weight 1, and one for the outgoing rate that counts CR status transitions from the set of unfinished work CR states to the set of finished work CR states. The group evaluation XML element then defines how the CR value calculators are combined to obtain the BMI as the final result.

6. PRACTICAL RESULTS

BugzillaMetrics was used in practice on a company database containing about 20,000 CRs, aggregated over 5 years.

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 the success of process changes can be controlled [7]. Besides these, an advantage of BugzillaMetrics is that the usage of metric specifications allows metric developers to concentrate on the model of the metric, not its implementation. In addition to the general pitfalls of metrics already observed in literature like manipulation of the data base [8] and the fact that interpretation is a must [9, 12], the usage of metric specifications also revealed some pitfalls of metric definitions caused by some subtle differences that would otherwise probably remain hidden in the 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.

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. Further work on the tool includes the development of a better user interface and algorithm improvements. BugzillaMetrics will be published as an open-source project soon [2].

8. REFERENCES

[1] Bugzilla 2.18.6. http://www.bugzilla.org.

[2] BugzillaMetrics. http://bugzillametrics.sf.net.

[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] T. DeMarco. *Why Does Software Cost So Much?* Dorset House Publishing, New York, 1995.

[9] N. E. Fenton and S. L. Pfleeger. *Software Metrics.* PWS Publishing Company, Boston, MA, 1996.

[10] L. Grammel. *Development of a tool for the evaluation of change requests.* Diploma thesis, RWTH Aachen University, 2007.

[11] S. H. Kan. *Metrics and Models in Software Quality Engineering.* Addison-Wesley, Reading, MA, 1995.

[12] 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.