

Master Thesis

Development of Query and Reporting Interface for a Change Request Analysis Tool

By
Malek Obaid

Submitted to

Faculty of Mathematics, Computer Sciences and Natural Sciences
RWTH Aachen University

December 2007

Research Group Software Construction
Prof. Dr. rer. nat. Horst Lichter

Reviewers:
Prof. Dr. rer. nat. Horst Lichter
Prof. Dr. rer. nat. Jan Borchers

Supervisor:
Dipl.-Inform. Holger Schackmann

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den 01-12-2007

Malek Obaid

Contents

1	Introduction	1
2	Background and Related Work	3
2.1	Analysis of Change Requests	3
2.1.1	Change Requests	3
2.1.2	Change Request Management	4
2.1.3	Software Metrics and Measurements	6
2.2	BugzillaMetrics Tool	8
2.2.1	BugzillaMetrics Core	9
2.2.2	Metrics Specifications Parameters	10
2.2.3	Example	11
2.2.4	Case value calculators	12
2.2.5	Simple Frontend	15
2.2.6	Efficiency achievements	15
3	User Interface Design and Metrics Tools	17
3.1	User Interface Design	17
3.1.1	Introduction	17
3.1.2	Interface Design Process	18

3.1.3	Design Guidelines	20
3.1.4	Design Patterns	21
3.1.5	Evaluation of Interfaces - Standards	24
3.2	Evaluation of Existing Metrics Interfaces	26
3.2.1	Bugzilla	27
3.2.2	JIRA	28
3.2.3	Polarion	30
3.2.4	Code Beamer	32
3.2.5	Summary	34
4	Requirements	37
4.1	Introduction	37
4.1.1	Goal and Intention of the System	37
4.1.2	Users of the System	38
4.1.3	Assumptions and Dependencies	38
4.1.4	System Environment	39
4.1.5	Definitions and Concepts	39
4.2	Functional Requirements	40
4.2.1	Basic Interface Scenario	41
4.2.2	First Level Interface	41
4.2.3	Second Level Interface	42
4.2.4	Event Filter	47
4.2.5	Third Level Interface	48
4.2.6	Saved Queries	48
4.2.7	Evaluation and Charts	50

4.2.8	State Management	52
4.2.9	Authentication Management	52
4.2.10	Database Requirements	53
4.3	Non-Functional Requirements	53
4.3.1	Requirements of User Interface	53
4.3.2	Performance Requirements	53
4.3.3	Maintenance Requirements	53
4.3.4	Security Requirements	54
5	Architecture	55
5.1	Google Web Toolkit (GWT)	55
5.1.1	General GWT Architecture	56
5.1.2	GWT Features	57
5.1.3	Project Structure and Modules	60
5.1.4	Challenges and Current State	61
5.2	System Architecture	62
5.2.1	General Architecture and Integration	62
5.2.2	Client and Server Abstraction	63
5.2.3	Second Level Interface Widgets	66
5.2.4	XML Selections Builders	67
5.2.5	Variation points of other categories	69
5.2.6	XML Parsers	70
5.2.7	Evaluation and Charts	71
5.2.8	System Authentication	73
5.2.9	Database Access	75

6	Evaluation	79
6.1	System Overview	79
6.2	Evaluation Approach	81
6.3	Software Product Evaluation	81
6.3.1	Functionality	82
6.3.2	Reliability	82
6.3.3	Usability	83
6.3.4	Efficiency	84
6.3.5	Maintainability	84
6.3.6	Portability	85
7	Summary and Future Extensions	87
A	Interface Screenshots	89

Chapter 1

Introduction

Management of the development and maintenance of large software systems is a major part for the improvement of their quality. One of the management aspects is measurements and metrics that define specific observations of characteristics of a software process or product.

Support of change requests management is essential in the software products monitoring, control, and planning. Tools of change requests management do not only facilitate the systematic collection, processing, and administration of bug reports and new change requirements. The administered data by the tool contains important information to uncover needed improvements in the development process, as well as evaluation of the implemented improvements.

However, Evaluations of change requests provided by existing tools are rather limited. Some tools provide only fixed variant measurements of software metrics, or others have limitations on types of results that can be defined.

Kisters AG is an industrial cooperation partner of the research group software construction. They use the widespread open source tool Bugzilla for the administration of change requests. Several extensions were added to the tool, like additional change requests states and an improved dependencies tracking between development projects. A flexible tool BugzillaMetrics was developed in cooperation with Kisters AG to wide the variety of possible metrics calculations, which uses XML configuration files to define metrics for charts generation [Gra07].

The purpose of this thesis is to complement BugzillaMetrics in a graphical user interface, which allows users specifications to define metrics rather than only XML. The Interface should conform to high usability standards and provide understandable query and reporting component for defining complex metrics. Concepts like predefined metrics and detailed classifications of types of metrics should be con-

sidered, in addition to saving and manipulating defined queries.

The thesis task includes literature study of change requests management, evaluation of related tools, study of user interface design principles, and finally design and implementation of the required system based on requirements. The details of requirements are formulated based on further and continuous discussion meetings with the potential users at Kisters AG.

Structure of the Thesis

Background and related work are discussed in chapter two, which includes change request management related basic terms, and details of the tool BugzillaMetrics. The next chapter contains literature of user interface design principles, and an evaluation of existing tools in terms of features and design. These two chapters provide motivation and basis for the interface development process. Chapter four discusses the functional and nonfunctional requirements of the needed interface system. The architecture of the developed system components is detailed in chapter five, as well as some information about the used technology for implementation. Chapter six includes an overview and the evaluation of the developed interface system. Finally, the last chapter provides a summary of the thesis and possible future extensions. Screenshots from the interface are also provided in the summary.

Chapter 2

Background and Related Work

This chapter discusses basic terms related to change requests and their management, and goes into details of the tool upon which, the thesis interface is developed. The first section talks about analysis of change requests, metrics that can be measured in change request analysis tools, and their benefits. The second section discusses the change request analysis tool BugzillaMetrics since the developed reporting interface is developed for it.

2.1 Analysis of Change Requests

This section is devoted to discuss analysis of change requests, which gives an understanding of what the requests are and their management concepts. Additionally, it discusses software metrics that describe characteristics of the software development process and product.

2.1.1 Change Requests

Change Request (CR) is a request of change in a software system, which can be a detected problem in the system, an enhancement call to improve an aspect or task of the system, a new needed requirement, or a change in the requirements. A request can be a description for a bug, enhancement request, or support call. It may be referred by different names based on their management tools, such as an issue, a bug, or a case.

The change requests are formed as reports by stakeholders like the customers, quality assurance department, or managers. Each request report includes many helpful

properties for later management, such properties as the following:

- Description of the problem is essential property as it explains what the request is.
- The used version of the software on which the problem is detected, in addition to some other optional request environment properties like product, component, or target milestone.
- Priority of the request to specify how essential this request is. While priority indicates how important or critical a request is for the organization internally, another property 'severity' indicates the importance of the request for customers.
- The user to which this request is directed or assigned, which is helpful for managing requests based on their assigned users, and for users themselves.
- The request state is important to show what the current status of the request is. It can begin for example to be assigned by the manager to specific developer, and verified or closed shows that the request is completed.
- Sometimes a deadline can also be specified if required.

Reporting these change requests is not only beneficial for the management of software development by project managers to keep track of the work in an administered manner, but also makes it easy for customers (or stakeholders) to identify needs or requirements changes to raise such requirements in an effectively captured way and their impact on the system can then be evaluated.

2.1.2 Change Request Management

The process of changing the software after delivery is called software maintenance. Change request management (see figure 2.1) is a software maintenance activity for tracking and monitoring such changes in order to maintain control and administration of the system evolution.

By raising all change requests in a central repository, they are then subject to filter, keep track, or analyze their costs and benefits based on components they belong to.

There are electronic systems that are usually used to manage and administer change requests, which are called Change Requests Management Tools, which are sometimes referred to as bug tracking systems or issue tracking systems. These tools provide reporting and graphical observations like charts to help managing change

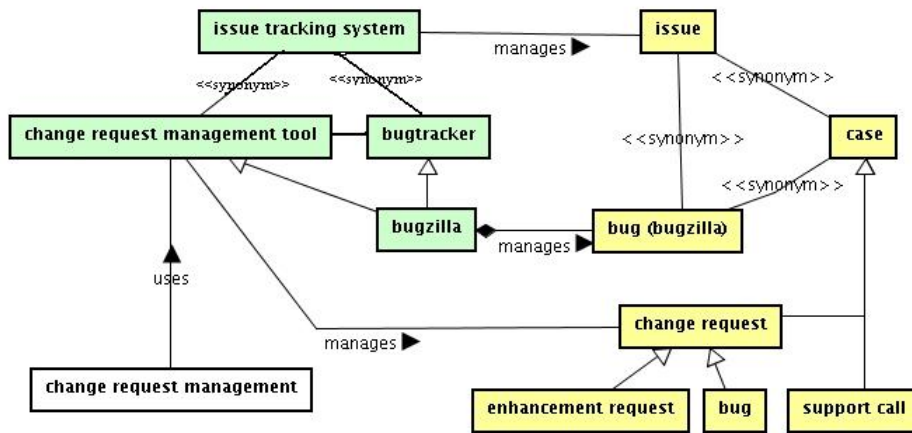


Figure 2.1: Change Requests Management [Gra07]

requests. They are often used by companies using their own solutions or some already developed commercial or non-commercial systems for this purpose.

Main benefits of using change request management tools are detection of requests with their priority levels that also allows effective changes detections and then faster and organized development, providing a solid risk management observation of the company products, and metrics specifications and results that give managers fast measurement and evaluation of the work efficiency.

One common system is Bugzilla [BUGa], which is a comprehensive tool for change request analysis. Users can add, search, and visualize change requests. In addition to produce charting or reporting overviews of specific measurements they may seek. Bugzilla allows individual customizations and is published as open source project.

Change Request Workflow

Workflow is an important property of the change requests management tool, which describes current request state or status. Workflow states can be up to organizations customizations as needed. Therefore, they may differ from an implementation to another. Current Bugzilla tool basic workflow [BUGc] is taken in order to give an idea about requests workflows and transformations in between.

Besides the fact that also these states can be customized as Bugzilla provides, the following is an illustration of current Bugzilla CRs workflow (see also figure 2.2).

- **New** bug state can be the initial state of a Bugzilla CR, or the state after moving from **Unconfirmed** in case of confirmation or enough votes.
- **Assigned** state comes either after **New**, **Unconfirmed**, or **Reopened** requests are assigned to a user or group.
- **Resolved** state comes after fixing a request, which could have been in any of **New**, **Unconfirmed**, or **Assigned** states.
- After verification of a solution, **Verified** is a possible next state. Then, it can be **Closed**.
- **Reopened** state indicates that a request is now a subject of being **Assigned** or **Resolved** again. A request that was in **Resolved**, **Verified**, or **Closed** can move to this state.

2.1.3 Software Metrics and Measurements

Software measurement is the process of assigning numbers or symbols to attributes of software related entities like software products, development process, code, or design specifications [Som04]. Measuring change requests and software problem reports provides a feasible and organized method for characterizing, evaluating, predicting, and improving software system within and after development.

Software metrics are software measurements, which can be direct, where simple metrics are measured like number of code lines, and defects. Indirect to calculate derived measurements from direct ones like defect density that is number of defects divided by total product size. Finally, prediction measurements that predict for example the required effort. For more metrics classifications, see [FP98].

In general, these metrics have different categories; each is related to different aspects of software measurements. The following are some important categories:

- **Data quality**
Metrics of this category are related to measurements of how accurate, complete, and correct the data are. For example, calculating the percentage of resolved change request without actual effort values among the set of resolved requests.
- **Product quality**
Metrics for measurements related to the current state of the product are in this category, like defect rate to calculate the percentage of defects or bugs

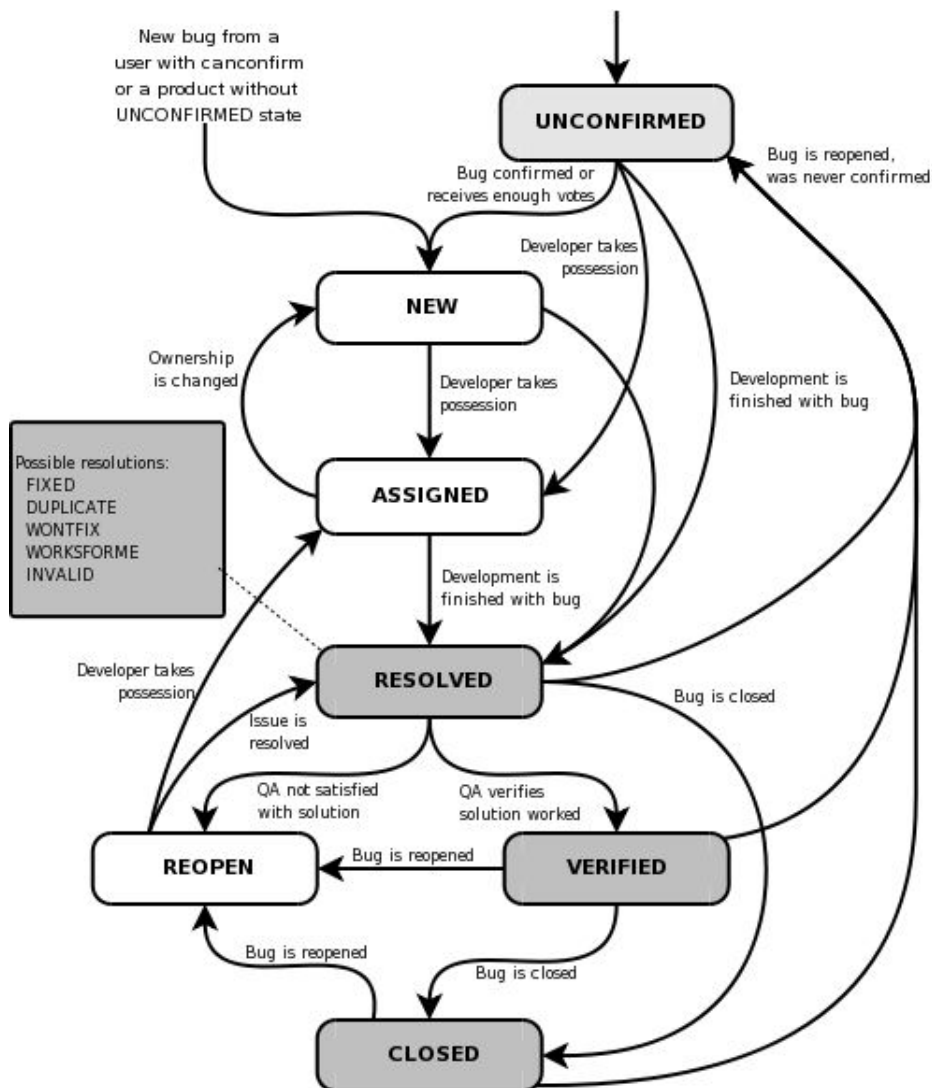


Figure 2.2: The Bugzilla life cycle model [BUGc]

reported for a product at a point of time, or original effort estimation accuracy to calculate how accurate the initial estimation of a change request comparing to the effort taken to close it.

- Workload

Metrics to measure how much work was done and how much work is still needed are in this category, like estimated remaining workload, and user activity distribution that calculate how many change requests are assigned for each user.

- Processing performance

Metrics of this category are related to measure how fast processing of change requests is. Visualizing such measurements can be helpful to motivate or change focus of work when e.g. slow components are detected with forgotten or delayed requests. Examples of this category metrics are totals overdue to calculate the percentage of beyond deadline requests, number of requests states changes before resolution, or state residence time that calculate how long a change request stayed in a specific state like assigned.

The previous categories are not totally comprehensive, many other metrics and categories can be measured in change request management tools like cost, or schedule related metrics. The diversity of software metrics shows how beneficial change request management tools are, and therefore, how important for industry organizations having a suitable tool that satisfies their needs.

As a result, the idea of building a flexible tool was initiated, which is discussed in next section. And this thesis came as a complement to enhance its usability by providing a flexible adaptable GUI that allows to specify the variant software metrics provided by BugzillaMetrics.

2.2 BugzillaMetrics Tool

Change requests management (CRM) has been analyzed and implemented by many tools, like Bugzilla, JIRA, Polarion, and Code Beamer. Some of them will be evaluated in the next chapter. In this section, BugzillaMetrics tool will be explained, which overcomes some of the shortcomings of the last tools, especially in the metric specifications flexibility aspect.

BugzillaMetrics [Gra07] is a change requests analysis tool that exceeds the limitations of existing CRM tool by the possibility to specify more variants of metrics using XML, and therefore, more flexible evaluations can be defined as queries of XML to get their charts, such different charts help to show the evolutions of software processes and products, monitor and administer the change requests (CRs), and depict the awareness of the development weaknesses or improvements. In the rest of this section, new terms and related concepts are introduced, then explaining the parameters of the XML metric specifications and the algorithm of parsing and processing the metric, and finally, an XML example and the different potential metrics and evaluations are mentioned.

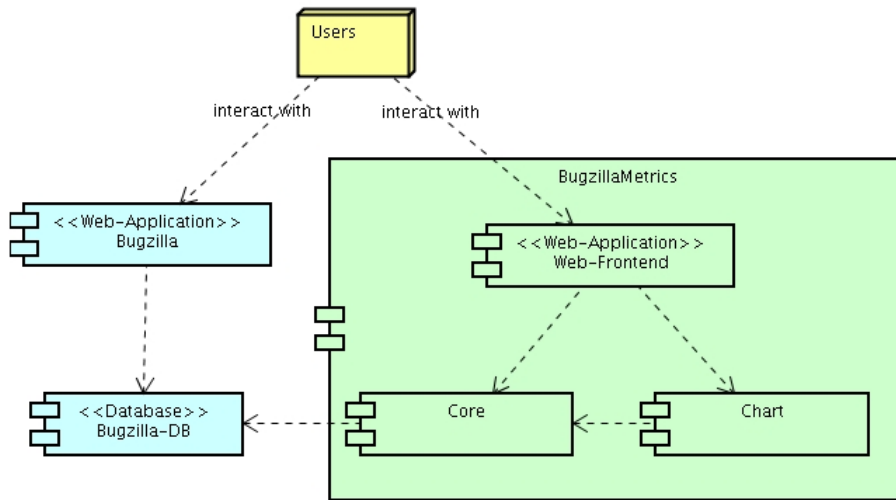


Figure 2.3: Architecture of BugzillaMetrics and environment [Gra07]

2.2.1 BugzillaMetrics Core

The tool supports a separation between the metrics evaluation and the data sources access, which increases its adaptability and usability. It uses the Bugzilla database as the data sources, and JFreeChart library to generate the charts based on the data retrieved from the parsed metric specification XML and chart configurations, the parsing and data retrieval from XML is processed in the core of the metric. Figure 2.3 gives an idea about the major parts of the tool and its integration in the existing environment. The tool includes a simple frontend to accept the XML specifications and in the backend, the core of the parsing and evaluation is processed and the chart component generate the configured chart, cases data retrieval of the DB is performed in core façades to access the Bugzilla DB tables.

Before proceeding into the details of how the tool identify metrics xml, some general terms and extended software process related changes are to be introduced:

- Case

It is a tracked issue, which is called bug in bugzilla and change request in general. A case can be a bug (to describe a defect), an enhancement request, or a support call. Filtering cases is a high level specification that contributes into the needed metric calculation, based on defining the values of some of the case properties. Examples of these properties are the product (specifying the products to which the needed cases belong), component, version, assignee (the cases that are assigned to specific assignees), the priority and severity of these cases, deadline date, actual effort (the number of hours

spent working on the case), and estimation accuracy, which is an estimation of how many work hours are needed to resolve the case.

- Case State

It is the state of work a case is currently in. A case can be in different states at a point of time like new, assigned, reopened, waiting customer information, processing, resolved, verified, shipped, and closed. These states depend on the organization using the tool, since as mentioned before; the tool supports the underlying independent data source (the database) that is used.

- Event

An event is the occurrence of a change on the case. For example, the creation of the case, a change in the case properties e.g. it hits its deadline, a case state transition, or end of a time interval. Specifying the event is a core part of the metric specification, since it is used in the case value calculators (later in this section).

- Weight

It is used in the calculation of case values on a specific event, mainly used to get more flexible results of the metric calculation. The default is the normal number of cases calculation, but it can be also the age in days like calculating the age of cases before resolution case state, estimated remaining workload, original effort estimation accuracy, or a manual field mapping by giving correspondent number for a set of fields values.

- Chart Configuration

A chart configuration is defined separately from the metric specifications, in which, user can define the properties of the desired resulting chart of the evaluation. These properties are like the width and height, title, type (like lined or stacked), and references to which case value calculator to be considered since more than one calculators can be specified in the metric specification and one or more calculation can be presented in the chart or in separate charts.

2.2.2 Metrics Specifications Parameters

The algorithm of metric evaluation can be formed by the following parameters. These parameters form the complete metric specification (see Figure 2.4):

- Basic filter

The basic filter is the high level filter to specify which cases are considered in the evaluation, by determining the values of the needed cases properties;

for example, allowing specific product cases, the cases belong to specific assignee, components, or cases with specific priority, severity.

- Grouping parameter

The grouping parameter specifies how the cases will be grouped in the evaluation, like grouping by product will evaluate the cases with each product cases as group, other groupings can be component, assignee, or version.

- Case value calculators

This is the core of the metric specification that will determine the requirement of the metric. It specifies how the cases will be calculated on certain events. There are four types of case value calculations, like count events calculation, count events until another event occurs calculation, interval length between the occurrence of two events, and state residence time calculator. These calculations types will be mentioned in details in this section later. Each event is determined from an event filter or more combined by an ‘And’ or ‘Or’, which includes few options as creation of a case, end of time interval, transition of any of the case states, and finally the state filter that is actually a basic filter, i.e. includes all the case properties values options but for the specific calculation; therefore, each calculation can additionally has its own cases filtering. More than one calculator of the same type can be in one metric specification, this give the flexibility to determine many calculations of the metric and combine them together in the group evaluation calculation if needed.

- Group calculation

In this part, how the results of the specified case value calculators are calculated and/or combined is determined. More than one group calculation can be added and mathematical operation over the calculations can be also specified.

- Evaluation time period

The time period of the evaluation is determined here.

- Time granularity

This parameter determines how the time period of the evaluation is split into time slots; for example, it can be week, day, or a month.

2.2.3 Example

After you have now an idea about the parameters of a metric specification, and the nature of each parameter, the example in Figure 2.5 gives an XML specification

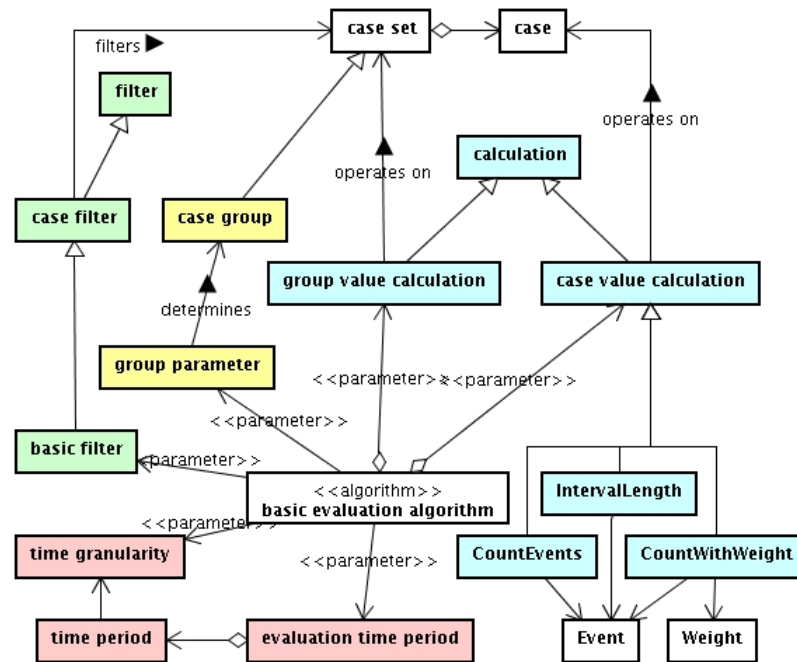


Figure 2.4: Metrics Specification Parameters [Gra07]

illustration to specify the Backlog Management Index (BMI), which is calculated in this example as the division of outgoing cases rate over the incoming rate. As you can see, there are two case value calculators, the first for the incoming rate that is specified by an event to filter the cases that have been created in that time period, and the second calculator is for the outgoing rate that is specified by an event to filter the cases, on which a state transition from new and assigned states into resolved or closed.

In the group evaluation, the two case value calculators are retrieved and a division is performed to get the BMI based on its formula. Additionally, you can see at the beginning the base filter, and at the end, the time evaluation period and time granularity splitting the period into weeks slots.

2.2.4 Case value calculators

Since these calculators are the major part of specifying the metric, different forms or types of these calculators are supported in order to provide the ability of defining the wide range of metrics this tool supports. In the following, the four supported categories are described with examples of different metrics that can be defined in

```

<baseFilter>
  <or>
    <value field="product">4</value>
    <value field="product">5</value>
  </or>
</baseFilter>
<groupingParameters>
  <fieldGrouping>product</fieldGrouping>
</groupingParameters>
<groupEvaluations>
  <calculation name="BMI" >
    <divide>
      <sum caseValueCalculator="outgoing" />
      <sum caseValueCalculator="incoming" />
    </divide>
  </calculation>
</groupEvaluations>
<caseValueCalculators>
  <countEvents id="incoming">
    <event>
      <or>
        <create />
      </or>
    </event>
    <weight>
      <default />
    </weight>
  </countEvents>
  <countEvents id="outgoing">
    <event>
      <or>
        <transition field="status">
          <from>NEW</from>
          <from>REOPENED</from>
          <from>ASSIGNED</from>
          <from>WAITCUSTINFO</from>
          <from>PROCESSING</from>
          <to>RESOLVED</to>
          <to>VERIFIED</to>
          <to>SHIPPED</to>
          <to>CLOSED</to>
        </transition>
      </or>
    </event>
    <weight>
      <default />
    </weight>
  </countEvents>
</caseValueCalculators>
<evaluationTimePeriod>
  <timePeriod>
    <start>2004-07-01</start>
    <end>2007-03-08</end>
  </timePeriod>
</evaluationTimePeriod>
<timePeriodGranularity>
  <week />
</timePeriodGranularity>

```

Figure 2.5: XML Specifications Example

each.

- **CountEventsCalculator**

This is the most flexible calculator, the evaluation in this category is based on counting specified events in the event filter, with the weight parameter selection to determine the calculation of the numerical values of the cases values. Metrics that can be defined in this category are like incoming rate which is the number of cases added in a time period, outgoing rate to calculate the resolved cases, the defect rate which is the percentage of the cases of type bug in the set of all cases, totals overdue to calculate the cases that are beyond their deadlines, estimated remaining workload, or original effort estimation accuracy to measure how precise the initial effort estimation is, compared to the actual effort needed to resolve the evaluated cases.

- **CountEventsUntilCalculator**

The case value calculator of the evaluation is determined based on counting the occurrence events specified in a first event filter until another event occurs, i.e. each evaluation of the metric is specified by two event filters, one is to filter the counted event until the second occurs. Metrics that can be defined in this category are like the number of assignee changes before processing, or the number of state changes before resolution to calculate how often a case state is changed before their resolution.

- **IntervalLengthCalculator**

This calculator is used to evaluate the length of time in days between the occurrences of two events. For example, calculation of age before processing, to calculate the time between the creation of a case and its processing time point.

- **StateResidenceTimeCalculator**

This calculator is used to calculate the whole length of time in days a case stayed in a certain state before an event occurs. An example metric is the assigned state residence time that calculates the period of time the evaluated cases stayed in the assigned case state.

Although the last two calculators calculate the interval length, they have a conceptual calculation difference. For example, if a metric in the IntervalLengthCalculator is specified to calculate the length between a specific case state start and end, it will not lead into the same result as calculating that case state residence time in the StateResidenceCalculator, since the second considers every period of time, a case was in that state, and it normal that a case can be in the same state more than once in its history.

2.2.5 Simple Frontend

The tool supports a simple frontend to provide the metric XML specifications and the chart configuration, and after validating and processing of the XML, a chart is generated for the result of the evaluation, or the result can be provided as XML also. The tool gives less importance of the interface, as it focuses more on the idea formation of evaluation based on metric specifications, and provides the core algorithm concepts and implementation of this evaluation mechanism. The task of providing a good interface for the tool is the concern of this thesis.

2.2.6 Efficiency achievements

In the algorithm of metrics evaluation, the optimization of execution time had a major attention, starting from considering only the cases of the base filter, and case value calculators are performed only once and at the same run over the history of the cases. Additionally, a major attention was considered in the performed SQL statements like using IN instead of OR statement, minimizing the use of some costly statements like the JOIN, and adding time restriction to the bugs activities retrieved results based on the evaluation time period of the metric, such restrictions considerations affect the overall time execution of the evaluation significantly. The execution of most metrics took less than twenty seconds. Another important achievement is the memory use optimization considered in the tool. Some affecting considerations were used like using hash maps that consumes less memory than using the java.util package implementation, defining new Numbers class for the often used zero and one values in weight and calculations instead of BigDecimal objects, and using object stream instead of string for the XML output. With such considerations and others, the execution of the evaluation produces guaranteed no out of memory problems and therefore, an efficient system in terms of memory.

Chapter 3

User Interface Design and Metrics Tools

In this chapter, the basics of user interface design and general design issues are discussed. Aspects like design process, guidelines and rules to enhance the design, and design patterns that provide practical design guidelines are covered. Additionally, evaluation of interfaces of some existing CRs Analysis tools is discussed in the second section.

3.1 User Interface Design

This section covers basics and aspects of interactive design issues. Benefits, process, rules, and patterns are discussed in detail.

3.1.1 Introduction

User Interface design is the design of devices, software applications, and web pages with the focus on user interaction. This however, should not reduce or ignore any of the features and functionalities the system is supposed to do.

Designing interfaces is not easy, users demands well behaved, good looking, and easy to use systems. Therefore, designers must always keep in mind guidelines and rules when designing a system. But the question is why and what are the benefits of following such guidelines, the following are some answers [AC]:

- Users will learn and adapt faster if the interface behaves in expected and easy

interactive way.

- Users can accomplish their tasks quickly, because if interface is designed well, it will not get into their way.
- Users with different experiences will still find the system accessible and achieve what they need.
- Interface system will appear in good look, which may attract users and encourage them to use it.
- Interface systems will be easy to document, because systems with standard behavior requires less documentation.
- Customer support calls will be reduced.

Designing interface principles do not only care about the look, but also they should lead to a good system in many other aspects. Characteristics of good systems include having high performance, ease of use, reliability, adaptability, interoperability, mobility, in addition to attractive appearance (from Apple Interfaces Guidelines [AC]). Hence, designing principles may go sometimes into the underlying architecture rather than only the interface.

The following subsections discuss the interface design process, guidelines when designing interface, and some of design patterns that can be used to handle common design problems.

3.1.2 Interface Design Process

The process towards designing user interface system does not start immediately from the graphical look. Designers described designing process in variant descriptions or activities. All of them share the core process or at least the meanings of process steps even if the descriptions may differ.

Considering many studies of the process, and mainly according to [PRS02]. The following are the basic activities that contribute in the design process:

- Identifying needs and establishing requirements

Designers need to know first what desired system is about, understanding the users, their needs, and hopes. This helps to know the future work to be done and what designers should try to achieve in the design activity.

This activity starts with gathering data, analyzing and interpreting them to capture requirements. It can be an iterative process itself, which is helpful

sometimes because designers may discover influences or other needs need to be discussed further or confirmed with stakeholders.

- Information architecture and alternative designs

This is the process of mapping and creating ideas to meet the requirements, and describe these ideas as different imagined scenarios. Such scenarios will be more understandable by users to describe what the system will do, behave, and look. Taking care of different possible alternatives of design, two models can be created at this step. Conceptual models to depict the scenarios and physical models that will consider details like colors, menus and icons design.

- Building interactive design versions (Prototyping)

This process includes making design decision and creating a design version of the system. It does not necessary mean completing a software version, a prototype is the main goal, which can be achieved by means that range from paper prototypes, screen shots, to software products. A software that includes some bugs or a simulation would be sufficient to get to users the feel of the interactions and behavior of the system. Within this process, also iteration is helpful and valid since users can feel here the product and find it clearer to know if this is what they want or what else they wish for.

- Evaluating designs and Implementing

By evaluation, designers know how good the design or product is and determine its usability and acceptability. Evaluation needs a high level of user involvement, as designers find out if they matched the requirements, how users use it, and what problems or errors users face or make using the design.

After decision on the prototype of the design version is made and enhancements are performed based on evaluation, designers can start implementation of the real life system knowing that they have a good understanding of different aspects of the system (functionality, behave, look).

Finally, within the overall process and activities, there are three main characteristics of the design process [PRS02], which form a key part to its success.

- Focus on users to whom the design is developed. All stakeholders should be taken into consideration, and feedback is very beneficial to improve design of a satisfying interface.
- Usability and user experiences goals, which will help designers to choose from different possible prototypes and designs, when these goals are clearly documented and agreed upon from the beginning.

- Iteration is a good manner to refine the design based on feedback from already designed parts, and reduce time instead of designing a complete system that may not be as customers imagined. Engagement between users and designers on requirements discussions in different stages give a great result in systems interface design.

3.1.3 Design Guidelines

Interface design depends on experience. There is no magical step-by-step exact methodology to end up with a perfect software design, but there are guidelines and principles in order to achieve a good interface. These principles are statements of policy that designers are advised to follow during the design process.

It is natural that many authors have different guidelines, which are described based on their experiences. We discuss the Shneiderman [Shn97] eight golden rules of interface design, which are as follows:

- **Consistency:** Designers should implement systems that preserve consistency, including consistent sequences of actions in different places of the system, consistent terminology and concepts within the system and with real world, and consistency in look like layout, colors, and fonts.
- **Enable frequent users to use shortcuts:** Designers should provide short ways to perform some actions, like hidden commands, abbreviations, and special keys. It is beneficial to have such shortcuts for both first time users of the system that give them fast impression when fast results can be achieved, and for expert users who use it often as it gets boring and time consuming when they need to do some action over and over.
- **Offer informative feedback:** Interface systems should respond in some-way for users' actions. Such response or feedback keeps users aware of what they are doing or what is happening.
- **Design dialogs to yield closure:** Interface systems should include sequences of actions that go from beginning to an end. Users should know that they finished their action at the end, which give them satisfaction of accomplishment, self confirmation of process end, and they can forward to next actions they want to perform.
- **Offer error prevention and simple error handling:** Designers should not allow serious errors to be caused by users on their systems, by limiting what they can do and where, e.g. detect letters in fields where numbers should be

entered. Besides, when an error occurs, it should be written with simple clarification. Simple and comprehensive mechanisms for error handling should be provided.

- **Permit easy reversal of actions:** Designed systems should always be able to offer to reverse actions, in case of errors or any other user's reason. When users know what they can undo actions or errors, it encourages them to explore and try any unfamiliar action in the system.
- **Support internal locus of control:** Experienced users usually desire to have the sense that they are in charge of the system and its actions. They like to be initiators of actions rather than responders. When users face surprising actions, inability to produce desired actions, and difficulty in obtaining information always lead to their dissatisfaction.
- **Reduce short-term memory load:** Interface systems should not include sequences of actions that require a lot of human memorizing. Users' memory load can be reduced by designing screens as simple with visible actions and options.

As mentioned before, these were rules of one chosen description aimed to give an idea about rules and guidelines designers may get benefit from. Some other authors went further to cover more details of design principles and guidelines. For example, Mayhew [May91] who gave a very detailed and big number of guidelines we will not go into their details, instead they are mentioned shortly. Additional great resource for interfaces design guidelines is the Apple Human Interface Guidelines [AC].

The general principles of UI design by Mayhew include taking care of user compatibility, product compatibility, consistency, familiarity of users with the system's concepts and terminology, simplicity, control, flexibility, WYSIWYG (what you see is what you get), responsiveness, invisible technology, protection, and robustness.

3.1.4 Design Patterns

While design pattern [GHJV95] is a general solution to common occurring problem in software design, there is interaction design pattern, which is a design pattern but provide solutions for common usability and accessibility problems in interface design.

These solutions are descriptions or templates, which do not give a finished design ready for implementation, but practical advises designers can put immediately to

use. Each of them has its time and situation to be recommended to use, i.e. not all are applicable anytime.

To reveal any misunderstanding with the designing guidelines concept, it is important to mention some advantages and difference between them. Guidelines do not give suggestions how to solve problems like patterns, sometimes they are guidelines for specific interfaces like specific operating system, and finally, there are too many of them, and designers may conflict trying to use all of them or know when to use the right ones. design patters can be beneficial to avoid design errors, for training, or to introduce design solutions to new designers.

The next are some selected patterns that were helpful in our designed system query interfaces, and Table 3.1 show brief descriptions of general overall interface design categories and their patterns according to Tidwell [Tid05].

1. Extras on Demand

This pattern is important in the organization of the content in interfaces, where their design keeps simple UI with simple and important data shown, and additional options are hidden. Hidden information or options are displayed after single gesture. A familiar example is the color picker dialog in windows, where basic colors are shown and additional hidden extension includes variety of all possible colors, when define custom colors button is clicked, hidden colors extension is shown.

This patterns has been used in many occasions in the developed metrics evaluation system, one example is the date picker where users have to enter dates to specify evaluation time period, when mouse curser or focus is directed on date text box, a calendar widget is shown to pick dates from. The example is shown in figure 3.1. Other examples in metrics query interface are adding second evaluations or event filter on demand when needed.

2. Clear entry points

This pattern is beneficial in task oriented applications, where designs should provide clear few entry point into interface, and each directs to a specific task or area. A familiar example is the Google homepage, in which users can easily notice entries options that lead into different other task oriented pages, like google images, maps, mail, and many more.

The metrics evaluation interface supports this pattern since it provides the whole application as six categories, based on tasks needed to be performed. Common metrics category, my queries, and detailed categories are displayed as entry doors to each of their related functionality. Users will not be confused or lost searching for a page in order to do a task.

3. Visual framework

Evaluation Period: always till today

Time Granularity:

(a) Simple date input

Evaluation Period: always till today

Time Granularity:

Nov-2007						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

Today

(b) A calendar displayed on demand

Figure 3.1: Example of extras on demand pattern

Visual framework pattern advises to design each page of systems with the same basic layout, colors, and elements styles. The system will look the same which is easier then to use, and users do not have to figure out new layout in every page. It increases users' awareness of different categories as they easily notice changes and differences among them. An example is given in figure 3.2.

A practical advise is to separate such design properties specifications from UI content in such as CSS style sheets. The metrics evaluation system implements exactly this pattern as separate CSS are applied to same layout in all categories, the system looks consistent.

4. Card stack

Implementing this pattern implies defining subsections of content in the same page. Each can be visible after a click over it, and one or more can be visible at a time.

In metrics evaluation query detailed query specification, this pattern is used since a lot of details should be available in such categories and displaying everything at once may be discouraging for users even to start thinking of figuring it out. Therefore, specifications of query are classified in stack panels, which users can control each section's visibility.

5. Good defaults

Good defaults pattern means to prefill some of the fields as best guesses. It can be very helpful for new users when they do not have to make all selections till they get a result, as well as for frequent users who do not want to



Figure 3.2: Example of using visual framework [Tid05]

enter all fields every time they want to perform a specific action.

Again, defaults were often used in metrics evaluation interface. For example, evaluation time period is pre selected always as the last year. In common metrics page, users can simply select only the metric and leave all defaults, which evaluate for last year, for all products, grouped by products.

6. Same page error messages

It is advisable to write error messages in constant and consistent manner, and on the same page where users made their specifications of inquiry for action.

At last, there were few of many patterns could be applied to different interfaces to practical perspective for the achievement of better interfaces design. Table 3.1 mentions almost all patterns, but organized under different categories.

3.1.5 Evaluation of Interfaces - Standards

Applications interfaces are designed for users, so they should meet requirements and users usability and experiences goals. Evaluation is not performed only for

Categories and their patterns	
Focus on users	Users normally intend by using a system to find, learn, perform actions, create, control and monitor, etc. Design can use some patterns to ease these functions, patterns like self exploration, instant gratifications, incremental construction, deferred choices, and spatial memory.
Content organization	Applications with lists of subjects, tasks, actions, categories, or tools need to be organized. Practical helpful Patterns are two panel selector, alternative views, one window drilldown, wizards, or extras on demand.
Navigation and Signposts	Patterns to care about navigation and reduce its cost have many options. Examples are using clear entry points, global navigation, pyramids. For signposts, sequence maps, breadcrumbs, annotated scrollbars.
Layout of Page Elements	Patterns for organization and layout of pages are visual framework, central stage, titles selections, card stacks, alignments, and responsive disclosure or enabling.
Actions and Commands	Using patterns like button groups, action panels, smart menus, having multi level undo, and commands history.
Showing Complex Data	Beneficial patterns for complex data arrangements, display, and relations are Cascading lists, tree table, local zooming, data tips, data brushing, making row striping, sortable tables, jump to item, and new item row.
Getting Input from Users	In forms and data input, patterns examples are using forgiving formats, structured formats, autocompletion, list builders, dropdown choosers, good defaults, and error messages on the same page.
Builders and Editors	Possible patterns can be smart selections, edit in place, composite selections, one-off mode, constrain resize, guides, and past variations.
Visual Style and Aesthetics	For the final looks of interfaces, good patterns include deep backgrounds, corner treatments, skins, hairlines, and contrasting font weights.

Table 3.1: Brief descriptions for possible patterns, [Tid05] for more details

existing interfaces or systems, it can also be performed within the development process to improve and enhance users and requirements goals. But this section is intended to discuss evaluation standards for existing interfaces.

Even for evaluations after system development, some approaches are designed to evaluate mainly based on user behavior. In Kirkpatrick's approach [Kir98], four levels of evaluation are discussed. The levels starts from users reaction, learning to check if users learned anything from using the interface system, behavior (performance), and finally result and impact that helps for new improvements. Such approaches are irrelevant in this chapter, since we are interested in evaluating existing systems interfaces based on familiar standards.

The following evaluation standards are usability heuristics from ISO 9241 [Tra03] that apply to interaction between users and systems. The main reason to present them is that most of them are going to be used in next section evaluation of existing metrics queries interfaces.

1. **Suitability** for the task where dialogues and interfaces should be suitable for the user's task and skill level.
2. **Self-descriptiveness** since dialogues and queries interfaces should make it clear what the user should do, can do, and should do next.
3. **Controllability** since users should be able to control the pace and sequence of the interaction.
4. **Conformity with user expectations** where systems should be consistent in all aspects, concepts, and tasks sequences terminologies.
5. **Error tolerance** where systems and forms should be forgiving, and not allowing serious errors.
6. **Suitability for individualization** where dialogues and queries specifications should be able to be customized to suit users of different experiences.
7. **Suitability for learning** where users should be able to learn using the system from the system.

3.2 Evaluation of Existing Metrics Interfaces

In chapter 2, tools of change request management and their importance were discussed. This section is devoted to evaluate existing tools interfaces, covering a description of their provided features at first. Then, interface design evaluation,

which is performed based on last section interface evaluation standards and across some of the designing rules and patterns as necessary.

Four commonly used tools were picked for evaluation, including Bugzilla, JIRA, Polarion, and Code Beamer. Each is evaluated in two major parts or sections, one for interface features, and the other for interface design.

3.2.1 Bugzilla

Bugzilla [BUGa] is an open source change request management system, developed by Mozilla organization [MOZ]. Version 3.0.2 is evaluated from [BUGb]. Bug is used as the term refers to a change request.

Features:

Bugzilla provides a framework to report bugs, where internal organization users and external customers can enter bugs. Searching existing bugs is also provided through an interface to filter needed bugs based on any of their properties. In addition to reporting and charts generation features.

Charts can be generated as different types like stacked, line, bar, and pie charts to visualize user defined metrics. Tabular reports to generate metrics results as tables. Number of bugs within different calculations and properties are the only supported metrics evaluations in Bugzilla, i.e. user can not define metrics to calculate time length periods or number of certain events on a bug like state changes.

More powerful supported charts are the new charts, which are used to generate reports change over time. Bugzilla allows different searches as data sets, which can be reused or more than one data set can be added to a chart, with user specified evaluation time period. Old charts are simple fast charts creating category, in which products and status can be chosen to generate line chart over automatically specified -last year- time period.

Design:

Bugzilla interface framework provides consistent visual and task oriented system. Three main entry doors into the system we will refer to as categories, which include enter a bug report, search existing bugs, and summary reports and charts. The following are some observations and comments about the Bugzilla interface:

- Consistent framework in terms of terminology and concepts, in addition to the visual outlook of all categories. For example, bugs properties and specifications are consistent across the different categories. Bugs filtering is arranged and visualized in the same way in both bugs search and reporting and charts generation queries interfaces.
- Filtering and queries options describe themselves, and users can click additional option for help that will redisplay the same page, but a help descriptive text is shown when the mouse over any option or widget.
- Users can perform actions of queries specification over and over without any harm, which encourage them to try more and learn.
- Conformity with users' levels is not always taken into consideration. In search bugs, two search categories exist, one for fast search includes only choosing products, bug status, and a keyword if need, then perform search process, and the other is detailed bugs properties filtering to search by. In reporting and metrics evaluation, users can choose to generate reports which will result in a tabular result, or generate charts after specifying its type. Only one possible page or category is provided, by which, users can specify their query and no category for fast metrics selections as common metrics is provided. So users should always start from scratch in order to specify a query unless it is saved.
- In the chart or report display page, it is always possible that a user changes the displayed chart type after clicking any different type under it, and the new type of the same result is directly displayed, so users do not need to go back to choose it from the query interface. The chart itself can be resized, but its attributes are not all possible to change, like evaluation time, granularity, and the fact that no weight on bug numbers is possible and only number of bugs can be evaluated.
- No serious or critical errors can result within any query specification, but it would be better if query options input errors are not displayed in a new page.

3.2.2 JIRA

JIRA [JIR] is a commercial issue tracker tool developed by Atlassian software systems. Issue term refers to the change request, which can be a bug, improvement, new feature, task, or sub-task.

Since the main JIRA site does not provide a free demo server anymore, another JIRA based system called GridGain computing system [GGSb] was used. Version 1.6.1 was evaluated from [GGSa]. The following is a description for its features, and comments on its design.

Features:

JIRA framework provides functionalities like create new issue, find issues, and browse projects that include common filters and supported reports for direct evaluation.

Find issues category is flexible to filter a wide range of parameters and issues properties. Filtering can be performed over projects, components, target versions, issue types, assignee, status, priorities, text based search across issues summaries or descriptions, time period of creation, update, or resolution.

JIRA reporting has an advantage of supporting age based and time tracking reports, with fixed number of possible reports to generate. In browse projects, a project or more can be specified, to which links to some fixed reports, common filters, and project summaries are displayed for evaluation. Examples of supported reports are:

- Issue tracking report that shows a progress bar and accuracy measurement for specified issues group (usually based on versions), in addition to a table of estimated and spent time values for each issue.
- User workload report shows workload by providing number of unresolved issues for each user, configured on projects selection basis. Version workload report is similar but more flexible to show workload for users in selected versions of a project.
- Other metrics like created vs. resolved issues, recently created issues, and average age can be additionally configured by basic filter of issues, evaluation time period, or days previously from now that should be included in the evaluation.

Although JIRA chart plugin was not available at the mentioned evaluated version, but it is be beneficial to mention that JIRA supports variant types of charts e.g. bar, line, and pie charts with illustrative labels and colors to visualize calculations.

Design:

The following are some comments on JIRA interface:

- JIRA provides an easy individual customization for its pages, specially the home page, which includes by default projects, common filters, saved filters, and few reports. These shortcuts save much time and effort for frequent and new users.

- Besides the home page, the system is organized as task oriented categories that can be chosen from an upper menu. The menu includes create an issue, search issues, and browse projects.
- In search issues category, a left side column consists of stacked cards for different filtering. The stacked panels pattern is good design option due to wide range of available parameters and properties. After filtering, resulting issues can be viewed on the rest of the same page as a table of issues. Each with some of its properties values as additional columns, where users can easily add, remove columns, or sort rows based on any of them.
- Browse Projects page provide statistics for a selected project, such as links for few reports, project summaries include brief statistics of issues distributed as e.g. their status, priorities, and assignees. Users can actually see some results of calculations in this page directly without any effort.
- Errors rarely occur due to the limitations of user specifications with fixed reports to choose from. When an error occurs, for example entering invalid date in some search issues fields, an error message is displayed on the same page, with a clear colored notification of where and what the error is.
- JIRA does not provide multiple levels of query specification, only some common metrics can be evaluated. In metrics calculations, the user behavior is controlled and limited by the system. An extra screen of wizard to select any parameters according to each metric is provided, which include maximum only three parameters to filter.
- Users can easily learn how the system works as soon as they understand what the reports/charts are. Additionally, many of issues properties widgets are combined with a help icon, which display a new popup window with a detailed description of what these properties values mean. Nevertheless, sometimes descriptions are not located on the main interface as they should be, e.g. reports have no descriptions shown unless the user chooses it.
- The system is suitable for users interested in its fixed possible reports (mainly age based and time tracking).

3.2.3 Polarion

Polarion [POLb] is a commercial application lifecycle management tool developed by Polarion Software GmbH. Version 2.5.0 was evaluated from [POLa]. A work item term is used to refer to change request, which can be a task, requirement, change request, or a defect. The following are an evaluation of its provided features and interface design.

Features:

Polarion dashboard provides demonstration of required evaluations and work items based on projects. Search work items can be performed in a very flexible mechanism, many properties fields can be specified and combined like text of description, title, and comments, other fields such as priorities values and their changes, assignee, type, status, and time point, in addition to Dates and approval properties based search. Search queries can be saved and/or executed to show list of resulted work items with ability to show details of each.

The support of time series reports is limited in Polarion. Traceability, accuracy, and estimation reports are mainly supported. The only two supported types of charts and reports are:

- One line chart for work items trend, which demonstrate the number of open, new work items, and unresolved requirements. Last 30 days are always shown in the chart time series.
- Many other reported calculations are shown by single stacked bar charts, which have fixed length and split into several colored parts to represent work items groups of the calculation. Supported reports include unresolved work items by priority or severity, resolved vs. unresolved work items, assigned vs. unassigned, scheduled vs. unscheduled, project plan accuracy, estimate accuracy for work items, and tractability of work items to commits and vice versa.

Design:

Polarion is designed as a dashboard includes most of users needs. It can be further customized on interface level to arrange options and results. The following are some comments on Polarion systems from interface design perspective:

- On one page, the dashboard, users can perform any of the provided tasks. Variant tasks are integrated within the same page, which consists of left side column for fast selections/tasks and rest of the page for results display.
- The left side column integrates project selection, task selection, and shortcuts link to common search queries. The integration allows fast moves from one task or property to another. For example, when a user selects to show unresolved work items, he or she can select afterwards a different project or work item, which reloads the result part of the page to show the new results. This mechanism is efficient for frequent users of the system, but including

everything integrated in the same page can be confusing for new and non expert users.

- Variant visualizations of work items search results can be selected, like tables, trees, live plan, or matrix. Selecting any row shows immediately the work item details. Performing a search is flexible and can be performed on two levels, fast common selections or building a complete query search from a new -on demand- pop up window.
- Description and help is weak in some points of the system. The small tips on mouse over are beneficial, but widgets, categories of tasks and reports has no details, and their help icon direct users to a very detailed and long manual for Polarion supported portlets. This affects the learnability negatively.
- Despite of variant charts types limitation, the most used single stacked bars are expressive according to Polarion supported calculations. For example, unresolved work items by priority report shows a bar with different colored parts and their lengths represents each priority field. The number of work items is also written on each part.
- Errors rarely present in the system, the functionalities are performed within selections does not give chance to users to go in wrong direction. In rare occasions where user may enter false values like in date fields of work items search, it is totally ignored and users are not notified, instead good defaults are used and the search is normally executed.
- In summary, main tasks are not explicitly divided by different categories. All supported reports and charts are displayed in the main project page, which can be a fast and enough feature for Polarion users to visualize reports on project basis. Individualization is the worse among this section evaluated tools, and the system is only suitable for users who are interested in its fixed possible reports (mainly about traceability, accuracy, and estimation).

3.2.4 Code Beamer

Code Beamer [CODa] is a commercial collaborative development platform, developed by Intland GmbH [INT]. It provides change tracking capabilities in addition to task and workflow management. Version 5.0 Beta was evaluated from [CODb]. The term tracker item is used to refer change request. An evaluation of its features and design is performed as the following:

Features:

Code Beamer provides a collaborative environment for change request (trackers) management, reports, and charts. On projects basis, trackers can be searched using summaries or titles text, tracker type, severity, target, status, and resolution.

In trackers category, which is visible after a project is selected, summaries of different calculations are shown. Similar to Polarion, there are fixed supported calculations with limited time series configuration, and only line and stacked charts are available. Examples of supported reports by charts are:

- Simple line chart for Last 7 days changes to show the number of submitted, edited, and closed track items. And Last 7 days trend as stacked bar chart.
- Charts to demonstrate bugs by date, bugs by severity, tasks by date, requirements changes by date, and estimated vs. spent task efforts by hours. Flexible data series of these charts are configurable by specific last period from today, or manual start and end date. Granularity is not possible to specify.
- Few reports grouped by status and severity are supported for different trackers types like features and bugs.

Reports category is used to show and customize groups of filtered tracker items as tables. A report can be created, customized and saved, or executed to show its tracker items in detail. The process of creating or customizing a report includes specifying projects, track item types, title and description, visible columns, and any additional filtering of items properties like status, severity, and platform. SQL WHERE query can also be used here for additional search and filtering.

Design:

Code Beamer collaborative system starts with 'My Start' home page, which is a full-blown wiki page and users can customize it by editing as they want. Different statistics can be added for fast observations of specified reports. The following are some comments on interface structure:

- Besides the start page, tabs for categories form entry doors into different aspects and tasks such as usage manual wiki, projects, trackers for supported charts, reports of specific filtering using tables, and forums for discussions, announcements, and FAQ.
- Reports can be created and customized using wizard of 5 steps, which is a good idea due to the big number of possible specifications (described in

previous features section). Each wizard step is self-descriptive and can be easily understood. The general interface categories and reports are also self-descriptive, more than other tools.

- Similar to Polarion, since fixed functionalities and charts are provided by the system, no serious errors can result. In reports wizards, possible input errors or requirements are identified and users can not proceed to next step unless they fix them.
- Tracker summaries category includes four categories of charts, each lead to a page of multiple charts with a description for each.
- In some points, like moving between different categories of charts, the browser back button is the only way to go back to the list of charts categories. Browser back button is inconvenient to use in other places like reports customization wizards as an annoying browser security message keeps showing, alternatively only wizard's back buttons can be used to move back to previous step.
- Choosing a project is always a first step before proceeding to reports or tracker charts categories. If a user is in charts page, he or she has to go sometimes to browse projects category in order to choose a new project. Moreover, it is not needed step for reports category since choosing a project is provided in one of its wizard's steps.
- Finally, it is always possible for frequent users to save their needed queries and provide their links in start or each category home page. This makes it with better individualization than Polarion.

3.2.5 Summary

The evaluated change request analysis tools share the main beneficial purposes of reporting change requests, search them, show reports of related evaluations, and visualize these evaluations as charts. Some are more flexible to generate variant reports, or different calculations and types of charts.

All of them support basic and advanced change requests search and filter. Using good defaults is a common facility that allows users to define what they need and leave other fields without need of detailed specifications. On the other hand, it is always possible to specify advanced filtering range from using regular expressions (e.g. Bugzilla), combine any property for complex search, to users' ability of using SQL WHERE part (e.g. Code Beamer).

As for reports provision, Bugzilla is distinguished by giving users the ability to define and filter their own reports in detail. This makes Bugzilla more flexible than others in the possible created reports. In design perspective, it makes other tools

easier for users, since they provide only fixed number of variant reports, which need no effort from users to specify, only one or two parameters are sometimes needed before getting the provided reports such as project, or version. The question is only whether these fixed reports suffice for users needs or not.

Bugzilla supports different types of charts like line, bar, and pie charts. Different types for different reports can be chosen, and evaluation time period is only possible in its new charts feature. JIRA supports more variant charts such as bar, line, stacked, pie charts, and progress bars. But the specific type is fixed for each of the supported reports, JIRA charts are mainly distinguished as time tracking and age based. Polarion supports only two fixed live line charts for the last 30 days, in addition to other variant also fixed bar stacked charts, which mainly care about accuracy and traceability. Code Beamer is much likely as Polarion, with fixed charting reports as line or stacked bar. A good facility over the last two tools is that all of its charts are grouped in one page, and evaluation time period can be directly specified to reflect all charts results.

Finally, all evaluated systems are provided in simple interfaces, which can get advanced in some aspects depending on users' desire. Any of them can be more suitable according to what kind of observations customers want to evaluate. As for bugs reporting and searching, all share the same scope and functionality.

Chapter 4

Requirements

The purpose of this thesis is to develop user interface for the change requests analysis tool (in section 2.2), this interface should allow the use of the functionality of the tool's core, and provide a user friendly interface to achieve easy and full evaluation of metrics, after specifying their query. In this chapter, the functionality and requirements of the interface system will be explained.

4.1 Introduction

4.1.1 Goal and Intention of the System

The intended system should provide a good Graphical User Interface (GUI) to specify queries and their manipulations to allow a usable metrics evaluation tool. It should adapt different levels of complexity to specify a metric, including simple or detailed GUI, and keep a low level specification using XML. In a short summarized description, the system should provide the following:

- **First level** of metric specifications, which is a category for commonly used metrics to provide a fast few clicks way of some metrics evaluation.
- **Second level** specifications to allow more detailed interface for defining queries of different custom metrics, this level of interface is also divided into various categories according to the types of metrics a user needs to evaluate.
- **Third level** of specifications to make it always available for the user in any category to generate only the XML of his or her specifications, change it or/and evaluate from the XML specifications.

- **Save queries** and have the possibility to manipulate or evaluate the previously saved queries.

4.1.2 Users of the System

The system originally initiated with the cooperation with Kisters AG. The different employees of Kisters AG will use the system, including the project managers, department managers, and quality assurance staff. The various potential users are considered in the different levels of the interface that allow them to use the system easily with any desired level of complexity to specify the needed calculations and criteria. Moreover, an open source version is developed with few changes to accommodate with the intended public use.

4.1.3 Assumptions and Dependencies

Mainly, the system will consider the modifications that are implemented in the BugzillaMetrics tool, to which this interface is to be developed, keeping the same separation between the system and its data sources (in Bugzilla database). The following are some related considerations:

- In Kisters AG version, the Bugzilla metrics tool is implemented on a modified Bugzilla 2.18.5 installation running on a MySQL 4 database [1]. The tool's assumptions that implied using concepts like cases, which include bugs as the software defects will be reflected in the interface. Additionally, all fields or cases properties available in the used Bugzilla version should be reflected in the interface consistently.
- Due to the flexibility in configuring the metrics in the Bugzilla metrics tool, the interface should provide the ability to conform to the possible calculations specifications, which is achieved by the XML level of the system's interface. The user should not be left alone struggling in the XML neither force him to ignore this facility. Therefore, jumps from the interface selection to its correspondent generated XML should be provided.
- The log in mechanism is using the authentication of Bugzilla itself, so users of the system will be allowed to use the system only when they are logged in the correspondent Bugzilla itself, and the session management is therefore maintained based on that.
- Two new tables were added to the Bugzilla database, queries table to store the users saved queries and some other properties like the owner and the type

if its public or private, and metrics table to store the common metrics (see next section).

4.1.4 System Environment

The required interface system is the frontend of BugzillaMetrics tool, it provides filtering to define queries of metrics, generate XML of these metrics. For the evaluation, it has to interact with the underlying tool, send the XML to be parsed and processed, and retrieve the resulting chart data that will be displayed on the interface.

As the underlying BugzillaMetrics access the database, the system should also have suitable mechanisms to access and retrieve any data from the database when needed.

4.1.5 Definitions and Concepts

Common Metrics

The common metrics are few chosen metrics that will be used in the common metrics page for fast user evaluations, these metrics were determined first based on some often used and needed evaluations, but it did not seem efficient and flexible mechanism for the future tool's use, so now the administrators who has access to the Bugzilla database is able to change them and add their desired common metrics to the metrics table that has been added for this purpose in the Bugzilla DB. Initially, the following metrics are considered as common metrics:

- Totals. This metrics calculates the number of open cases.
- Totals overdue. The number of open cases that are beyond their deadline.
- Backlog Management Index. It calculates the ratio between outgoing rate and incoming rate; they represent the number of resolved cases, and the number of added cases, respectively.
- Case lifetime. The age of current open cases, and the age of closed cases at their closing.
- Product work distribution by components. This metric calculates the totals for each component of a product.

- Original effort estimation accuracy. The original effort estimation accuracy is a measurement how precise the initial effort estimation for a case is compared to the actual effort required to close it.

Query

Query is used to refer to the complete metric specifications and chart configurations of an evaluation. A query can be saved by a user with its unique name, description, owner, and the type if it is for public use or private.

Before the user is able to evaluate or save a query, its XML is generated if it was created in the interface, and XML is validated and checked for its correctness in case the user made some changes of its XML.

Base Filter

Base filter is mainly used in the second level interfaces. It includes the base parameters to filter properties of the cases that are to be considered in the evaluation. These parameters are case type, products, components, versions, target milestones, case status, severity, priority, assignees, beyond deadline to consider only the cases that are beyond their deadlines, and actual effort.

Basic Filtering

Basic filtering is the selection of few chosen parameters of the base filter, which is used in the common metrics page where only such basic filtering is allowed, in addition to the selection of the metric itself, the evaluation period and granularity. For the base filter, it was decided to allow the selection of only products, components, version, and case type.

4.2 Functional Requirements

The interface structure has different levels based on metrics categories that are supported for evaluation. In this section, the functional requirements of the system are explained in detail.

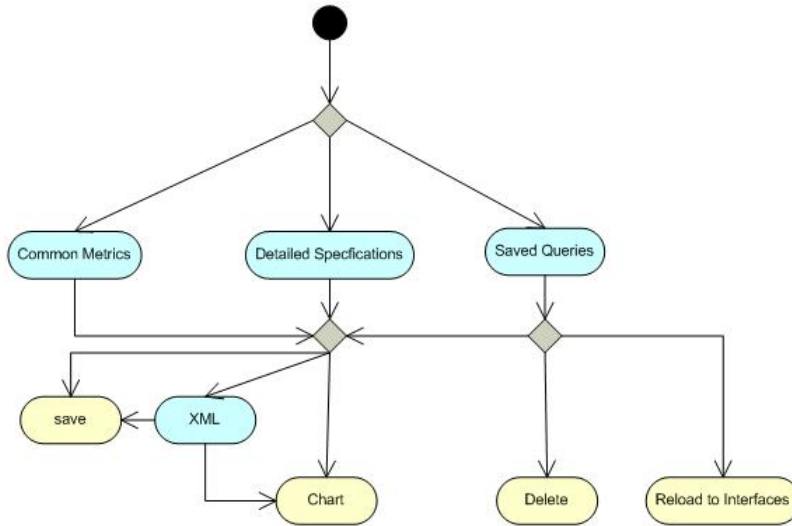


Figure 4.1: Basic interface scenario

4.2.1 Basic Interface Scenario

The high level scenario (see figure 4.1) includes going into one of the categories to specify a metric. These categories are common metrics, saved queries manipulation, and the detailed metrics specification categories. In common metrics and detailed categories, the user can define the query in the interface and then, evaluate it to get its chart, save the query, or reload the specification into XML, from which user can edit or/and save or evaluate. In saved queries category, queries can be evaluated, deleted, loaded into XML, or reloaded into their correspondent interfaces, from which the query was saved.

Next sections will cover each level of interface and their functionalities in detail, and the state management among them.

4.2.2 First Level Interface

This is the common metrics category (screen shot in figure 4.2), in which basic filtering is supported. The user is able to perform fast few selections including one of listed the common metrics for evaluation. The functionality in this category goes as the following:

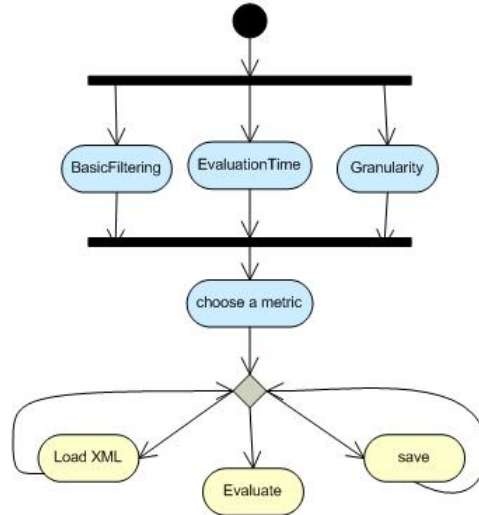


Figure 4.2: First Level Interface (Common Metrics) Activity Diagram

- The user has to select one of the common metrics.
- Filters any needed case properties values from the supported parameters in this category, in addition to the time period of the evaluation, and time granularity.
- Decides which action he or she desire to perform.
 1. **Evaluate** to generate the chart of the query.
 2. **Save as** to save the defined query, additional parameters are entered here like the query name, description, and query type of its private or public.
 3. **Save** to overwrite the query in case of a save as has been previously performed at the same category in the same session, if there was no saved query before, this acts as a save as.
 4. **Reload XML** to generate the XML specifications for the interface selections. From the XML, user can also perform save and evaluate.

4.2.3 Second Level Interface

This interface refers to the detailed specifications of metrics, without the existence of any predefined metrics; the query has to be specified from scratch, starting from

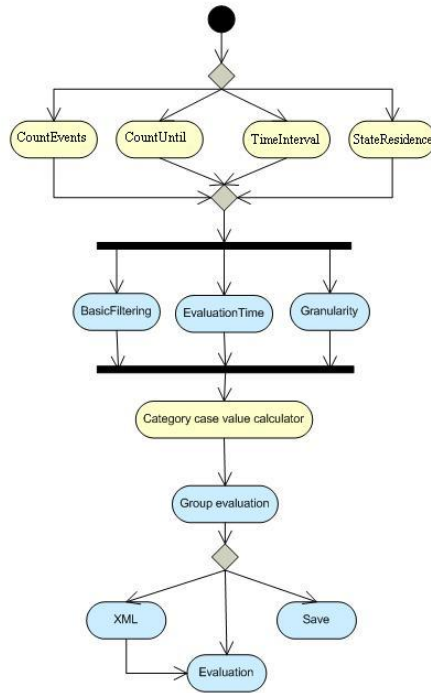


Figure 4.3: Sharepoints of Second Level Interfaces Specifications

base filters and including the specific metric core (case value calculators), ending in some chart properties specification.

In this interface level, various metrics are supposed to be supported. These metrics are classified based on the backend processing of the XML (see section 2.2), and the only difference between these classifications is the case value calculators, which are divided into four categories. Due to this differences of metrics XML specifications, which reflects classifications of groups of metrics, and to ease the usability of the system for interface design reasons, these categories should be present in the interface as four separate categories under the second level interface. Other than the metric core specifications, the four categories share the same properties and interface options (see figure 4.3), that can be summarized as:

- The base filter includes all the cases properties values, like products, components, versions, case state, and assignees.
- The evaluation time which includes the evaluation time period specifications and the time granularity.
- The metric core which differs according to the four classifications of metrics, it includes events specifications in different ways of calculations depending

on the category, these categories are count events, count events until an event occurs, time interval length, and state residence time. Each of them will be explained in details later in this section. Additionally, this part includes the group calculations of the specified case value calculators, a choice of a mathematical calculation for each evaluation must be here chosen, and its displayed name can be specified also. A maximum of two case value calculators can be specified by each metric, and they can be combined in the evaluation to calculate the percentage of the first over the second, addition of the two calculations, subtraction, division, or multiplication.

- The Chart configuration is to specify some of the resulting chart properties like the title, the y-axis calculation label, and the type of the chart if it's lined or stacked.

Count Events

This is the most used category in the second level interface, due to the wide range of metrics that can be specified within it. This category calculates the case values based on an event specified by the event filter, based on the weight field parameterized with it. Two maximum calculations can be specified, and combination of their calculations is possible to evaluate. The weight parameter can be default, that leaves the calculation to count with a fixed '1' for a case state, original effort estimation accuracy, age in days, or estimated remaining workload.

Figure 4.4 shows the scenario to specify this part of the query specification, the other shared detailed will not be explained here, like the base filter, the evaluation time period, and chart configuration.

Sequence of An Example: Calculating the totals overdue percentage metric needs two calculations. In the first, calculates the sum of cases under the event that filters all cases at end of time interval and state filter to include only the cases that are beyond their deadlines, and the second calculation is the sum of an event which filters all cases at end of time interval with default weight parameter A combination of percentage of the first calculation over the second calculation will result in the totals overdue percentage metric.

Count Events Until

This category (see figure 4.5) counts the number of times an event occurred for a case until another event happens. The two events are calculated by event filters, the first filter is for counting, and second is the event until which, the counting

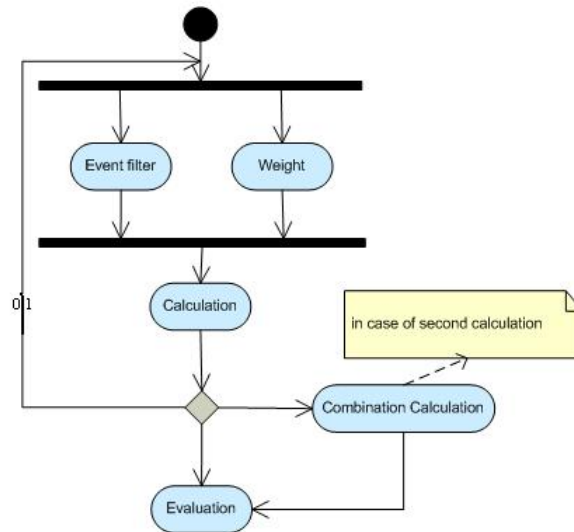


Figure 4.4: Activity diagram of events-based category case value calculator

stops. The user is also able here to specify a maximum of two calculations, and their combination if needed.

Sequence of An Example: One beneficial metric example in this category is the number of assignees changes before resolution to calculate how often the assignee of a case has changed before the resolution of this case. The metric core of this metric is specified by the first filter of a transition on the assignee property, and the second event filter is a transition of the case state from any status into resolved state.

Time Interval Length

This is a category to calculate the length of time in days between the occurrence of two events, which are specified by event filters (see figure 4.5 also), the core looks the same as the previous count events until category, but the specifications are mapped in its XML in a different way, where the first event is mapped as 'from', and second event filter is 'to'.

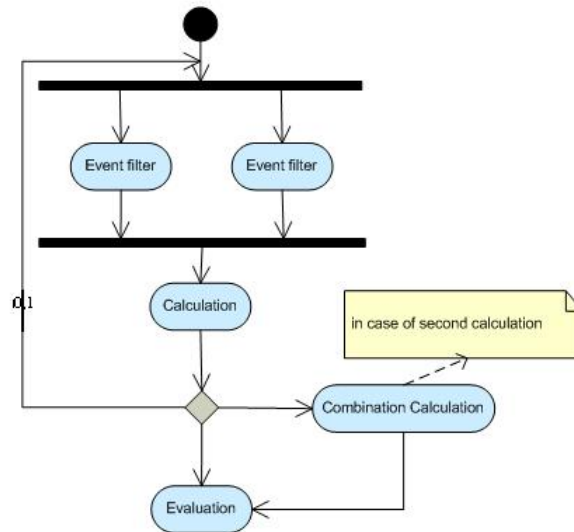


Figure 4.5: Activity diagram of count-events and interval-Length case value calculator

Sequence of An Example: The calculation of age before processing metric can be specified in this category. The core metric of this query is to specify the first (from) event filter as creation of a case to filter all cases from their creation time, and the second event is a transition of case state from any status to processing. The metric then calculates the time in days between the creation of cases and their processing state time.

State Residence Time

This is to calculate the time in days a case stayed in specific state until a time point of an event occurs. At the moment, only case state (status) property is supported in the interface since it's the only one that makes sense or is interesting to calculate its residence time, others can be supported in future if needed. Therefore, the user is able here to choose any of the case states (new, assigned, processing, resolved, etc.), and use the event filter to determine the time point to which, the evaluation is considered.

Sequence of An Example: Assigned state residence time is a metric to calculate the whole time a case stayed in assigned state. It is determined in the interface by

specifying the state as assigned, and the event filter can be specified to make the calculation include the open cases (still not resolved) and the closed or resolved cases at their resolving time.

4.2.4 Event Filter

The event filter (see figure 4.6) was mentioned before since it is used in all categories of the second level interface, in this section, its options and specifications in details will be covered. In specifying the event, more than one filter can be included and combined. The metric tool supports actually any number of combinations of events filters, but in the interface, only two event filters can be added and combined to specify an event since most metric do not actually need more than two, and it will lead into complexity of the interface if any number of filters is allowed to be added.

The combination of two event filters in each event specification can be combined by 'and' or 'or'; moreover, all of the options in each filter are combined by default by 'or'. In this way, not every option of the event filter needs a new event filter to specify it. The following is a description of each of the event filter options:

- **Creation** refers to the creation of a case, choosing this option filters all the created cases in the evaluation time period.
- **End of time interval** is a useful option to trigger calculations for all cases at the end of time interval.
- **Added to base filter** to include the cases that entered the set of base filter cases, this option is useful when the users wants to include all incoming cases within a time period, and creation does not suffices since some cases has been created before the evaluation period, but entered to the cases set from other product or any other set.
- **Removed from base filter** is used to filter cases removed from the base filter cases set within the specified evaluation time period.
- **Transition** to filter cases based on the occurrence of a change on one of the case properties, only transition on assignee and case status are supported currently. In this option user first chooses the field and then, specify the values of transition needed. For example, when the case status is chosen, two columns of status values are displayed, the first is for the values from which the transition is needed, and the second column is for the values to which the transition is configured.

- **Additional filtering** that pops up a dialog of the same base filter parameters including all the case properties that can be specified, and the user can filter more cases specific to this calculation, it is called the state filter.

4.2.5 Third Level Interface

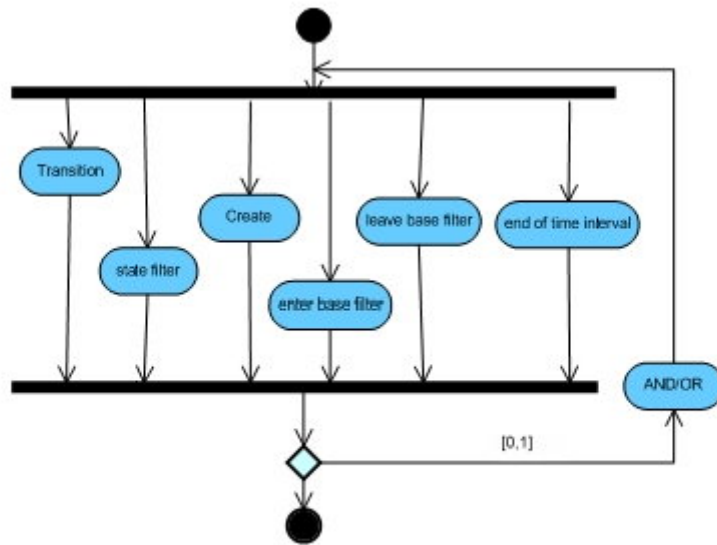
On this level, specifying the metric is performed via the XML. The user can specify here any metric he needs with full flexibility according the underlying Bugzilla-Metrics tool. This level exists in parallel in all categories of second level interface, in addition to the first level interface, XML page of any category can be displayed and user is able to specify his or her XML completely, or just modify and add to the current generated XML from the category.

Despite of the integration of this level with all other levels categories, it's not connected to these categories except the one way connection from each category to it and not the other way around. From each category, and after the needed selections specifications, load XML option can be performed to generate XML of these selections, and use can then edit, evaluate, or save the query. This mechanism is useful for the learnability of using XML in the system as users has the possibility to play around and changes few selections each time and generate the XML, with a reasonable time, the user can learn the whole XML mechanism of specifications if he or she is interested, and therefore, reach the maximum flexibility of defining queries.

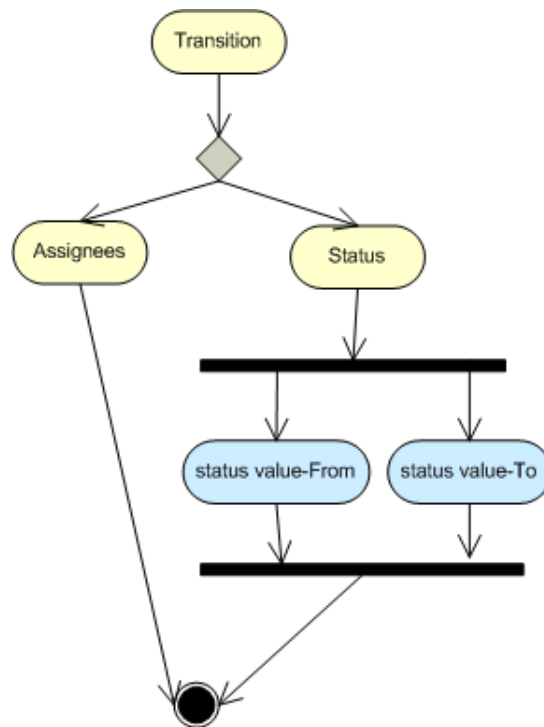
At this level, and since the available flexibility for users to edit or write their own XML, a full parsing to validate the XML is performed before any action like evaluating or saving is processed. Saving queries (section 4.4) from XML is preserved in the database as XML query category, which prevents the saved query from being reloaded into interface even if the user did not edit the XML; it is only supported in this case to reload it to XML specifications in the saved queries page. Therefore, users should be aware of this fact before saving from XML regardless in what category they are.

4.2.6 Saved Queries

This category is to manage and manipulate the saved queries, all the user's queries and other user's public queries are displayed here. Each saved query should have a name, description, type if it is public or private, owner, metric specifications as a string for its xml, chart configuration data as XML string also, and its category from which it has been saved; it can be from common metrics page, count events,



(a) Event specification



(b) Transition specification of an event filter

Figure 4.6: Activity diagram of an event specification

count until, time interval, state residence time, or from XML category. This property is important for the reloading action that is supported in the saved queries manipulation. The following are the possible action and manipulations supported for the saved queries category (see figure 4.7):

- **Evaluate:** The user is able to evaluate his saved queries and any of the public queries that has been saved by other users. More than one query can be selected and multiple evaluations layout can be performed, each in separate chart at the same page.
- **Reload XML:** The XML of any of the listed queries can be retrieved, and evaluated from the XML page, or overwritten after editing if needed. Naturally, only one query can be reloaded at a time.
- **Reload to interface:** Any of the listed queries can be reloaded to its correspondent interface category with the selections displayed on it, the metric specifications and chart configuration XML will be parsed in this process, and all parts of the XML will be mapped and displayed as the correspondent selections in the interface. Queries that are saved from the XML page can not be loaded into interface, only reload XML option is allowed in this case.
The reloaded query can be overwritten, saved as another query, or evaluated again from its category. The public queries are allowed to be reloaded also, but overwriting option is not allowed in this case.
- **Delete query:** The user can delete only the queries that he or she owns. If the query is owned by the user and its public, an extra warning message has to be confirmed since users' public queries could be in use by other users. Public queries of other users surely can not be deleted.

4.2.7 Evaluation and Charts

The evaluation result is represented as charts, they can be lined or stacked area charts (see figure 4.8) according to specifications in the chart configuration of the evaluated query. The x-axis shows the range of the time period divided into the slots of granularity. The range values are determined automatically according to the metric result values, but the interface should allow the user to label the range axis as needed, in addition to the title of the chart that users should also specify within the interface.

The data sets in the chart are grouped by the grouping parameter in the chart for the calculation specified, if two calculations are calculated; only the combination calculation will be considered. Although multiple calculations are not supported

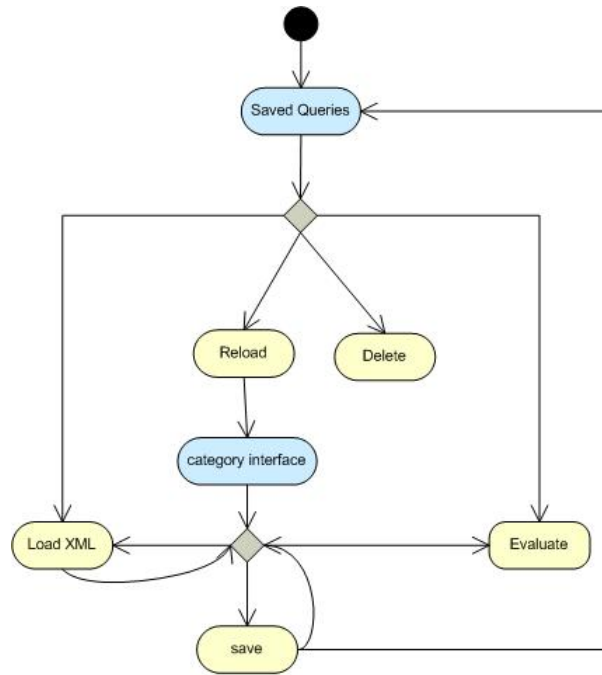


Figure 4.7: Activity diagram of the saved queries category

for display in the interface, it should still be possible to display different calculations in case of specifying them in the XML. Finally, the legends are automatically formulated based on the grouping parameters and calculations names.

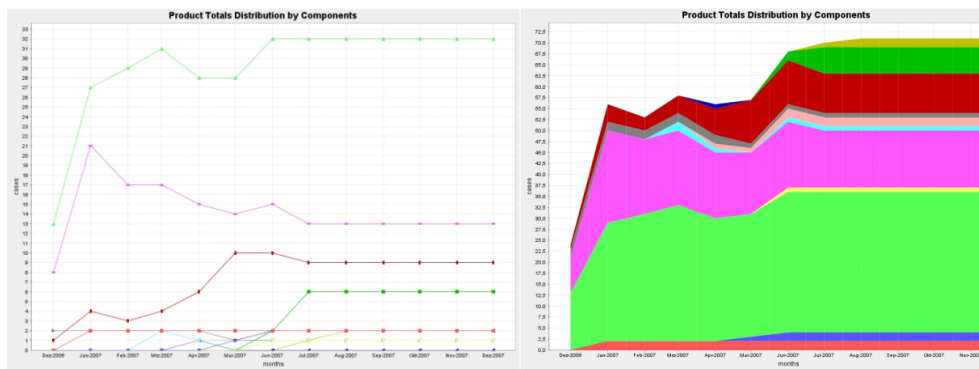


Figure 4.8: Example of supported charts types, line and stacked

4.2.8 State Management

The interface should preserve some level of consistency between its different related levels and categories. There are many options and mechanisms that can be adopted to manage the system states, specially in the presense of different categories, and evaluation result for each. In addition to integrations between each category and third level interface. The following are some of the aspects that should be considered in the system states management.

- Different selections and evaluations are separated in each of common metrics, and second level interface categories, which means user can be in one of the previous categories, and goes at any time to any other category for new filtering, if he/she goes back to the previous category he/she was in, the same selections will remain as it was, in other words, the initialization of a page is performed only once at the first show of a category, then any show of the same category will be preserved and the same as it was before leaving the category. By this mechanism, user can be defining different queries in different categories at the same time; each query is kept in its category.
- Regarding the third level interface (the XML), and as explained before, it exists within each category, and connected in one direction; the user can always reload the interface filtering into XML, evaluate or goes back for different filtering and proceed again to the new XML. Each third level interface is grouped with its category, if a query is saved, it can be also overwritten from the category or from its XML.
- The back button of the browser is fully supported and works between the different categories or from the chart display page back to its category, an additional back button is added in the chart page to direct back to the category of the evaluation.

4.2.9 Authentication Management

An authentication mechanism must be implemented for the interface system. Users should not be able to access the system unless they are authorized.

Different mechanisms are considered to apply the authentication. The main focus will be on allowing only the users who are logged in the correspondent Bugzilla to access the system.

4.2.10 Database Requirements

In addition to the need of Bugzilla database adaption, other data of the system should be considered to be stored in the repository. For example, the common metrics that can be stored and retrieved in and from the database, and the saved queries to store/display queries and their properties like name, description, type, metric specifications, and chart configurations.

4.3 Non-Functional Requirements

4.3.1 Requirements of User Interface

The interface system should be a web based application, which should be integrated with the backend analysis tool (BugzillaMetrics), which is developed using JAVA. Additionally, other important factors should be taken into consideration are efficiency, getting relatively fast responsive application. Such factors are beneficial for the future system adaptability and its use.

4.3.2 Performance Requirements

The system should respond in a reasonable time when an evaluation is requested. The database at Kisters AG is subject to increase as new cases will be added within the time. Since the queries are created based on the filtering and evaluation will be executed by the BugzillaMetrics tool, the main interest here is to achieve a fast XML generation and send this XML to the tool through the server side. Generating the XML from selections is to be performed on the client side on the browser to achieve it fast; on the intended technology, it can be performed in around one second or less. Widgets of the interface should interact with an immediate response to achieve a usable efficient interface.

4.3.3 Maintenance Requirements

Since the installed Bugzilla can be variant, and different versions can be considered; the developed system should be always possible to work with any changes of Bugzilla DB. For this purpose, the values of case properties will be displayed in the interface from the retrieved data from the DB itself. In this case, any changes of the DB will reflect automatically into the interface values. The system should be

extensible by new categories if needed in case of new extensions in the supported metrics of the BugzillaMetrics tool.

4.3.4 Security Requirements

The system will be used for internal purpose in the company. Unauthorized users should not be allowed to retrieve the sensitive data of the DB. Moreover, the design of the GUI will ensure the control of the processes that can be performed by the users.

However, users of the system will have no access control on queries, for example if a user intends to limit the visibility of a query on only group of users. Either the query is private and no other user can see it, or its public and all users can access it. Or if the organization does not want each user to define metrics on all products, this is not possible.

Chapter 5

Architecture

This chapter discusses the architecture of the system, the categories and their functionality implementation design. In the first section, the Google Web Toolkit (GWT) technology that is used to implement the system is described, and its general architecture. The second section goes into details of the architecture components of the interface system and their design patterns.

5.1 Google Web Toolkit (GWT)

GWT [GWT] is an open source Java software framework, in which developers can create AJAX web applications using pure Java. AJAX stands for Asynchronous JavaScript and XML, the idea of AJAX is to develop more responsive web applications with fewer exchanges of data between client and the server, and not every request of change on the browser needs the server so the page does not need to be completely reloaded on every request. This result in more interactive, and fast web page. GWT allows the development of dynamic web applications easily using any of the Java tools the developer prefers, with no extra effort to check the functionality on different browsers as it provides browser independent product after deploying the application.

One of the core concepts of the toolkit is the GWT-compiler, which compiles and translates the client side java into java-script and HTML pages runs on the browser. In this section, the architecture, features, advantages and some difficulties using GWT are described.

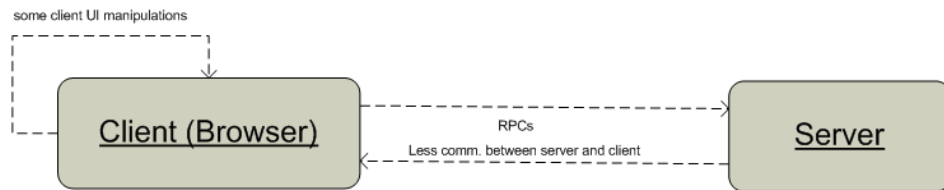


Figure 5.1: Client-Server load in GWT

5.1.1 General GWT Architecture

In GWT, the Java code can be distributed on server and client sides; after deployment, the client side code runs as java-script on the browser, and any business logic code can be processed on the server. Unlike the traditional web applications, this result in less pressure on the server with less work load, and end users can interact with the interface on the client without the need of fetching all data from the server (see figure 5.1), while interactions with server, when needed, can be handled through Remote Procedures calls (see section 5.1.2) that is used to pass Java objects back and forth over HTTP. The following is a description for some major components of the GWT.

- **JRE emulation library:** This is the supported part of the Java in the client side, as GWT includes implementations of Java-script for these libraries. The supported library is a subset of the Java 2 Standard and Enterprise Edition libraries (J2SE 1.4.2 or earlier), other unsupported parts are mainly not applicable to web applications like most of the packages of Java.io. Most of java.util and java.lang packages are in the emulation library supported by GWT. Code that is compatible with J2SE forms a major part of the translatable client code for compilation, in addition to the GWT UI libraries.
- **GWT UI Libraries:** These are the libraries of GWT to create interfaces components, and widgets on the web browser. They include many ready-to-use widgets like textboxes, panels, buttons, and grids; in addition to custom widgets a developer can create. GWT includes may fancy custom widgets also a developer can reuse.
- **GWT Compiler:** This is the heart of the GWT, which translates the client side Java into java-script that will run on the browser after deployment. At first step, a check against the JRE emulation library and the compatibility with J2SE is performed in order to identify such errors at early step.

However, few issues are need to take care of, and may cause problems, like the use of Serialization and regular expressions. GWT does not support the

standard Java Serialization; instead it provides the RPC calls that automatically generate serialization for the objects transferred between the server and client. Also, sometimes regular expressions in Java may have different meanings than the ones in Java-script, so developers should take care of this fact when working with regular expressions, like in `replaceAll` and `split` methods that use regular expressions, but differences are not major or critical, e.g. empty strings will not be retrieved in `split` method when they are intended to be.

- **Server Side:** At server side, the developer is free to use any Java package he or she needs. The side's code is not translatable and is not considered by the GWT compiler; after deployment, this code runs as java byte-code on the server.
- **Hosted Mode:** Developers may use any Java Integration Development Environment (IDE) they prefer while working with GWT. They can edit, debug, and test their applications in the hosted mode, in which, the code is executed in the java virtual machine as java byte-code, without being translated to java-script. An embedded browser window will show the application when the developer runs the code, this provides a fast mechanism for developing and testing, integrated with the used IDE.

5.1.2 GWT Features

In this section, some of the interesting features provided by GWT are described; most of them are considered success keys and advantages achieved by the toolkit. Some of the features explained are the dynamic UI framework, GWT RPC, history management, fast efficient deployment, and the integration of Java-script and CSS within GWT.

The dynamic User Interface

GWT UI classes' library includes many of the UI components that are similar to the Java UI packages like SWING and SWT, but instead of the pixel oriented creation of these components, widgets are used in GWT that are created as dynamic HTML. The following are some of the aspects and components of GWT UI library.

- Widgets form the base class for UI components that can be used and displayed on panels. GWT library includes wide range of different widgets like textbox, buttons, tree, table, dialog box, stack panel, suggest box, disclosure panel, rich text area, and different layout panels like vertical, horizontal, and dock panels to contain other widgets.

- Custom Widgets are easy to create in GWT using composites, which may contain other UI components as panels. This makes it easy and efficient to create complex widgets by grouping widgets, or create new widgets from scratch. Custom widgets are beneficial for reusability during the development of complex interfaces where developers can make a composite class including any combinations or groupings of widgets, and reuse it as needed.
- Interface Listeners are used to handle events on the browser on widgets by defining methods for this purpose. Widgets call these methods when triggering an event. Implementation of specific listener interfaces in a class allow this class to receive the associated events, this implementation includes the handling of the event, and checking to which widget it belongs to perform the needed action.
- Cascading Style Sheets (CSS) are used to enhance the look of the user interface, each widget can have its own default style name that can be used to add some attributes like size, font, etc. CSS are specified in separate files in GWT, and complex CSS can be defined to do more than just simple attributes, like identifying specific behavior to different widgets.
- DOM Interface can be used to manipulate the browser's DOM directly instead of widgets, but it's rarely used and preferred only when it is critically needed as using widgets is faster and more efficient to use.

Remote Procedure calls (RPC)

RPC is the mechanism of interaction between client and server when needed. Most of the UI related logic is implemented on the client, and server is not needed, but some other implementations can be performed on the server, and this gives the client more efficient performance with reduced bandwidth and fast response time. RPC is used to fetch data between server and client, and allows easy sending of java objects over HTTP between them.

The architecture of making RPC (see figure 5.2) is that there is a Service interface on the client that includes the methods that will be called and implemented on the server. This service extends a marker interface called `RemoteService`. Additionally, `ServiceAsync` interface must be also created on the client side, which makes the call asynchronous (does not block the user till it completes). The asynchronous interface includes the same methods of the Service with specific modifications like passing the async callback within the methods that is notified when the call completes.

RPC are easy to make and form an important efficiency factor of GWT, within the asynchronous nature of calling, execution will not block the other implementations,

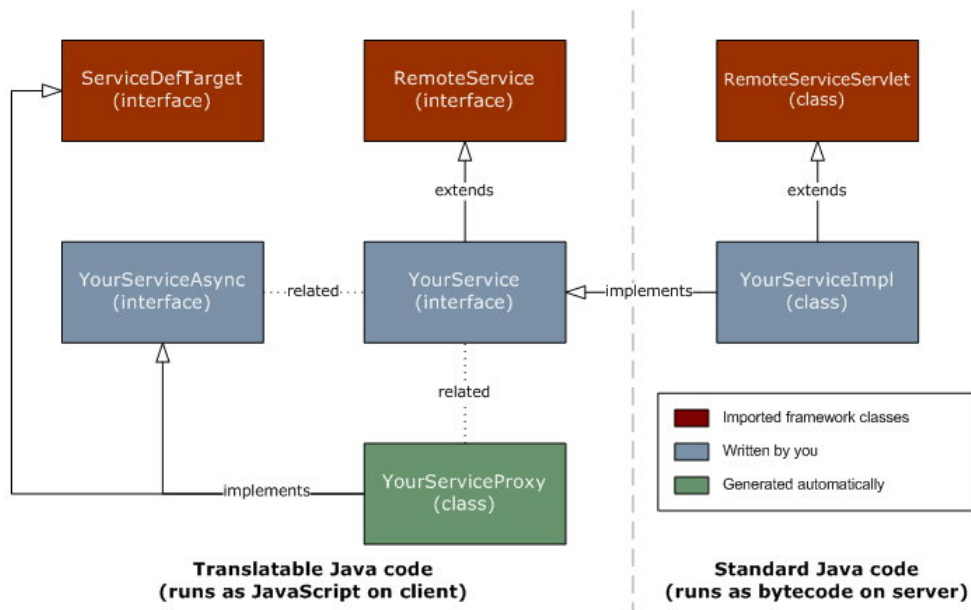


Figure 5.2: GWT Remote Procedure Calls Architecture [GWT]

and this saves run time as considering this example; if the call itself needs 1 second to execute, and the server implementation needs another 1 second to perform, then unlike the traditional web applications, it needs here 1 second in total, since it starts the execution of the server side method at the start point when the caller initiates the RPC and calls the Service method.

Browser Support and History Management

The GWT application can work across any browser in the web mode. GWT provides a convenient browser's history management support. History class allows the interaction with the back/forward buttons of the browser, adding of items (tokens) to the browser's history stack. Developers can programmatically move backward/forward, and display the correspondent state when a history stack item is asked for. However, slight problems were faced till the moment in this aspect, where custom button was meant to behave as the browser back button programmatically. Although it does not affect the main functionality and main tasks of the system behaviour, but further investigations are to be studied to overcome this misbehavior.

Deployment

Deployment is to run the web application on the normal browser, in which the code runs as java-script. For testing purposes, developers can compile the application code from hosted mode or directly, to display the product in what's called web mode. Hosted mode contains embedded hosting server that makes this compilation possible. To deploy the product in the final phase on a URL, some files and packages need to be dropped on the deployment server as following:

- On the `.. \server-directory\webapps` directory, drop the few XML, HTML, and java script files that will be generated after compilation of the client side code.
- On the `.. \webapps\WEB-INF` directory, a `web.xml` file must be created in which the services are mentioned with mappings to their service implementations.
- On the same directory, two other directories must be added. `classes` directory to include the java classes of the application with the same application project directories hierarchy, and `lib` directory to include the JAR files libraries that are used in the application.
- That's it, the application now is ready to use after restarting the server, for example, `http://1.0.0.1:8080/appl_name.html` where `1.0.0.1` is the host on which the server is deployed, `8080` is the port it is listening to, and `appl_name` is the entry point class name.

5.1.3 Project Structure and Modules

When GWT project is created, the general high level structure is automatically created, but what is this hierarchy and where to develop the application code and affiliated files. This is explained in the following project packages:

- `com/example/frontend` is the main root package under the source code of the project, it includes the other different packages, in addition to the modules XML file in which the entry point class of the application, source paths for code distribution, and any inherited modules are specified. Most of the paths are generated automatically by GWT, but additional settings can be added here when for example a new module needs to be inherited.
- `com/example/frontend/public` package includes the main HTML page of the application, CSS files, and any other files that are to be used in the application by an asynchronous HTTP requests.

- `com/example/frontend/client` is the package where the translatable java code is written, as UI classes, any client side manipulations, and services. Any sub-packages can be added here to arrange different groups of classes in big size projects.
- `com/example/frontend/server` is the package where the server side classes are created. No limitations on java standards are applied in the code of this directory, where access to databases, connections to other GWT or non-GWT projects can be implemented from this package classes.

5.1.4 Challenges and Current State

The idea of using Java to develop AJAX dynamic web application is considered a promising achievement; however, few issues which may not reach to be called disadvantages since they are not closed-door obstacles and they need additional effort for beginners to be overcome.

- Client side java supports only JDK 1.4 at the moment, and developers need some time to get familiar with the new framework and classes implementations of GWT.
- Only asynchronous calls are available in the run time execution of the application. Although this is an efficiency advantage, but sometimes it causes few problems when some methods are supposed to finish executing before another is called. Some tricks can be used to overcome this problem like using timers, fire some event listener when the first method completes, and calls the second in the implementation of this listener.
- Poor documentation on some aspects, like deploying on a java server machine.
- Performance problems of the list boxes when they include large lists. This and any other problem can be solved at the moment by overwriting GWT methods by including native JavaScript methods within the Java code.
- New and developing tool. The GWT is quite new, and improving every day, and many problems were being solved by the GWT team while we are working on the interface system, so development speed and flexibility of the project that we started working on before 6 months would differ if we begin it now for example.

The current state of the GWT version is already interesting and achieving, but it is also promising in the near future while new fancy widgets are added every day,

and integrating with new functionalities is also growing fast like adding the support of using the Google APIs, for example Google Gears is already now launched to allow the web application to work offline. An important role that makes the GWT rapid growth is the fact that the tool is now fully open source and adapted by a big and strong company.

5.2 System Architecture

This chapter explains the architecture of the interface system, and the different components and categories implementation packages. The following aspects are discussed:

- General architecture and the integration of the system in the environment with other tools and database.
- Client side and server side packages are discussed first as an abstraction to give a rough idea of the functionalities of each side.
- Categories of the system and the widgets used of the second level interface category.
- XML selections builders package, in which the generation of the XML from the interface selections is processed.
- XML parsers that parse the XML of the saved queries to get their correspondent interface selections when reloading a saved query action is performed.
- Charts and evaluation process architecture.
- Authentication check mechanism that is performed at the initialization of the system.
- Database access methodology that is used to get all kinds of data from the DB is discussed at the last section, in addition to the details of the new two tables added to the Bugzilla DB.

5.2.1 General Architecture and Integration

The graphical user interface system (see figure 5.3) is divided into client and server side, each plays a role to form the interface and functionality of the whole system and they communicate through RPC calls to send Java objects of data back and forth.

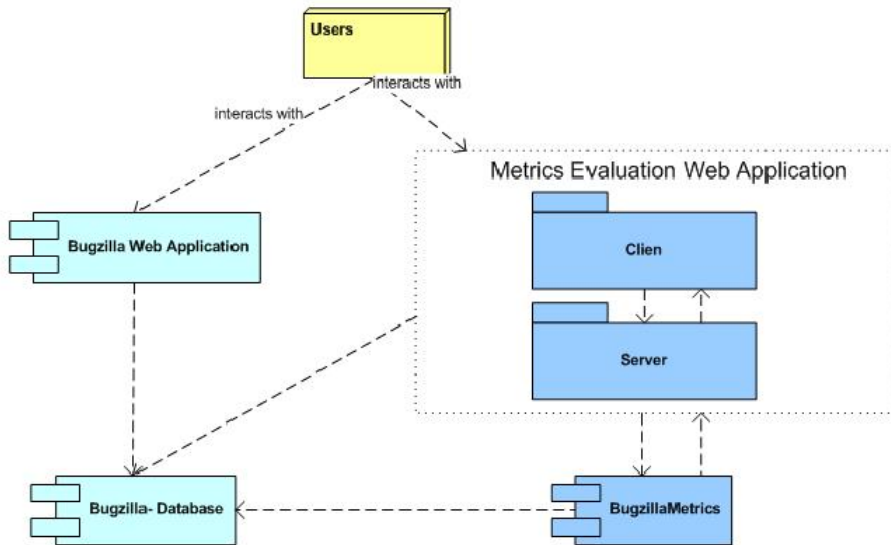


Figure 5.3: Interface system integration and environment

As the figure shows, the system communicates with the BugzillaMetrics tool to send the XML specifications of the specified query and get back the result chart data to be displayed in the interface. The Bugzilla database is used to get the needed data for the system, some data that are used and retrieved from the BugzillaMetrics tool are also retrieved through the tool for the interface system to preserve consistency in the mechanism of accessing them, other data are gotten directly from the server side of the interface like the queries, and common metrics.

5.2.2 Client and Server Abstraction

On client side, the translatable code to java-script is implemented that runs on the browser. The main tasks performed on this side are building and displaying the used widgets of the user interface, in addition to the XML generation from the UI selections and parsing the saved XML queries into UI selections. The following are the descriptions of client side packages (see figure 5.4):

- The base client package that includes the entry point class `MetricsUI`, in which the initiation of the system is handled as authentication check, adding the system categories, and how these categories are displayed.

- `gui` package includes the main categories implementation as `MyQueries`, `CommonMetrics`, `EventsBasedEvaluation`, `CountEventsUntilEvaluation`, `TimeIntervalLengthEvaluation`, `StateResidenceTimeEvaluation`.
- `widgets` package includes some of the used widgets in the categories, each class of this package is a custom widget that can be reused in more than a category or within the category itself. Some of these widgets are the `EvaluationTimeWidget`, `BaseFilterWidget`, `EventFilterWidget`, `TitledPanel`, and `Dialogs` for save or state filter.
- `data` package includes an interface for the filling methods of the Bugzilla data base data like assignees, case status, components, and target milestones. In addition to `BugzillaFacade` class to implement the retrieval services for these data from Bugzilla database.
- `xmlSelectionsBuilders` package includes the classes to write the XML for selections when evaluation or loading XML actions are performed. Classes to generate XML for each category and widgets are in this package.
- `xmlParsers` package handles the parsing of the saved query XML and fills all retrieved data in parameters that will be used in the categories to identify and display the selections when a reload query is performed.
- `rpcServices` package includes the used RPC services for the communication between the server and client.
- `serializedObjects` package includes the classes for the retrieved data from the server as serialized objects, like `queries` class that has the attributes values of a query, metrics, assignees, and products.

The server is the side where any java implementation can be handled, the interface system deals with bugzilla database and the `BugzillaMetrics` tool from this side, client side makes requests for more data processing that are implemented here, the following classes are implemented on the server (see also figure 5.4) with short descriptions:

- `DataServiceImpl` class handles retrieval of the base filter parameters data from the Bugzilla database data, like products, components, assignees, status, severity, priority, and target milestones. The methods do not access the database directly, rather through the `BugzillaMetrics` tool.
- `CommonMetricsServiceImpl` class implementation for the `CommonMetricsService`, it retrieves the common metrics from the metrics table of the Bugzilla database directly.

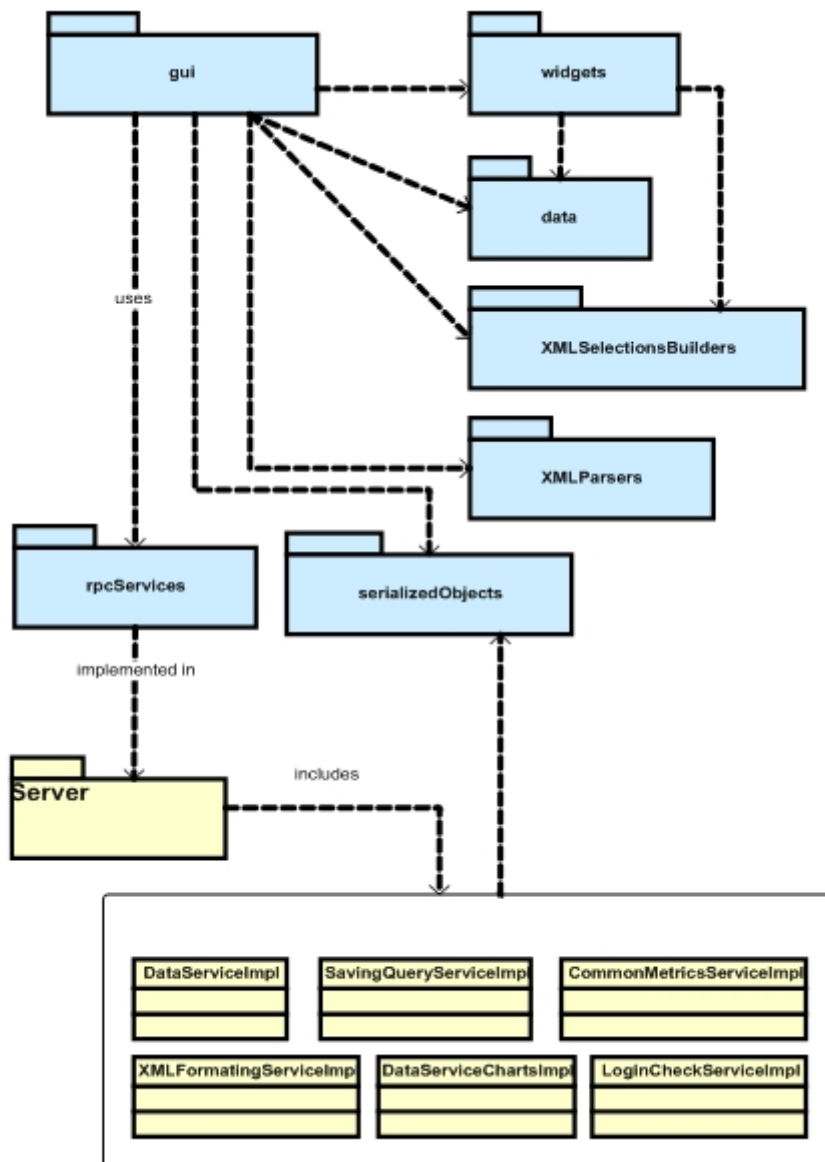


Figure 5.4: Main packages of client and server packages

- `SavingQueryServiceImpl` handles the retrieval of the saved queries to display them on the queries category, and implements the methods to save or overwrite queries when such action is performed by inserting or updating the queries Bugzilla database table.
- `DataServiceChartsImpl` handles the communication between the system and BugzillaMetrics tool when an evaluation is requested. The generated XML on the client is processed here by sending it first to the `calculateDOM` method of the `BugzillaMetrics CoreFacade` class to retrieve the metric run calculation element. The metric run result is processed again by the `ChartFacade` to create the chart and get it as `JFreeChartObject`.
- `XMLFormattingServiceImpl` class makes the generated XML looks better; none of the GWT classes supports the display as XML. This class takes the string of the XML and sends it back after formatting it to be displayed.
- `LoginCheckServiceImpl` handles the authentication of users of the system based on the browser cookies and their IP addresses comparing with data stored in the database. Users will be required to be logged in the correspondent Bugzilla when they use this system.

5.2.3 Second Level Interface Widgets

The first and third levels of interface are created using simple widgets of GWT due to the simplicity of the interfaces themselves. In this section, the used widgets and the creation of second level interface categories are discussed. Many custom widgets are created separately since they are reused in all of these level categories. Regardless the packages of the used classes, the following are the classes and their connections to initiate the second level interface categories (see figure 5.5):

- `BaseFilterWidget` class handles the base filter parameters look and display, which is used as the first widget for the filtering of all second level categories.
- `EvaluationTimeWidget` class includes a panel to display the evaluation time period options and granularity. It is also used in all categories.
- `ChartPropertiesWidget` class includes the chart configuration options like the labels names and chart type.
- `EventFilterWidget` class includes the event filter list options, every category uses this widget but it differs how many of this widget is needed

according to the category. It uses `StateFilterWidget` and `StateFilterDialog` to display consequent components like when choosing the transition option that may need the status fields twice again, or state filter that requires a state filter dialog to choose calculation specific case properties parameters.

- `StateFilterDialog` class is a popup dialog that includes the base filter parameters again to select additional filtering for the event filter.
- `WarningDialog` is class of a dialog that will appear when a warning about overwriting or deleting a public query, and a confirmation is needed.
- `StateWidget` class is only used in the state residence time category, it includes a choice of a case property, only case status is supported at the moment, and selection one or more of the case status values to evaluate its state residence time. This widget uses `StateFilterWidget` to get the status list instead of getting it directly from the base filter widget through a new object due to efficiency reasons.

5.2.4 XML Selections Builders

This section discusses the loading XML process from each category. Building the XML from interface selections is not only performed when load XML action is requested in the interface, but also it is the first step when an evaluation of a query is required. The events based category is discussed first in details and in the same section later, other categories are briefly described with the variation points and differences.

To generate XML from UI selections, all widgets has their own XML Selection classes (named consistently as `classNameSelection`), each includes a main `generateXML` method responsible to generate XML for that widget selections. Besides, each category has also its own XML Selection class that is the first target to call when XML generation is required. These classes add the suitable parent nodes and calls all included widgets XML Selections generated seperately in order to gather their XML into the main metric XML. The following is the sequence of objects and methods performed when building an XML from an event based query specification (see figure 5.6):

- The category interface class calls the `calculateXML` method that is a partially different in each category; the next actions are performed in this method.
- `EventFilterSelection` object is created to send the one or two event filters specified in order to filter their selections into values.

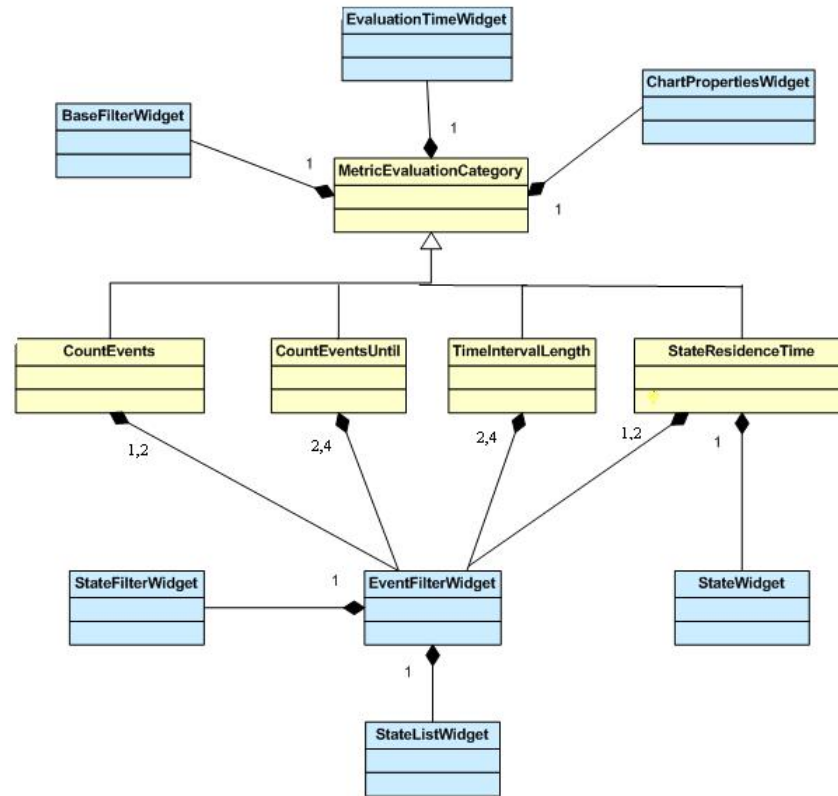


Figure 5.5: Main widgets classes used in second level interface

- WeightSelection object is created to send the weight list(s) to fill their selection as values for later generating XML use.
- GroupEvaluationSelection object is initiated to send the calculations choices and their names.
- EventsBasedSelection object is created to send the EventFilterSelection and WeightSelection objects in order to specify where and how the filter and weight XML will be generated since these sent objects can be created with different categories but for example under different parents names.
- generateXML of GroupEvaluationSelection is executed to build the XML of grouping evaluations.
- generateXML of EventsBasedSelection is executed to start building the

case value calculators of the category, and request the underlying event specification XML builders like event filter and weight.

- The `EvaluationSelection` class is called to decide which objects generated XML is appended into the root of the XML metric. This class constructor accepts the objects of `BaseFilterWidget`, `EvaluationTimeWidget`, `GroupEvaluationsSelection`, and the category selection object which is in this case, the `EventsBasedSelection`, and adds the generated XML from each to the metric root.
- Regarding the chart configurations, a stored chart sample XML file is requested over HTTP, and the chart properties data are filled in the XML.
- Finally, `formatXML` method is called from the `XMLFormattingService` to send the generated XML string to the server, and get back the well-formatted XML ready for display.

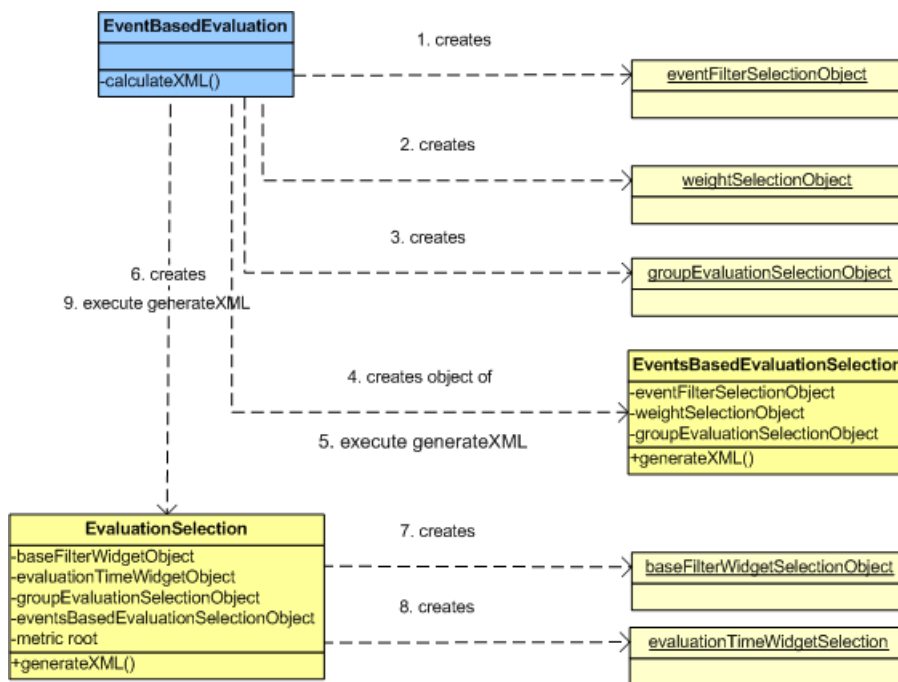


Figure 5.6: Generation of XML of interface selections

5.2.5 Variation points of other categories

Some differences are mentioned here for other categories than the previously specified events based. They share most of the sequence but with different objects sometimes based on what widgets are in each category calculation specification.

- Count Events Until based evaluation

Two `EventFilterSelection` objects are created because this category calculation needs an event to count events, and another event to stop counting instead of the weight in events based. Moreover, `EventsCountUntilBasedSelection` class handles the two events, and sent to the `EvaluationSelection`.

- Interval based Evaluation

Also two `EventFilterSelection` objects are needed, and sent to the `IntervalBasedSelection` class constructor, which is in turn sent to the `EvaluationSelection` class within the usual base filter and evaluation time widgets.

- State residence time evaluation

It creates one `EventFilterSelection`, and a `StateWidgetSelection` objects, the later is responsible for the XML for the state widget included in this category. And `StateResidenceTimeSelection` takes these two objects and after generating the XML, it is sent to the `EvaluationSelection`.

5.2.6 XML Parsers

The action of reloading a saved query to an interface is discussed in this section. It is an action performed at the saved queries category that requires parsing the query XML to map it to its correspondent UI selections in the original query category, from which it has been saved.

Parsers classes in the `xmlParsers` package handle the parsing and data retrieval from queries XML. The main parser class gets the XML blocks of each correspondent widget and some are handled by methods like base filter XML, grouping parameter, and evaluation time. Others are handled by classes for each, like group calculations, event filter, weight, and state widget. Each of these classes parse their correspondent XML and store the selections values in variables, which are retrieved later in the categories interfaces to map their selections in the interface when a reloading action is required. In the following, it is discussed how this process is implemented in the system (see also figure 5.7):

- When a reload to this query is allowed, `giveQueryDetails` method is called to get the selected query details and give them to the `QueriesXMLParser` class constructor. And gives an order to show the suitable category interface after that.

- `QueriesXMLParser` receives all the details of the query, and execute methods like `parseBaseFilter`, `parseEvaluationTime`, `parseGroupingParameter`, and `parseGranularity` by sending the parent node for each XML block responsible for these widgets XML.
- In case of a common metrics interface based query, it stores the query core (the case value calculator and group evaluations) for later comparison to compare which of the common metrics cores match it.
- `parseAdditionalSecondLevelInterfacesParameters` method is performed for the second level interfaces, and creates other parser according to the query. Events based category needs an `EventParser` and `WeightParser`, count events until based and time interval based evaluations need two `EventParser` objects to send the two XML blocks used for their events specifications, and state residence time evaluation needs an `EventParser` and `StateParser`.
- All of these parsers check the XML blocks sent to them, and store the needed values into parameters.
- In each category, and in the `onShow` method, the needed data for each are retrieved from the correspondent parsers and selections are made in the UI based on the parameters values.

5.2.7 Evaluation and Charts

Evaluation a query results in a chart visualization of the desired metric calculation. It is the main functionality of the system and all interface categories has the possibility to perform this action. The result of the first, second, and third level interfaces evaluation is a chart, but in my queries category, a layout to display multiple charts or evaluations is possible. Both chart and layout of charts evaluations are discussed in this section.

Evaluation of a Chart

When an evaluation is requested from any category including common metrics, second level, and third level interfaces, the XML of the query and chart configurations are processed in the system and result at the end in a chart image to display. The process in the system is performed as the following sequence of methods and classes (see figure 5.8):

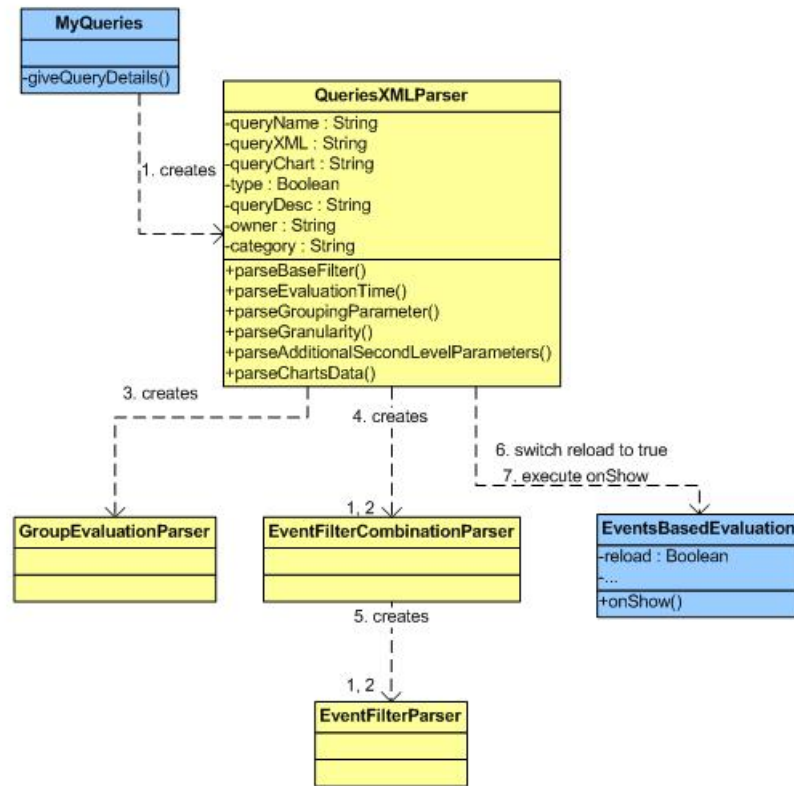


Figure 5.7: XML parsing of a saved query for reload

- `calculateXML` method is called to generate the XML of selections in the text areas for third level interface. In case of evaluation from third level interface, this method is not needed since XML is already in the widgets.
- `getChartData` method is then called, it takes the widgets that holds the XML as parameters, sending the widgets themselves as parameters is due GWT related issue since `calculateXML` and this method are both executed in an asynchronous way although `getChartData` is called after, in other words, `getChartData` does not wait till the end of `calculateXML` methods execution.
- In `getChartData` method, the metric and chart XML are sent to the server side by `getChartImage` method of the `DataServiceCharts` service.
- `getChartImage` method implementation on the server handles the XML, metric run is produced from the `calculateDOM` method `CoreFacade` class of the `BugzillaMetrics` tool, chart XML is configured and built from the `ChartFacade` of the underlying tool, and both results are added to

a diagram configuration element. This element is passed after validation to `createChart` method of `ChartFacade`, and the chart retrieved as `JFreeChart` object, which is retrieved as an image using the `ServletUtilities` Java class.

- The string resulted of the chart image is received back in the client and `displayChart` method is called if no errors occurs. This method sets the image URL based on this string and the chart is displayed.

Layout of charts

At my queries category, multiple selections of queries is possible to evaluate, which results in a layout of multiple charts images displayed. The following is the process sequence of requesting evaluation of saved queries:

- `evaluateLayout` method is called, in which XML specification and chart configurations are added to arraylists for each. Then, pass the two arrays to `getChartData` method.
- `getChartData` method send the specifications to `getChartImages` method by the same `DataServiceCharts` service.
- `getChartImages` is implemented on the server by `DataServiceChartsImpl` class. This method calls the `getChartImage` iteratively for each requested query, and an array of strings for each chart image is sent back to the client.
- After the success of the `getChartImages` service method, `displayLayout` method is called that displays the charts images for the user as the evaluation result.

5.2.8 System Authentication

System Authentication was a tricky functionality to choose its mechanism; some options were studied and discussed to perform the authentication and login of the system. It was the best choice to perform it using the Bugzilla users profiles data, to preserve the level of consistency between Bugzilla and our tool, no new usernames and profiles are needed to be created from BugzillaMetrics interface, instead using the same user data saved in the Bugzilla should suffice.

The users of the system must be logged in the used Bugzilla version in order to access the interface system. There is `BugzillaSettings` class that handles

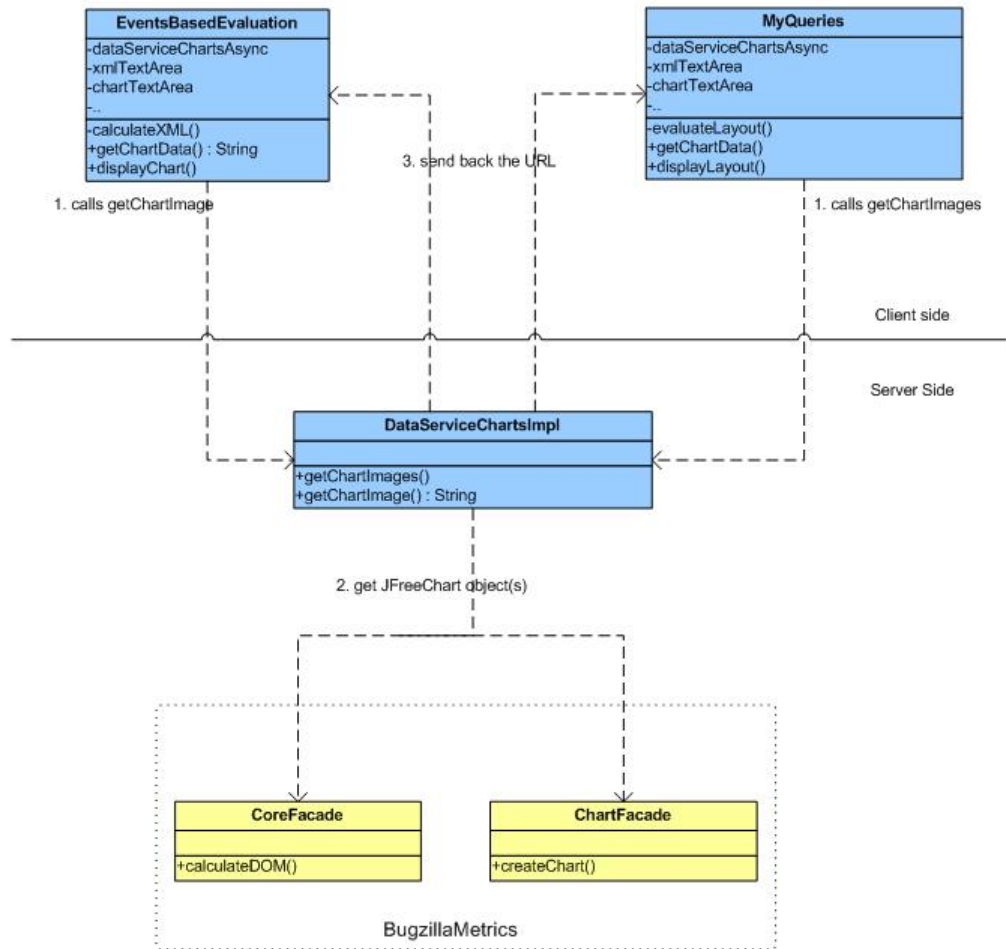


Figure 5.8: Evaluation process to generate charts

the general settings options of the authentication, such as if login in the first place is considered for the system or not, and the Bugzilla version URL.

On the initiation of the tool interface, a check for the cookies is performed, to check if the user is logged in the correspondent Bugzilla or not. If no cookies have been found, an empty page with a message asks to log into Bugzilla, and a link to the Bugzilla login page is displayed. If some data about logged in user is found in the cookies, the system performs a further authentication check based on the stored data in Bugzilla DB across the IP address of the computer, so the user must be logged in Bugzilla from the same IP address. These checks are the same as the authentication mechanism used by Bugzilla. A mandatory constraint in this mechanism is the requirement of deploying the system on the same domain

where the Bugzilla is hosted. For example, if Bugzilla is installed on `http://bugzilla.kisters.de/`, then the interface system should be on `http://bugzilla.kisters.de/someURL`. Otherwise, cookies can not be accessed due to browsers cookies access constraints.

The authentication mechanism (see figure 5.9) is implemented exactly as the Bugzilla does to authenticate its users; the mechanism within the system is performed as the following:

- `MetricsUI` entry point class checks the cookies available on the browser, mainly the `bugzilla_login` cookie that has the user id as value, and `bugzilla_logincookie` as the login cookie.
- It sends the values of the cookies to the service method `checkIfUserIsLoggedIn`, which is implemented on the server class `LoginCheckServiceImpl`. This method performs the authentication similar check to Bugzilla, which includes the following steps:
 1. Both `Bugzilla_login` and `Bugzilla_logincookie` cookies are not null.
 2. `Bugzilla_login` value is the `profiles.userid` of a row in the `profiles` table.
 3. `Bugzilla_logincookie` value matches a row in the `logincookies` table.
 4. The userids of these two rows match.
 5. `logincookies.ipaddr` matches the `CGI_REMOTE_ADDR` that is the IP address of the client.
 6. `profiles.disabledtext` is empty.
- If this method returns true for the client, the system categories are initiated and the user id is saved for session management.
- Otherwise, `getBugzillaURL` method is executed in the same service to get the URL of the used Bugzilla. A message of login failure is displayed with the URL for Bugzilla to login in.

5.2.9 Database Access

Database of the system is the same Bugzilla database, in addition to two more tables for saved queries and common metrics were needed. `BugzillaMetrics` tool also uses the Bugzilla database and has a package `sql` in its core to access many of

the cases properties data, so Bugzilla data including the base filter case properties are preferred to be accessed through the same classes of the tool for consistency reasons, where other data like login related, common metrics, and queries are retrieved directly from the interface system. The next two sections discuss both ways to database access (see figure 5.10)

Database access through BugzillaMetrics

Most of the Bugzilla database tables are accessed from the BugzillaMetrics tool such as products, components, versions, assignees, milestones, priority, severity, and status. These values are required to display in the interface, and the process is performed on the server side class `DataServiceImpl` as follows:

- Initially, a connection is established from the `ConnectionFactory` class, which is implemented using the `SQLFacade` class.
- In some properties that involve retrieving data more than just the strings, rather values and IDs; a `Select` class object is created to use its methods to add the needed parts of the SQL statement. Then, the object is passed to `executeQuery` method of `SQLFacade` class to get the result set.
- Other properties values like severity, priority, and status are retrieved as a set by `getPossibleValues` method of `EnumeratedValuesSet` class to get the defined values of such fields in the Bugzilla database.

Additional new database tables

Two additional tables are needed in the database as a system requirement, which are queries table to store the saved queries data including query name, description, metric XML specifications, chart configuration, owner, and category; and metrics table to store or add any metric desired to be displayed in the common metrics page, the metric has a name and description to display, metric XML specifications, and chart configurations.

When the common metrics category is initiated, a `CommonMetricsService` method called `getCommonMetrics` is used to get the common metrics from the database, and `getSavedQueries` of the `SavingQueryService` is used to get the saved queries when my queries category is shown.

The database access is executed directly, where `executeQuery` method is called by the `Statement` Java class object, which result in `resultSet` to get the values of the required table.

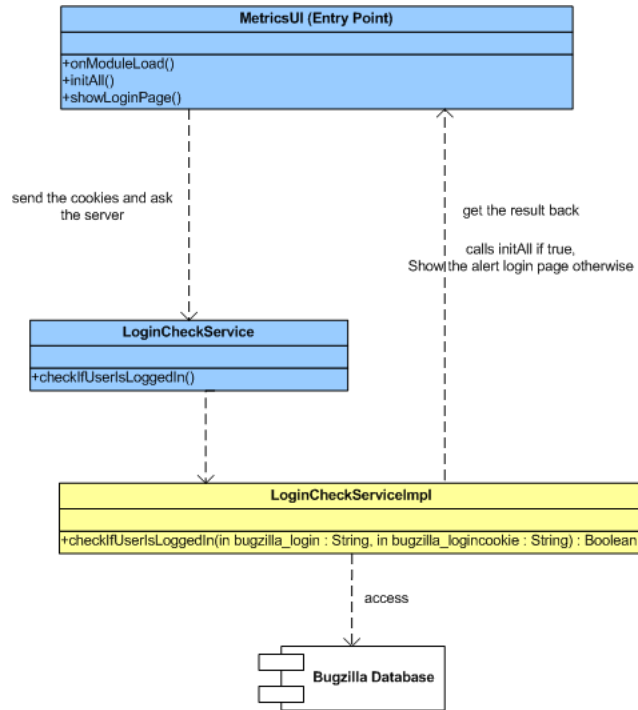


Figure 5.9: Authentication of metrics evaluation system

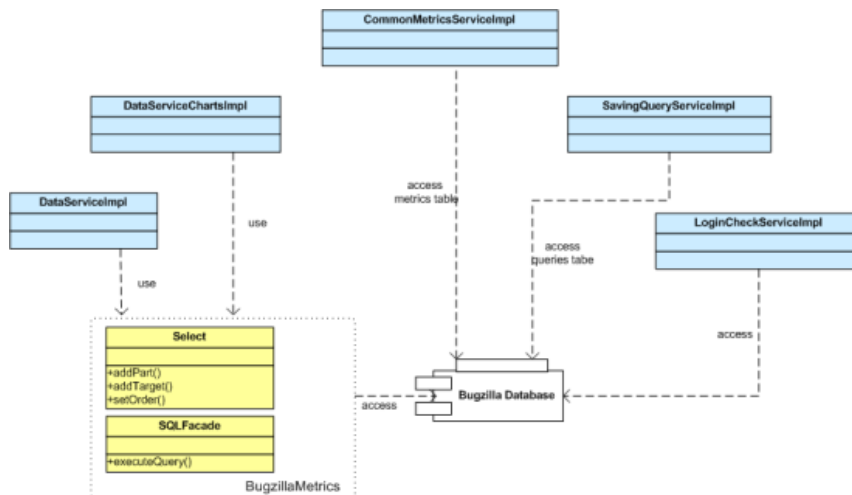


Figure 5.10: Database access mechanisms to interface data

Chapter 6

Evaluation

In this chapter, the developed system is evaluated. It starts with an overview of the system that has been developed. Then evaluation approaches and standards are discussed. Finally, the metrics evaluation interface system is evaluated in the last section.

6.1 System Overview

The metrics evaluation interface system succeeded to achieve its functionality requirements. The result is a web application with different levels of interfaces to specify, manipulate, and evaluate variant kinds of metrics. The framework is based on the XML based tool BugzillaMetrics, so some knowledge of its basic terminology is beneficial if users want to define complex metrics or XML. In general, the interface is easy and straightforward to use in most cases. The levels of specification range from direct selection of predefined common metrics, second level of detailed specifications, and the possibility to manipulate XML if needed. Here are description of the interface levels and functionality categories.

The **common metrics** page includes few visible filtering, more on demand, and a predefined metric selection. The displayed common metrics are subject to change or increase by the administrators who have database insert access. When the additional filtering and other specifications like time period and granularity are defined, the metric becomes a query and ready for evaluation or save.

Other detailed second level of metrics specifications are designed to evaluate or save more complex and detailed queries. Full base filter parameters, evaluation time period, chart display information can be defined at this level categories. They

are divided into four categories based on the kinds of queries that can be specified. In **CountEvents**, users are able to define metrics that count events on cases with different weight values like totals, totals overdue, defect rate, percentage of resolved cases without actual effort, case life time, products work distribution, incoming rate, outgoing rate, reopened rate, estimated remaining workload, user activity distribution, and correlation between severity and priority. The second detailed category is **CountUntil**, in which metrics that count events until another event occurs can be defined, like number of assignee changes before resolution and number of status changes before processing. The third is **IntervallLength**, where users can define metrics to calculate time length between two events like age of cases between before processing. Finally, **State Residence** category that can be used to define metrics for a specific case state residence time calculation until an event occurs.

For all first and second level categories, additional level of metric specification by XML is provided. Users may load interface specifications into XML before evaluation, and edit it if needed. Although the use of this level is limited to expert or familiar users with XML, which makes it less usable, but it is considered the maximum flexible way of metric specification.

Another provided functionality is saving queries. After saving a query from any of the interface categories, **saved queries** page includes the list of users queries information and other users public queries ready for evaluate or reload. Reloading can be a beneficial facility for learning since users may reload their or public queries again into the graphical interface.

Metrics evaluation results in charts, only two types of charts are supported, line and stacked. In metrics specifications categories, users may generate only one chart with customized title, range label, and type. Any number of variant charts can be evaluated and displayed at once only from saved queries evaluation, or from chart configuration third level interface using XML, by which it is possible to generate more than one chart or more than one calculation within the same chart.

For more details, screen shots from the system interface are provided in the Appendix of the thesis. You can see the usage of consistent framework, classifications of different aspects of filtering as card stacks, and some are reused in all categories of this level. Extra widgets are visible only on demand when needed, like adding a second evaluation, a second event filter, event filter transition details, and additional state filtering within an event filter.

6.2 Evaluation Approach

Evaluation is strongly related with monitoring. According to [CES02], "monitoring is about systematically collecting information that will help you answer questions about your project". This information can be used for reporting on projects and helpful for evaluation. By the same source, evaluation is defined as "using monitoring and other information you collect to make judgments about your project. It is also about using the information to make changes and improvements". It usually includes analysis of actual progress and comparisons with prior plans.

The final judgments can give an idea of the significance, worth, and achievements of the system, represented by two main functions [RLF04]: Quantitative evaluation (How well did we do?), and qualitative evaluation (How much did we do?). Evaluation should not only demonstrate achievements, but also shows how it was done, what was the most effective, strengths, and weaknesses of systems. At end of evaluation, it should also be clear if the system should be continued and how can it be improved.

There are two main kinds of evaluations [Kir98], formative that is performed while program is being developed, collecting continuous feedback from stakeholders in order to revise the system as needed. The second kind is summative evaluation that focus on the outcome and judge at the end of system activities.

Although the first formative evaluation was implicitly done while developing the system, as most of requirements were not clear or specifically identified, But it is irrelevant here since the system is already developed. Therefore, this chapter's evaluation concentrates on the outcome, measuring some aspects (quantitative and qualitative) and judging them.

Next section uses the ISO 9126 [Bal98] for software products evaluation standards, which include six major characteristics to describe software quality for the developed metrics evaluation system.

6.3 Software Product Evaluation

This section applies the official ISO 9126 Evaluation standards to the developed interface system. It covers most of the system's quality aspects to give an understanding of what and how well did we do.

6.3.1 Functionality

Suitability. The interface system covers all the functional requirements, which were discussed previously in the requirements specifications. It covers all metrics that can be specified in BugzillaMetrics within its different levels of interface, and other functionalities like saving and manipulating metrics queries. The interface conforms to different levels of specifications as well as users. Analysis of supported metrics and several incremental improvements were success keys for the achievement of a satisfying interface for all stakeholders.

Accuracy. The interface specifications of queries are mapped into the correspondent XML, so they result in correct and precise metrics results since XML is processed by the tested BugzillaMetrics. The interface system was tested with real data and resulted in correct and expected behavior.

Interoperability. The system is a web application that works across any browser, and based on Bugzilla data and BugzillaMetrics. It is merged with the functionalities provided by BugzillaMetrics, and works with Bugzilla database from version 2.19.3 up to 3.0.2, in addition to two needed tables for queries and common metrics. Therefore, the system is easy to use by any tool or organization as long as such data is available and compatible.

Compliance. The interface was developed based on the framework of BugzillaMetrics, which is the only restriction our interface considered. BugzillaMetrics was a standalone and initiative idea with no official compliance.

Security. The system requires users to login on the correspondent used Bugzilla. After authentication, users can perform full functionalities of the system. Users groups are not supported, only user based accessibility rules, which means private queries are only for their user and public queries are visible to all of the organization users. The system is used in two directions: It was originally installed for Kisters AG, where employees can use the system internally after their login at Kisters Bugzilla. The second is the open source version, by which, any organization may use it based on its Bugzilla and switch on/off the login requirement if needed.

6.3.2 Reliability

Maturity. The system results with a low frequency of failures, especially in the major issues and functionalities. All aspects were achieved properly due to the continuous testing during the incremental implementation. However, small issues still need enhancements (which are not failures) like metric description display with the generated chart, and more flexible user sort of the saved queries list.

Fault tolerance and recoverability. Since the system does not result in mentioned failures, this property is not a problem. In case of other uncontrolled failures like network connections, no data will be lost since the system provides straightforward evaluations, and saving queries perform a direct save into the database, and then, will be always available. Moreover, no harmful affect of any kind is probable on the bugs data since only read access is performed on Bugzilla database tables.

6.3.3 Usability

Understandability. As the case with any other change request analysis tool, the system has its own concepts and terminology. Therefore, the users should be familiar with the framework behind this tool. The purpose was to come up with a new flexible tool to meet essential metrics requirements. As soon as users understands the terminology behind the system metrics specification aspects like events on change requests lifetime, they will find the system easy and straightforward to use.

Learnability. This is related to the last point of understanding. After a basic understanding of the framework, the system functions in a way allow users to learn in some aspects, like the XML. Each category of interface query specifications is combined with another XML page, and users can perform only load XML instead of only evaluating. Users can then change specifications and load its XML for observation if they are interested to learn the XML level of specifications. Moreover, since the worst case of any specification is a result of an empty chart, i.e. no harmful errors, users can be comfortable and free to try, and trying brings users to learn.

Operability. The different levels of the system's interface help to ease users' usage and operating on it. The common metrics can be easily performed, public saved queries by other users are also ready metrics to use. Other detailed specifications are divided into stacks of classifications, with necessary visible widgets and extras on demand using AJAX based technology of fast responsive time (see section 5.1).

Several design guidelines (section 3.1.3) were considered to improve the usability of the system. The design of different categories was consistent in both concepts and visual outlook, good defaults allow frequent users to perform their actions by simple and short specifications, minimized error possible occurrences and simple error handlers, and no long sequences of actions to perform tasks in order to reduce users' short-term memory needs.

Metric	Totals	Mean	Maximum
Totals Lines of Code	8807		
Number of Packages	12		
Number of Interfaces (avg/max per package)	14	1.5	12
Number of Classes (avg/max per package)	61	8.3	13
Number of Methods (avg/max per class)	342	5.6	24
Method Lines of Code (avg/max per method)	6572	18.2	269

Table 6.1: Metrics for BugzillaMetricsFrontEnd project

6.3.4 Efficiency

Time behavior. The time of response in the different aspects of the system was optimized. To have a realistic real data testing, Eclipse Platform Bugzilla database was used [August 2007], which contains 200.000 bugs, 82 products, 700 components, and more than 1000 possible assignees. The system initiates and reloads in a time less than five seconds. Lazy categories widgets load was implemented, where parts of each category are loaded on the first click on each, then categories are shown within one or two seconds at first click. Displaying categories after the first click won't take more than 0.2 milliseconds. Loading XML is implemented on the browser side, which takes around half a second to generate complete detailed query XML. Evaluating mainly depends on BugzillaMetrics processing of XML to retrieve charts data, which may take ten seconds maximum using real data and good working server. The Firebug add-on of Firefox was used to retrieve the previous actions processing time.

Resource behavior. The resource behavior of the system was tested, and no problems occurred in any case regarding recourses usage.

6.3.5 Maintainability

Analyzability. The system is able to show in most cases what the cause of the failure is when it exists. Any error or exception occurs is handled by specific class to add the exception description into the error message. Moreover, the usage of GWT enables detailed and pretty debugging within IDE.

Changeability. The effort needed to implement a change in the software is not very hard in most of the aspects. The implementation is structured into packages for each concern (see sections 5.2.2 and 5.2.3). For example, reusable widgets are implemented by separate classes in `gui.widgets` package. If the change is in the user interface, the developer needs to go to the category or widget constructor, which is not hard. Some changes need to be also reflected when loading XML

if they affect the potential resulting XML, which should be taken care of in the XML parsers and XML builders packages. Again, the place of change can be detected based on the changed widget of category since each widget class has its correspondent XML builder class. Table 6.1 shows some automatically calculated metrics about the system. Note that method lines of code average is higher than the well-organized JAVA programs. This was caused by the unavoidable complexity in constructing customized widgets that contain large number of other widgets like list boxes and their properties. However, related enhancements in few classes are recommended in the near future.

Stability. Although few additional improvements on the system are still beneficial, the overall system is stable in general. Any changes must be consistent with BugzillaMetrics and the relying database. The database tables that are accessed by BugzillaMetrics are not accessed directly from the system interface, instead `sql` BugzillaMetrics package is used in order to be consistent (see section 5.2.9), other table not accessed through BugzillaMetrics tool are queries and metrics tables since they have nothing to do with that tool.

Testability. For testing any new changes of the interface system, it has to be run as GWT project within the used IDE. Therefore, familiarity with GWT projects structure and compilation is needed.

6.3.6 Portability

Adaptability. The system interface development was the final stage to achieve the needed tool. However, it can be an operating environment for potential future improvements. For example, quality and descriptiveness of charts. In another direction, the system adapts itself to the underlying BugzillaMetrics and database information like different bugs properties. It retrieves such bugs information (e.g. priorities, states) from Bugzilla database through and consistently with the BugzillaMetrics.

Installability. The system is deployed as a web application. Therefore, users can simply access a webpage to use the system. For administrators in the server side, the installation requires configuring the database and creating the custom tables, which can be done using an `sql` file provided within the source code. Then, specifying some properties settings from BugzillaMetrics properties file like Bugzilla database connection setup, Bugzilla version, and URL. After that, deployment can be done using a provided file release or from source code. Details are available at BugzillaMetrics tool home page [BUM].

Conformance. The web application system is implemented using JAVA programming language based on GWT technology [GWT], which runs completely as JavaScript

on browser side and uses J2SE 1.4.2.

Replaceability. The system is standalone application, which plays as a complement and based on BugzillaMetrics. Any other tool that uses the same settings may link to and use it.

Chapter 7

Summary and Future Extensions

The goal of the thesis was to develop an interface system for a change request analysis tool. The tool BugzillaMetrics was developed to overcome the shortcomings of other tools. It uses XML configuration files for metric specifications. To make this tool usable, an understandable and user friendly user query and reporting interface should be developed.

For the project development, meetings of requirements discussions played an essential part in order to end up with a satisfying interface for stakeholder. In parallel, supported metrics by the tool were studied and analyzed in detail, and evaluation of existing tools features and interfaces design was done. The result of these activities was an initial requirements specification, which further included first prototype as activity diagrams and clarifying screenshots of the system.

The implementation was developed in an iterative way, parallel with several meetings and requirements enhancements. Common metrics was the first developed version, followed by other detailed more complex interfaces and other functionalities like saving and reloading.

The system achieved the requirements functionalities, and succeeded to provide an efficient and understanding interface. Different queries can be defined within only interface specifications, and several task and metric type oriented categories are provided. The first level (common metrics) can be added by database administrator, and users can directly evaluate them. Second level detailed categories were designed in consistent and stacked specifications with extra details on demand. The used technology GWT played an important role in the design to develop AJAX application, which is known in its fast response time and efficiency.

Several issues were considered to improve learnability of the system. XML specifications of defined queries can always be generated with an unmentioned time,

and edited or evaluated. Queries can be saved as private or public, and users can evaluate other users public queries or even reload them into their correspondent interface selections. Moreover, the interface design provides implementation of many design patterns and guidelines like the consistent visual framework, clear task oriented categories, usage of card stacks in the detailed interfaces to reduce complexity, extras on demand, good defaults, and simple error handlers.

There are several possibilities of future extensions and improvements to the developed system. The following is a list of such possible extensions:

- Extension of the BugzillaMetrics core to use data from version control systems like CVS or Subversion. This would allow evaluating more beneficial metrics from the CVS like number of changed lines for certain cases.
- Charts can be improved to include more information like metric description, provide links to the calculated cases or values display when the mouse over its lines, and additional charts types like pie chart.
- Presentation of metric calculations with different start points of the evaluation time periods within simple chart. This may result in beneficial charts like comparison of different releases.
- Automatic publications of specific live metrics results can be beneficial for fast monitoring at Kisters AG, which can be shown without users' effort.

To sum up, a flexible interface system for metrics evaluation was developed, which was put into production use at Kisters AG, and satisfying results were accomplished. Additionally, the project has been published as open source, which offers the possibility to contribute to the real work scenarios in this field.

Appendix A

Interface Screenshots

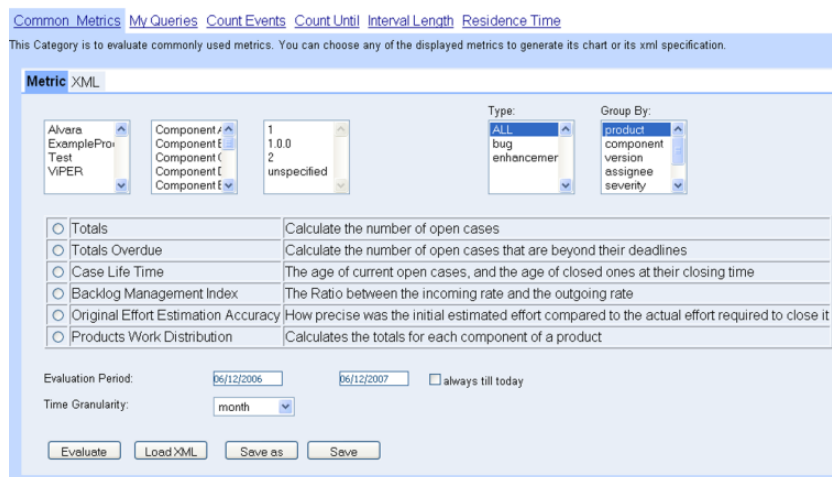


Figure A.1: Common Metrics Category.

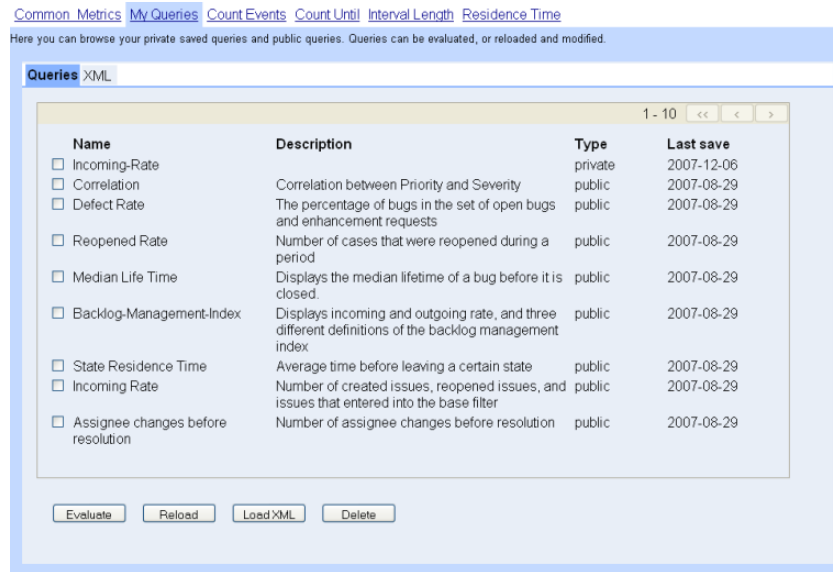


Figure A.2: MyQueries Category for saved queries manipulation.

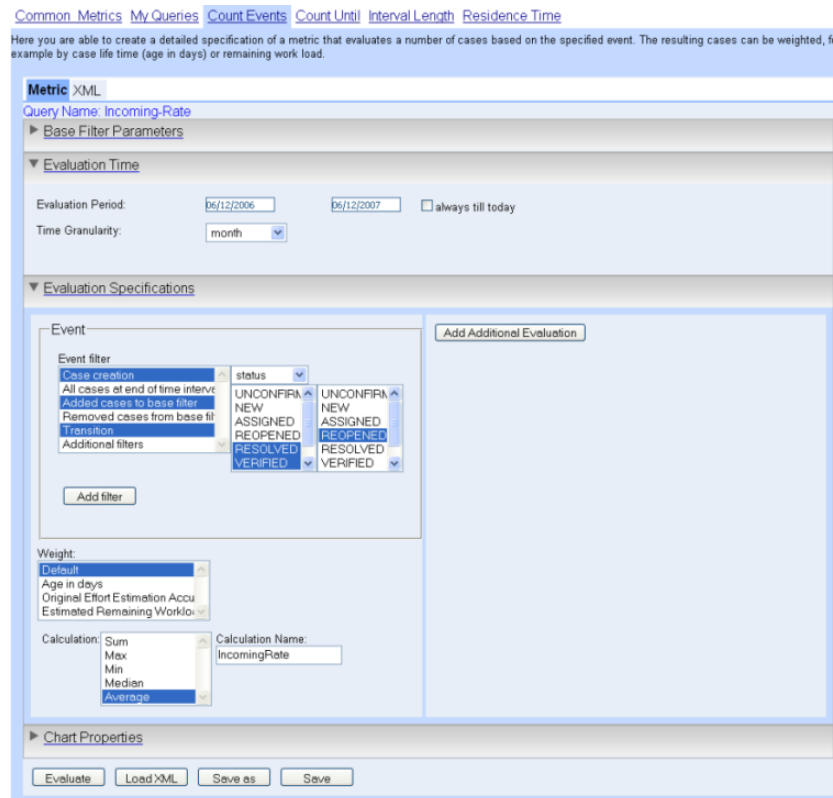


Figure A.3: CountEvents Detailed Category.

[Common Metrics](#)
[My Queries](#)
[Count Events](#)
[Count Until](#)
[Interval Length](#)
[Residence Time](#)

Here you are able to create a detailed specification of a metric that evaluates a number of cases based on the specified event. The resulting cases can be weighted, for example by case life time (age in days) or remaining work load.

Metric **XML**

Query Name: Totals
Query Specification:

```

<?xml version="1.0" encoding="UTF-8"?>
<metric>
  <baseFilter>
    <or>
      <value field="product">3</value>
      <value field="product">2</value>
    </or>
  </baseFilter>
  <groupingParameters>
    <fieldGrouping>product</fieldGrouping>
  </groupingParameters>
  <groupEvaluations>
    <calculation name="totals">
      <divide>
        <sum caseValueCalculator="totals" />
        <count caseValueCalculator="totals" />
      </divide>
    </calculation>
  </groupEvaluations>
  <caseValueCalculators>
    <countEvents id="totals">
      <event>
        <endTimeInterval />
      </event>
      <weight>
        <default />
      </weight>
    </countEvents>
  </caseValueCalculators>
  <evaluationTimePeriod>
    <timePeriod>
      <start>2006-12-06</start>
      <end>2007-12-06</end>
    </timePeriod>
  </evaluationTimePeriod>
  <timePeriodGranularity>
    <month />
  </timePeriodGranularity>
  <fixedFields />
</metric>

```

Chart Configuration:

```

<chartConfiguration>
  <title>Total Number of Cases</title>
  <chart>
    <calculation>totals</calculation>
    <rangeAxisLabel>cases</rangeAxisLabel>
    <type>line</type>
  </chart>
  <width>1024</width>
  <height>768</height>
</chartConfiguration>

```

Figure A.4: XML specifications level, combined with each category.

Bibliography

- [AC] Inc Apple Computer. Apple human interface guidelines. <http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/>. [Online, accessed November 20, 2007].
- [Bal98] Helmut Balzert. *Lehrbuch der Softwaretechnik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Spektrum, Akademischer Verlag, 1998.
- [BUGa] Bugzilla. <http://www.bugzilla.org/>. [Online, accessed November 20, 2007].
- [BUGb] Evaluated Bugzilla Version. <http://landfill.bugzilla.org/bugzilla-3.0-branch/>. [Online, accessed November 20, 2007].
- [BUGc] The Bugzilla Guide. <http://www.bugzilla.org/docs/>. [Online, accessed November 20, 2007].
- [BUM] BugzillaMetrics. <http://www.bugzillametrics.org/>. Online.
- [CES02] Charities Evaluation Service. *First Steps in Monitoring and Evaluation*. Charities evaluation service, 2002. <http://www.ces-vol.org.uk/>.
- [CODa] Code Beamer. <http://www.intland.com/products/codebeamer.html>. [Online, accessed November 20, 2007].
- [CODb] Evaluated Code Beamer Version. <https://codebeamer.com/cb/user>. [Online, accessed November 20, 2007].
- [FP98] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 1998.

- [GGSa] Evaluated JIRA Version. <http://www.gridgainsystems.com:8080/jira/>. [Online, accessed November 20, 2007].
- [GGSb] GridGain Systems. <http://www.gridgain.com/>. [Online, accessed November 20, 2007].
- [GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.
- [Gra07] Lars Grammel. Development of a tool for evaluation of change requests. Master's thesis, RWTH Aachen, Research Group Software Construction, 2007.
- [GWT] Google Web Toolkit GWT. <http://code.google.com/webtoolkit/>. [Online, accessed November 20, 2007].
- [INT] Intland GmbH. <http://www.intland.com/>. [Online, accessed November 20, 2007].
- [JIR] JIRA Atlassian Software Systems Pty Ltd. <http://www.atlassian.com/software/jira/>. [Online, accessed November 20, 2007].
- [Kir98] D.L. Kirkpatrick. *Evaluating Training Programs: The Four Levels*. Berrett-Koehler Publishers, 1998.
- [May91] D.J. Mayhew. *Principles and guidelines in software user interface design*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1991.
- [MOZ] The Mozilla Organization. <http://www.mozilla.org/>. [Online, accessed November 20, 2007].
- [POLa] Evaluated Polarion Version. <http://community.polarion.org/polarion/index.jsp>. [Online, accessed November 20, 2007].
- [POLb] Polarion for Subvision. <http://www.polarion.com/index.php>. [Online, accessed November 20, 2007].
- [PRS02] Jenny Preece, Yvonne Rogers, and Helen Sharp. *Interaction Design*. John Wiley & Sons, Inc., New York, NY, USA, 2002.
- [RLF04] P.H. Rossi, M.W. Lipsey, and H.E. Freeman. *Evaluation: A Systematic Approach*. Sage Publications, 2004.
- [Shn97] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

- [Som04] Ian Sommerville. *Software Engineering 7*. Addison Wesley, 2004.
- [Tid05] Jenifer Tidwell. *Designing Interfaces*. O'Reilly Media, Inc., 2005.
- [Tra03] David Travis. *Bluffers' Guide to ISO 9241 Usability Standards*. User-focus ltd, United Kingdom, 2003.