Integrating an object-oriented metrics tool into SNiFF+

Class Leverentz, Frank Simon
Software and Systems Engineering Group
BTU Cottbus

In the project Crocodile tools for supporting software quality measurement are developed. In particular, object oriented metrics are used to define product quality models for object oriented programs and frameworks. Our current tool implementation is based on and integrated into the SNiFF+ programming environment.

Crocodile’s approach is to use the parsed data from SNiFF+'s symbol table for extracting a basic set of structural program properties. This symbol table always contains the up-to-date data like classes, inclusions and references of the currently open software project. Because SNiFF+ does the mapping from the programming language into the symbol table, our tool itself is language independent. The whole measuring process is done within a SNiFF+ session. It’s started by a new SNiFF+ command, it’s configured through it and the feedback of the measured values is given back to SNiFF+'s user interface. Crocodile offers its own query-language for defining specific measures and quality models. A special mechanism to filter the amount of calculated measurement values to the critical ones helps the engineer to effectively do and understand the measuring. This data reduction is achieved by defining subsets of classes to be measured - considering usage and inheritance context - and by defining critical ranges for the self-defined object oriented measures.

Thus, Crocodile provides an easy way to create a specialized measurement process considering the user’s specific goals in engineering or re-engineering. Furthermore, measurement activities are well integrated into the particular software development model and tool support through the usage of SNiFF+.

Content:

1 Objectives and Principles for an Integrated Metrics Tool .................................................2

2 Measuring with Crocodile ..............................................................................................3

2.1 Defining measures ......................................................................................................4

2.2 Interpretation of measurement values .........................................................................5

2.3 Defining the detailed quality model ............................................................................5

3 Architecture and Implementation of Crocodile ..........................................................6

4 How to use Crocodile ...................................................................................................6

4.1 Preparation of a Quality Model ................................................................................11

4.2 Extraction of Data from the Symbol Table ...............................................................11

4.3 Selection of the measurement context ......................................................................12

4.4 Measurement and Interpretation of the results ........................................................13

5 Experiences and Future Work ......................................................................................13

5.1 Experiences with SNiFF+ as Integration Platform ....................................................13

5.2 Experiences with Using Crocodile .............................................................................14

5.3 Figure Work ...............................................................................................................15

6 References .....................................................................................................................16

Object-oriented construction techniques have the advantage of providing a uniform conceptual framework throughout design and coding and, thus, allow to combine design and code metrics in uniform quality models. Almost all of these metrics are independent from the actual object-oriented implementation language, but rely on general principles and rules for structuring programs with classes and how such class structures should look like.

Thus, in an object-oriented approach product metrics can be used as early as the first design exists and can be monitored throughout the evolution process of the program.

The hierarchical quality model itself depends much on the target environment of the software product and the design and coding standards to be followed by the developers. The definition of adequate quality criteria, called „quality aim determination“ ([Balz98], p. 269 ff.), is an important part of the process definition and should be done during requirements analysis phase. Particularly, the development and evolution of long living and reusable software products like object-oriented frameworks and component libraries require strong quality criteria (cf. [Erni96]). These additional layers in the quality hierarchy allow for better explanation of the measured values and, potentially, to describe potential corrective actions to be taken by the software designer. An example would be the principle of good decoupling of classes, which can be measured by a combination of several simple metrics, and in turn is a rule how to achieve well-structured future software implementations.

We advocate the use of design and code metrics as a feedback instrument for software engineers during development and evolution of software systems. The measurement process should support product review and inspection steps by providing a quick and objective analysis of structural properties and relating them to explicitly defined quality goals. This requires both, that product quality metrics are available for all types of design and code documents being produced during the construction process, and that appropriate measurement tools are accessible as an integral part of the software development environment.

The necessary data for measuring object-oriented designs can be represented as an entity relationship diagram as shown in figure 2.

1. Objectives and Principles for an Integrated Metrics Tool

As in other engineering disciplines, it is a major objective in software development to control the product and process quality. Software product quality is expressed both by internal structural/property aspects as well as by external quality aspects like maintainability and portability as well as by external quality aspects like maintainability and portability as well as for software product consisting of attributes like usability and correctness ([SiPa94], p. 49).

Because many of these attributes cannot be measured directly they are often subdivided into refined factors and criteria which in turn are related to one or more simple product metrics (cf. [Fen95], p. 42). This hierarchical approach has been implemented as a standard way to define and to measure software quality ([ISO9126]). Figure 1 shows a sub-tree of such a quality model.

The hierarchical quality model itself depends much on the target environment of the software product and the design and coding standards to be followed by the developers. The definition of adequate quality criteria, called „quality aim determination“ ([Balz98], p. 269 ff.), is an important part of the process definition and should be done during requirements analysis phase. Particularly, the development and evolution of long living and reusable software products like object-oriented frameworks and component libraries require strong quality criteria (cf. [Erni96]). These additional layers in the quality hierarchy allow for better explanation of the measured values and, potentially, to describe potential corrective actions to be taken by the software designer. An example would be the principle of good decoupling of classes, which can be measured by a combination of several simple metrics, and in turn is a rule how to achieve well-structured future software implementations.

We advocate the use of design and code metrics as a feedback instrument for software engineers during development and evolution of software systems. The measurement process should support product review and inspection steps by providing a quick and objective analysis of structural properties and relating them to explicitly defined quality goals. This requires both, that product quality metrics are available for all types of design and code documents being produced during the construction process, and that appropriate measurement tools are accessible as an integral part of the software development environment.

The necessary data for measuring object-oriented designs can be represented as an entity relationship diagram as shown in figure 2.

2 Measuring with Crocodile

According to the goal to use measurement-based analysis as early as possible in the development process, we concentrate on structural data available on architecture level. The object-oriented system model at least consists of:

• classes and the inheritance and association relations between them,
• methods and attributes with their visibility and their usage (which attributes are accessed and which methods are called from other methods).

The process of data collection for the measurement process has to be as simple as possible and not to put additional overhead to the CASE environment and the software engineer.

The metrics tool should be smoothly integrated into a CASE environment.

The underlying quality model as well as the metrics definitions have to be highly flexible and adaptable.

The metrics tool should give an immediate, understandable feedback to the software engineer against the loss of structure often encountered due to functional extensions and adaptations of software to new requirements.

These observations guided us to develop an adequate metrics tool for supporting measurement-based product quality management as an integral part of the software engineering process. The central requirements are:

• The process of data collection for the measurement process has to be as simple as possible and not to put additional overhead to the CASE environment and the software engineer.
• The metrics tool should be smoothly integrated into a CASE environment.
• The underlying quality model as well as the metrics definitions have to be highly flexible and adaptable.
• The metrics tool should give an immediate, understandable feedback to the software engineer even for more complex measures. It should easily allow for the tracking of problem spots in the program design or code according to a given quality model.

The process of data collection for the measurement process has to be as simple as possible and not to put additional overhead to the CASE environment and the software engineer.

The metrics tool should be smoothly integrated into a CASE environment.

The underlying quality model as well as the metrics definitions have to be highly flexible and adaptable.

The metrics tool should give an immediate, understandable feedback to the software engineer even for more complex measures. It should easily allow for the tracking of problem spots in the program design or code according to a given quality model.

2.1 Defining measures

Crocodile provides three kinds of basic measures: class related, (e.g. number of methods), method related (e.g. length of method), and attribute related measures (e.g. number of references to an attribute).

The definition of a new measure generally comprises:

• an iterator over a set of entities of a particular type (class, method, attribute),
• the name of the new measure, and
• an expression using SQL-like queries and calculations.

The following example shows the definition of a new class measure named number_of_all_classes.

For all classes C (number_of_methods x count methods where methods.class_id = c.class_id)

For each class C number_of_methods is calculated by counting all methods belonging to this class. In such cases the class_id of the method, expressing the association between classes and methods in the ER-schema, is equal to the class_id of C.

Our query definition language supports:

• any joins including aliases,
• regular expression matching,
• columns to column comparisons in WHERE clauses,
• complex conditions, and
• nested SELECT-statements.
Besides simple selection and joining of basic data, arithmetic operators are used to scale and to combine simple measures into more complex measures (corresponding to indirect measure in [Frei95]). For example after defining the queries for number_of_methods and number_of_attributes it is possible to define a new measure weighted_class_complexity with the following definition:

\[
\text{for all classes} \\
\quad \text{(weighted_class_complexity): } (3 \times \text{number_of_methods}) + (2 \times \text{number_of_attributes})
\]

### 2.2 Interpretation of measurement values

An essential part of any quantitative quality model is the definition of ranges for metrics values, which allow for the interpretation of measurement results as either good or critical with respect to a particular quality (sub-)goal (cf. figure 3). The definition of such value partitions or, more generally, quality levels (cf. [ISO9126]), provide a means to filter the huge amount of measurement values to those indicating critical situations. The software engineer should review such situations carefully.

In Crocodile we support the following ways to define critical values within the measurement context:

- values are critical if they are inside an absolute interval, e.g. number_of_all_methods [3, 5]
- values are critical if they are outside an absolute interval (as shown in fig. 3), e.g. number_of_public_attributes < 30, 4
- values are critical if they belong to the group with the x highest respectively lowest values, e.g. length_method (max, x)
- values are critical if they belong to the group with the y percent highest respectively lowest values, e.g. weighted_class_complexity (min, y%)

![Figure 3: quality levels](image)

The metrics together with such critical value ranges form the leaf level of the above described hierarchical quality model as for instance shown in fig. 1.

### 2.3 Defining the detailed quality model

For interpreting the calculated measurement results they have to be connected to some quality criteria. A critical value indicates that the corresponding criterion is not fulfilled. To be as flexible as possible Crocodile does not come with fixed, built-in quality models. So the full model has to be defined. Starting from the root which could be a general quality goal like reusability descriptions of directed paths from this class are defined. Starting from the root which could be a general quality goal like reusability descriptions of directed paths from this class are defined. To handle such situations and to allow the user for an explicit definition of what to include into the measurement Crocodile offers three mechanisms (cf. [Erni96]):

- Crocodile provides an easy way of selecting a class-context, which contains all classes to be measured. To measure particular parts of a composed program only those classes are included in the focus.
- Crocodile offers a solution to the distributed functionality of a class over an inheritance tree by providing the possibility to select an inheritance-context. The functionality of classes - containing methods and attributes - from the inheriting context is copied into subclasses of the focus. Thus, the class is changed to its flat representation and the measures are considering the full set of properties of this class.
- Crocori offers the possibility of selecting a use-context. When measuring a focus there are only considered references of used attributes and methods from classes within the focus. By using a use-context there exists the possibility to measure the focus containing connections to outside of the focus.

### 3 Architecture and Implementation of Crocodile

The Crocodile metrics tool is designed to be fully integrated into existing software development environments (SDE) that supply object-oriented design and coding tools (structure editors, parsers, source code browsers) and version management. The main components of the Crocodile tool are abstract interfaces to the SDE services and an SQL-database system, the quality model description, a query language interpreter, and additional user interface components to select the measurement context and to present the analysis results.

These components and the overall tool architecture are shown in figure 4 and are briefly described in the following:

- **Figure 4: Architecture of Crocodile**

  - The CASE interface allows to activate language parsers and to access the symbol table information built-up through the parsing process. Other required services are the usage of source code and structure browsers to feed back analysis results into the program representation maintained by the CASE tools. The CASE interface is implemented by specific wrapper classes to existing CASEs. Our current platform is SNIFF+ 2.3.1 from TakeFive. The integration is explained in more detail below.
  - The SQL database stores the basic structural data according to the schema described in section 2. This data is extracted from the CASE's symbol table. Additional data for complex measures is created and stored during the analysis process. The interface provides full SQL query access to all of this data. As above this abstract interface is implemented via specific wrapper classes on top of commercially available DBMS. Our current implementation uses mSQL 2.0 from Hughes Technologies (cf. [Rob95]).
  - The Context Selector allows interactive definition of the measurement context in terms of focus, inheritance context and use context selection as discussed in section 2.4. Based on

- **Figure 5: Architecture of Crocodile**

  - number_of_public_methods to the criterion completeness with a higher threshold value than in a connection to the criterion descriptiveness. Therefore, the definition of a sub-tree for the described criteria could look like:

    \[
    \text{quality \& portability \& completeness: number_of_public_methods > 15,50} \]
    \[
    \text{quality \& reusability \& descriptiveness: number_of_public_methods > 3,30} \]

  This quality model is used by Crocodile to provide an interpreting of the measurement results. The in [Erni96] described additional layers between criteria and metrics that consist of design rules (cf. chapter 1) can be realized by defining corresponding rules, e.g. descriptiveness keeps_public_interfaces narrow to number_of_public_methods > 3,30.

- **2.4 Class context for basic measures**

  When measuring object-oriented software three problems can occur:

  - In most cases object-oriented software builds on top of basic libraries, providing the access to the operating system, GUI support, or data structures and algorithms. When measuring such composed programs the software engineer might only be interest in analysing the self-written parts, because only those are under his control.
  - The functionality of a class - containing methods and attributes - might be distributed over all classes it inherits from. Therefore, the result values for some structural measures might not reflect the situation within this class appropriately.
  - Another kind of distributing functionality over several classes can be reached by association including aggregation. When measuring one class without considering its usage of other classes, e.g. graphical library, the result values for some structural measures could lead to wrong conclusions.

  To handle such situations and to allow the user for an explicit definition of what to include into the measurement Crocodile offers three mechanisms (cf. [Erni96]):

  - Crocodile provides an easy way of selecting a class-context, which contains all classes to be measured. To measure particular parts of a composed program only those classes are included in the focus.
  - Crocodile offers a solution to the distributed functionality of a class over an inheritance tree by providing the possibility to select an inheritance-context. The functionality of classes - containing methods and attributes - from the inheriting context is copied into subclasses of the focus. Thus, the class is changed to its flat representation and the measures are considering the full set of properties of this class.
  - Crocodile offers the possibility of selecting a use-context. When measuring a focus there are only considered references of used attributes and methods from classes within the focus. By selecting a use-context there exists the possibility to measure the focus containing connections to outside of the focus.
4 How to use Crocodile: A sample session

Referring to figure 6 the measurement process within SNiFF+ with the Crocodile measurement add-on can be divided into four phases:

1) Preparation of a quality model configuration file including
   - definitions of basic-measures and complex measures,
   - definition of a detailed quality hierarchy including the critical value ranges for the measures.
2) Extraction of the data from the SNiFF+ symbol table into the Crocodile database for further calculation.
3) Selection of a measurement context consisting of focus, inheritance context and use context.
4) Measurement and interpretation of the calculated and visualised values.

All steps except for the first-one are executed from within a SNiFF+-Session. In the following subsections screenshots of a sample session are used together with some brief comments to sketch the typical usage of our tool.

4.1 Preparation of a Quality Model

The functionality of SNiFF+ is grouped into the function-clusters „Project Management“, „Code Comprehension & Reverse Engineering“, „Development“, and „Build Management & Tool Integration Framework.“

All tools are working on a central data repository, basically formed by the symbol table of the SNiFF+ parsers. It contains all data about the structure of a project like classes, their attributes, methods, use-references etc. SNiFF+ provides good customising and tool integration facilities. By only relying on the symbol table and SNiFF+‘s tool access interface for using built-in browsers and source code editors, Crocodile has got the following advantages:

- it is language independent because it doesn’t have to deal with parsing, but simply re-uses the available parse results.
- Crocodile hasn’t to care about special macro expansions or a particular compiler.
- analysis of even incomplete code is possible due to SNiFF+ „on the fly“ parsing technology (Take96).

After modifying the .UserMenus.sniff file in the home directory the menu bar of the SNiFF+-source editor looks like figure 7. All measuring activities with Crocodile are controlled from within the current SNiFF+-session. The menu command Fill Database exports the current content of the symbol table to Crocodile’s database. The symbol table automatically contains the data of the current open project. When working with a subproject, only that data is exported. The project sources may be incomplete, syntactically inaccurate and having un compilable code because the symbol table only stores structural design data. Besides starting this data export no further user interaction is required for this step.

4.2 Extraction of Data from the Symbol Table

After modifying the .UserMenus.sniff file in the home directory the menu bar of the SNiFF+-source editor looks like figure 7. All measuring activities with Crocodile are controlled from within the current SNiFF+-session. The menu command Fill Database exports the current content of the symbol table to Crocodile’s database. The symbol table automatically contains the data of the current open project. When working with a subproject, only that data is exported. The project sources may be incomplete, syntactically inaccurate and having un compilable code because the symbol table only stores structural design data. Besides starting this data export no further user interaction is required for this step.

4.3 Selection of the measurement context

After storing the structural data in the database the measurement context has to be set. This task is started with a corresponding command from the source editor, too. The resulting window of a specialized multiple selection context browser is shown in figure 8. It has a SNiFF+ menu with display functions that allow to connect to the current open SNiFF+-session. Selected classes can be easily viewed by opening the SNiFF+ hierarchy browser or source editor.

With the file menu commands a context selection from a previous Crocodile session can be loaded and also be saved for future work (for example to do some trend analysis with the same measurement context in different source code versions). After completing the definition of the measurement context and storing it in the database the window can be closed.
A second point is to demonstrate the effectiveness of the integrated measurement approach in The key issue of our future work are the theoretical and empirical validation of existing improve the quality of the produced software and, therefore, it seems to be a useful tool for the process. The quality models can be easily adapted to the user’s specific goals and can be used

Crocodile environments to provide a broader platform for experimentation.

To use the measurement results for further analyses an export function can be used to transform them into an HTML document. This allows for structured presentation or further statistical processing with spreadsheet tools.

5 Experiences and Future Work

Here, we briefly report about our experiences with the integration of the Crocodile tool into SNiFF+, and also about experiences with using the tool itself.

5.1 Experiences with SNiFF+ as Integration Platform

We use SNiFF+ version 2.3.1 running with the Solaris 2.5 operating system. In our group SNiFF+ was used as programming environment before. In the following we restrict the discussion only on the integration interfaces, namely SNiFFAccess and the Symbols in Crocodile. First of all, one has to say that SNiFF+ provides very reasonable interfaces for tool integration. However, we experienced the following limitations and would suggest some enhancements of the integration interfaces.

6 References


[Tak96] Advertising material from TakeFree Software GmbH, Jakob-Haringer-Strasse 8, 5850 Salzburg, http://www.takefree.co.at