A PRODUCT METRICS TOOL INTEGRATED INTO A SOFTWARE DEVELOPMENT ENVIRONMENT

Claus Lewerentz, Frank Simon
Software and Systems Development Group
Computer Science Department, Technical University of Cottbus
P.O. Box 101344, D-03013 Cottbus, Germany
(lewerentz, simon)@informatik.tu-cottbus.de

INTRODUCTION:
The goal of the project *Crocodile* is to provide concepts and tools for an effective usage of quantitative product measurement to support and facilitate design and code reviews. Our application field is the realm of object oriented programs and, particularly, reusable frameworks.

The main concepts are
- Measurement tool integration into existing software development environments, using existing tool sets and integration mechanisms,
- mechanisms to define flexible product quality models based on a factor-criteria-metrics approach,
- the use of meaningful measurement contexts isolating or combining product components to be reviewed,
- effective filtering and presentation of measurement data.

Our current implementation platform for a tool providing these concepts is TakeFive's *SNiFF+*, an industrial strength integrated C++/Java programming environment (cf. [Pfei97]). *Crocodile* provides comprehensive static analysis features like those implemented e.g. in *Audit-C/C++* by *SemaGroup* (cf. [Mari97]) and it can be fully integrated into an existing development environment.

MEASUREMENT ENVIRONMENT

Most large today's software systems are developed in a software development environment (SDE). To create a specialized measurement process appropriate measurement tools are necessary. Only its seamless integration into the used SDE allows to include measurement as an integral part of the software development process.

The *Crocodile* measurement tool is designed to be fully integrated into existing SDEs that supply object-oriented design and coding tools (e.g. structure editors, parsers, source code browsers) and version management. The main components of the *Crocodile* tool are abstract interfaces to the SDE services. These services are used to display critical program parts through the SDE user interface and to extract the data that are necessary to measure. 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. Then the object-oriented system model at least consists of
- classes and their inheritance relationships,
- methods and attributes with their visibility and their use relations (which attributes/methods are accessed/called from which methods).

figure 1: Necessary data for measuring OO-designs
The necessary data for measuring object-oriented designs can be represented as an entity relationship diagram as shown in figure 1. Additional information about the control flow or syntactic complexity like calculated with QUALMS or LOGISCOPE (cf. [DuWi94]) aren't considered. 

Because Crocodile does not parse the source code itself but extracts the data from the SDE internal database, Crocodile is language independent. Our current implementation platform SNiFF+ supports among other languages C++ and JAVA-projects.

FLEXIBLE AND ADAPTABLE QUALITY MODELS

As in other engineering disciplines, it is a major objective in software to improve the product quality. Software product quality is expressed both by internal structural product properties as ease of maintenance or portability as well as external quality of a software product consisting of attributes like usability and correctness (cf. [Meye97], chapter 1). 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. [Fent95], p. 42). This hierarchical approach has been implemented as a standard way to define and to measure software quality (ISO9126)).

Erni enriched such quality models by including additional layers. They relate quality criteria and metrics sets by design principles and construction rules (cf. [Erni96]). These additional layers in the quality hierarchy allow for better explanation of the measured values and, particularly, 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 programs.

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], p. 65 f).

Fenton advocates the use of such flexible and adaptable quality models and calls this the „define your own quality model“ approach ([Fent95], p. 225).

A little part of such an example quality model, enriched with the additional layer of Erni could look like displayed in figure 2.

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 goal down to the concrete measures have to be
entered. It is possible to connect one measure to different design principles and criteria. The quality model itself is used by Crocodile to provide an interpreting of the measurement results. The measures are represented by the leaves of the quality tree. 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 measures are defined using an interpretative metrics and query definition language on top of a SQL database system (cf. [LeSi97]) that consists of a table structure to implement the data shown in figure 1. 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 [Fent95]).

MEASUREMENT CONTEXTS
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 interested in analysing the self-written parts, because only these 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.

• Association including aggregation can reach another kind of distributing functionality over several classes. 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-focus, 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.

• Corresponding to the inheritance-context, Crocodile provides the possibility of selecting an use-context. When measuring a focus there are only considered references of used attributes and methods from classes within the focus. Selecting an additional use-context gives the possibility to selectively include use-relations to outside the class focus.

FILTERING 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 2). The definition of such value partitions or, more generally, quality levels (cf. [ISO9126]) provides 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. \texttt{number\_of\_all\_methods \[0,5\]}  
• values are critical if they are outside an absolute interval (as shown in figure 3),
  e.g. \texttt{number\_of\_public\_attributes \[0,0\]}  
• values are critical if they belong to the group with the \(x\) highest respectively lowest values,
  e.g. \texttt{length\_of\_method \[max,x\]}  
• values are critical if they belong to the group with the \(y\) percent highest respectively lowest values,
  e.g. \texttt{weighted\_class\_complexity \[min,y\%\]}

The measures together with such critical value ranges form the leaf level of the above described hierarchical quality model as for instance shown in fig. 2.

EXPERIENCES

After very encouraging experiments with a previous prototype of a simpler metrics tool (cf. [ErLe96]), we are currently using the \textit{Crocodile} tool for three kinds of software development projects:

1) Together with industrial partners we are reviewing large programs written in Java and C++ (several hundred classes). We are measuring these source codes to prepare and support the developers’ review activities. This is done by focusing the review to classes, methods or attributes that got the most critical measurement values. In our first experiments the analysis results of \textit{Crocodile} proofed to be a good starting point for reviewing.

2) At classroom level students are using \textit{Crocodile} for object-oriented design and re-engineering. It helps them to

• learn object-oriented concepts: Programs written in an object-oriented language without object-oriented thinking are easily detected by \textit{Crocodile}. By using reduced quality models the attention of the students can be directed towards special properties of object-oriented programs (data encapsulation can be checked e.g. with the measure \texttt{number\_of\_public\_attributes}).

• judge the effectiveness of re-engineering and consolidation steps: Comparisons between measurement values „before“ and „after“ can validate the effects of restructuring activities.

3) \textit{Crocodile} can be embedded into a process model for quality assessment. In the phases "Assessment preparation" and "Assessment" \textit{Crocodile} is used. A separate phase "assessment reflection" allows of investigating the effectiveness of the usage of \textit{Crocodile} and the usability of some measures.

\textit{Crocodile} provides quite simple but powerful means to create a specialized measurement process. The quality models can be easily adapted to the user’s specific goals and can be used to support different activities in engineering and re-engineering of object oriented applications. Due to \textit{Crocodile}’s integration into a software development environment like \textit{SNiFF+} the measurement activities are smoothly integrated into existing software development processes. It can help to improve the quality of the produced software and, therefore, it seems to be a useful tool for the individual software engineer as well as for quality management.
REFERENCES


