CSECS 2011, pp. 000 - 000

The 7 Annual International Conference on
Computer Science and Education in Computer Science,

July 06-10 2011, Sofia, Bulgaria

FACILITATING QUALITY ASSURANCE THROUGH A SOURCE CODE METRICS FRAMEWORK

Nikolay GROZEV, Neli MANEVA, Delyan LILOV

***Abstract**:* The paper describes how a source code metrics framework can support quality assurance. The framework, comprising of a core framework and its extensions, is briefly presented. The described views for different groups of software practitioners show how the proposed framework functionalities can be integrated into their everyday professional workflows. Last, the preliminary results of using a framework prototype are mentioned thus proving the feasibility of our approach.

***Keywords**: Quality assurance, source code analysis, software metrics, framework.*

***ACM Classification Keywords**: D.2 Software Engineering*

Introduction

Optimizing software engineering activities has always been a top priority for researchers and practitioners in the field, especially in the circumstances of economy recession. Thus we decided to extend previous research on source code metrics, focusing our attention on quality assurance through static analysis. This activity is recognized as crucial and quite effective for improving the economical parameters of the software development during the whole software life cycle.

Searching for some science-based solutions, we decided to use the so-called CCC approach, which should be:

- Constant – to apply a systematic rather than ad-hoc approach, following a consistent and long-term strategy;
- Continuous – to start with some procedural regulations and their use for a few selected activities and only after their successful adoption to move to other ones. This approach can be combined with an incrementally developed framework, comprising a basic set of tools, which later can be enriched with additional functionality;
- Correct – based on some validated techniques and best practices – e.g. some of the suggested by Jones [Jones, 2010].

This approach is quite suitable for optimizing the so-called "umbrella" activity – quality assurance, comprising a number of different in scope and complexity sub-activities: quality analysis, measurement, control, etc. For example, we can use the results from static program analysis not only for checking source code correctness, but also for achieving other goals as improvement of source code understandability, maintainability, etc.

The paper is organized as follows: The first section describes briefly the design and the structure of the core framework for static source code analysis. The successive several sections are devoted to both technical and conceptual extensions of the core framework, aimed at facilitating its incorporation into the daily life of the software professionals. Section "*Views to the framework"* describes how the framework can facilitate the different roles in the software lifecycle. Section "*Prototype status and its preliminary validation*" summarizes the results of experimental use of a prototype of the framework, thus proving the feasibility and usefulness of our approach. In “*Conclusion and future work*” some ideas for further research and development are shared.

Core framework – a brief overview

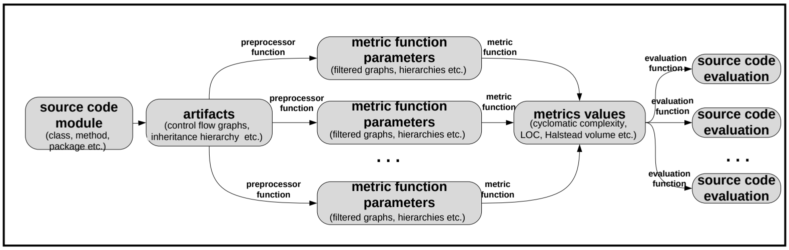
In a previous paper [Maneva, 2010] a detailed study of the different approaches to analyzing source code through metrics has been presented, together with a flexible but abstract framework, designed to overcome the majority of the identified in the study problems. In this paper we shall call this abstract framework – ***core framework***. The presented research herein extends the previous abstract framework by adding and motivating features that would allow it to incorporate in the workflow of various software professionals. In this section the previous framework is briefly presented thus making the paper reading easier. Then again the reader is advised to get acquainted with the previous paper [Maneva, 2010] for more details.

The main idea of the core framework is to provide a general template for automatic extraction of useful knowledge derived from the values of a predefined set of metrics called the ***base set***. The core framework can be thought of as a general source code quality evaluation scheme, which can be tuned through user specified logic. This user logic is “hooked” into the quality evaluation scheme in a predefined way so as to influence the eventual results in accordance with some contextual information.

The core framework comprises a number of separate modules, interacting with each other, which can be modeled as functions. The different types of functions that constitute the core framework are:

- Metric functions – their purpose is to extract the values of the metrics. For each metric from the base set of metrics there is a single “hardcoded” metric function in the core framework. Each metric function takes as input a number of source code artifacts and uses them to compute the value of the metric.

- Preprocessor functions – their purpose is to prepare the artifacts used by the metric functions. For each of the metric functions in the core framework there must be a preprocessor function.
- Evaluation functions – their purpose is to combine the values of the metrics from the base set of metrics into a meaningful evaluation of the code quality.



source code module (class, method, package etc.)
artifacts (control flow graphs, inheritance hierarchy etc.)
preprocessor function
metric function parameters (filtered graphs, hierarchies etc.)
metric function
metrics values (cyclomatic complexity, LOC, Halstead volume etc.)
evaluation function
source code evaluation
. . .

Figure 1 Evaluation scheme

Figure 1 depicts the whole scheme. First the source code is transformed into a set of artifacts – control flow graphs, dependency graphs, inheritance trees, etc. Describing this extraction is beyond the scope of this paper. After that the preprocessor functions are used to prepare these artifacts for the metric functions. This preparation usually consists of cleaning the artifacts from irrelevant data. As an example the preprocessor function for the Depth of Inheritance Tree (DIT) metric would normally remove from the inheritance tree the nodes that are irrelevant to the class being evaluated. After the artifacts are preprocessed by the preprocessor functions, they are passed to the metric functions which compute the actual values of the metrics from the base set. After that the evaluation functions are used to produce useful evaluations of the code.

As mentioned earlier, the core framework accommodates extension points, where user specified logic can be “hooked”. These extension points are the preprocessor and the evaluation functions. By specifying context dependent preprocessor functions a user can achieve metrics values with minimal “noise” which can be used as a basis for further analysis.

The purpose of the evaluation functions is to encapsulate specific logic about how to combine the metrics values. For example, such knowledge can be an identified design anti-pattern (a.k.a. “bad smell”). The approaches from [Munro, 2005] and [Lanza, 2006] could be used as a basis for such evaluation functions. Thus we consider that our core framework can fully accommodate these anti-pattern recognition strategies. In fact we believe that the usefulness of these strategies could be augmented by properly designed preprocessor functions. They can result in metrics values with less “noise” being used by the evaluation functions to do the actual recognition of the “bad smells”.

Evaluation functions can also be used to produce new numerical evaluations of the quality of the source code - e.g. an evaluation of methods maintainability in the range 0 - 10.

Extended framework – general description

The proposed core framework provides only the basic technical structure meeting the stated requirements. It is essential to devise ways to incorporate the framework into the daily work of almost everyone involved in the activities during the whole software lifecycle. This is important in order to improve everybody’s awareness of the estimated source code quality degree and needs for improvement.

To do so a lot of both technical and non-technical problems regarding the user - framework interaction need to be addressed. Some of these may seem as implementation specific details. However, in our opinion the difficult interaction of the users with some existing tools is one of the main reasons for them not being widely adopted. Thus the usability of the environment in which the technical framework is provided is central in our study.

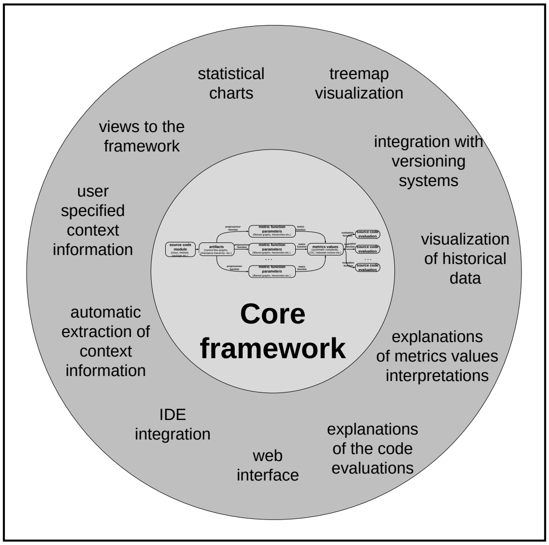

statistical charts
treemap visualization
views to the framework
integration with versioning systems
user specified context information
visualization of historical data
source code module (class, method, package etc.)
artifacts (control flow graphs, inheritance hierarchy etc.)
preprocessor function
metric function parameters (filtered graphs, hierarchies etc.)
metric function
metrics values (cyclomatic complexity, LOC, Halstead volume etc.)
evaluation function
source code evaluation
. . .
Core framework
automatic extraction of context information
explanations of metrics values interpretations
IDE integration
web interface
explanations of the code evaluations

Figure 2 Core framework and its extensions

In order to improve the overall usability of the core framework and to allow users to focus on quality control, we propose a number of extensions of the core framework augmenting its original functionality (see Figure 2). Unlike the core framework designed after a mostly thoughtful analysis, the extensions originate from a practical one. They have been “inspired” by our experience trying to incorporate a prototype of the core framework in the work of our fellow software engineers.

Next follows the description of different technical and conceptual extensions.

Extended framework - Interface

In order to be useful throughout the whole software lifecycle the framework should integrate well with the users' usual workflows and tools. The roles of the participants in a software process vary – from software developers to managing staff. Thus the framework should be incorporated into integrated development environments (IDE) and project management systems.

The non technical staff should be provided with a web interface. The interface needs to be well integrated with the most popular web based project management and issue tracking systems. This would provide the management with the ability to easily correlate source code maintainability information with other managerial information. The technical staff has to be provided with IDE integration so as to access the source code and the analysis results from the same place.

Extended framework – metrics values visualizations

In the last decade there have been some studies about the combined usage of metrics and visualization techniques (statistical charts, treemaps, spider charts, polymetric views etc.) to spot source code modules with poor quality and maintainability [Lanza, 2006] and [Diehl, 2007]. Our observations show that these techniques can increase a lot the benefits from the framework. Thus both the IDE and the web interface should employ such techniques.

More specifically, we found that various techniques for visualization of software evolution [Diehl, 2007] can be beneficial for the framework users. Visualizing historical trends of metrics values and code assessments is crucial for the continuous monitoring of source code maintainability and provides a way for early detection of problems (i.e. a junior developer is committing code that is not reviewed, etc.). Natural sources of historical analysis information are the versioning systems and the framework should be integrated with the most popular ones like SVN [SVN, 2011], and CVS [CVS, 2011]. The historical analysis information should be accessible at least through the web interface, because it should be available to the managing staff. IDE integration may also be useful for senior engineering staff.

Extended framework – explanation system

Most software engineers find it difficult to interpret the values of a set of metrics [Maneva, 2010]. This problem can be partially solved by providing interactive information on the metrics values from the base set. This information should explain the interpretations of the metrics values. Usually this is easy to implement by "hard coding" the explanations for distinct groups of values for each metric. This approach is used by some of the existing metrics tools.

Besides the metrics values from the base set, the aggregated knowledge (resulting from the evaluation functions) sometimes also needs to be explained. The primary reason is that the user may disagree with the produced code assessment. In this case the user may need explanations about the reasons the framework gave that assessment in order to decide how to proceed. The presence of such explanations is essential to the usability of the framework since it is often the only way to distinguish the so-called false positives and to take corrective actions. Besides such explanations may have certain educational effect especially on the less experienced software developers.

Thus an explanation system for the values of the base metrics and the extracted knowledge should be created "on top" of the core framework. Explanation systems are a subject of the Expert system field. Even though there are some achievements in this area which seem plausible for our goals, automatically explaining the extracted knowledge from the framework is not a straightforward task. This is because the evaluation functions represent user defined logic and only a few assumptions about their properties can be made. Our preliminary research in the area shows that by using domain specific knowledge it is possible to create a usable explanation system suitable for our purposes.

Extended framework - settings

The proposed core framework represents solely a source code evaluation scheme. That is the framework is not usable without its user specified preprocessor and evaluation functions. Defining the correct functions however may be a time consuming and tedious task requiring significant expert knowledge.

The solution to this problem comes from the fact that the suitable functions depend on the context in which they are used. Thus a set of both preprocessor and evaluation functions can be predefined for often recurring contexts. This gives the users an easier way to tune the framework by specifying the context information about what is being evaluated. For example a user may specify that a group of classes represent GUI components written with a popular GUI library. Based on this information the framework should automatically set the proper preprocessor function for the metrics used to evaluate these classes. Also evaluation functions for different assessments of these classes should be set automatically.

The idea of defining a set of functions for recurring contexts can be further extended. The user should be able to specify the context in a much wider sense than specifying it for separate source code modules (e.g. classes, methods etc.). For example a user may specify that a given application represents a web system, created with popular technologies (e.g. ASP, JSF, Hibernate, etc.) and the application is meant to be a content management system (CMS). Based on expert knowledge about the specified technologies and application domains it is then possible to infer how the different source code modules should be measured and evaluated. Similar approach has been used by some static analysis tools. Once again the algorithms for such automatic extraction of settings (preprocessor and evaluation functions) are not straightforward and require considerable amount of expert knowledge. However our study showed that these algorithms can dramatically reduce the need for user input and thus improve substantially the usability.

The need for user input can be reduced even more by applying algorithms for automatic context detection. These algorithms would typically exploit knowledge about the used binary files (e.g. dll or jar files) and the "import" statements in the source files to infer which technologies are used by the code. Other techniques for context recognition may use heuristics about naming conventions and the physical or logical structure of the source files typical for some design approaches and technical frameworks. The contextual knowledge extracted by these algorithms can be used to automatically define framework settings without any user input. This contextual knowledge however may sometimes be partially incorrect or incomplete and thus may need a user correction.

In order to minimize the needed user input we propose a stepwise procedure for creating the settings of the framework. First the user should review the automatically extracted knowledge about the system being analyzed, correct the mistakes (if any) and optionally input additional context information. Then this information is used to create the initial settings of the framework. After that the framework should be in a usable state but still may need further tuning. The user can provide this tuning by reviewing the framework analysis results extracted with the initial settings. Whenever the user disagrees with the assessment of a code module he or she may consult the explanation system and make a decision whether to correct the settings of the framework for this module by specifying additional context information for it.

Of course a user should not be banned from changing the settings directly, but we believe that the procedure described above represents the easiest, quickest and least error prone way to defining the framework settings.

Views to the framework

The described above features and workflows certainly can augment substantially the functionality of the framework. However we did not specify which members of a software development team should access these functionalities and how using these functionalities can be integrated into their professional workflow. We believe that predefining the ways the different groups of users interact with the framework is important to its usability because it lessens the possibility of misuse. Thus we consider that the framework should facilitate the different roles in the software lifecycle by providing different user account capabilities. We call the capabilities of a group of user accounts a ***view*** to the framework. Typically a view to the framework defines the way a user accesses the system (through an IDE or web) and the accessible functionalities.

An organization using the framework may define its own views to the framework in accordance with its internal structure, regulations and processes by defining the corresponding groups of user privileges. We propose several different views considering the main roles in the development process:

View for source code developers

This view is aimed at the software engineers creating, testing and maintaining the actual source code and thus allows access through an

IDE only. It should provide real-time access to the information about the code maintainability while writing, testing or debugging the code. Thus this view includes access to metrics values, aggregated knowledge (resulting from the evaluation functions) and all available visualizations. The represented information should be easily tracked to the original source code. The view also includes access to the explanation system which should give the developer explanations about the analysis results and ideas for improvements. This way the explanation system is expected to have certain educational effect especially on the less experienced engineers.

This view is also very useful when testing, since it allows easily spotting the modules posing potential maintainability problems. Such modules should be tested with many regression tests, so that they can be changed in future with less probability of undiscovered regression problems.

View for senior software engineering staff

This view gives access to all features of the framework. It should provide access to the analysis results in the same way as in the previous view.

In our opinion one of the main threats for the source code quality is the work of inexperienced developers, or developers who do not understand well the design and architecture of the software being developed or maintained. Thus this view should give access to high-level and historical information to enable the continuous monitoring and control of the developers' work. Both web access and IDE access are allowed. This is the only view that gives direct access to the settings of the framework since they require high technical expertise.

View for managerial staff

This view allows access only to the web interface. It should provide the ability to monitor the quality of the code and how it changes during the development so as to facilitate certain managerial decisions. Thus only summarizing visualizations and statistical charts showing aggregated information are accessible. Historical information about the state of the code is also provided so as to monitor the quality status over time.

Framework prototype and its preliminary validation

As mentioned, we have created an initial prototype of the core framework for experimental purposes. Based on our attempts to incorporate it into the work of a team of fellow practitioners some extensions have been defined. Each of them is justified by a need that we have identified when considering the difficulties reported by the team. For example, the need for an explanation system was identified after team members complained about not understanding the reasons for some code evaluations.

Besides a prototype of the core framework we have also implemented prototypes of a few of the described extensions, namely: Eclipse IDE integration, statistical charts, treemap visualization and tables of interpreted metrics values. Currently the framework prototype can be tuned only by a limited amount of context information that is manually input by the users. No explanation engine, historical information and support for views have been provided yet. Hence the practical benefits of these have not been validated yet.

The feedback for the implemented visualization techniques is generally positive. The adopting team found a great benefit of the close integration of the visualizations within the IDE, which allows easy navigation between them and the source code. Senior team members have reported that the visualizations saved a lot of time when conducting code reviews, because they can guide the reviewer to the code modules posing eventual maintainability threats.

The negative feedbacks were mostly related to the lack of explanations of the code evaluations and the absence of easy ways to tune contextually the framework. Actually the team could not manage to create the needed context settings itself and thus experienced some problems with “false positives”. Since the framework was accessible only through an IDE, it was not possible for members of the non-technical staff to test it. We believe that the proposed and not yet prototyped extensions will help to overcome these issues.

Conclusion and future work

In this paper a framework, supporting quality assurance is presented. We hope that such framework will meet the requirements for efficient, tool-supported and user-friendly performance of this essential software activity.

Our ideas for further research and development are in the following directions:

- To implement in a prototype all of the proposed in this study core framework extensions and to examine them;
- To study more source code metrics so as to decide which of them can be added to the base set of metrics.

Acknowledgements

This work has been partially supported by the Bulgarian National Science Research Fund (project ДМУ02/18 / 18.12.2009).

Bibliography

[CVS, 2011] CVS - http://savannah.nongnu.org/projects/cvs, May 31, 2011.

[Diehl, 2007] S. Diehl. Software Visualization. Berlin: Springer Verlag, 2007.

[Jones, 2010] C. Jones. Software Engineering Best Practices, Mc Grow Hill, 2010.

[Lanza, 2006] M. Lanza, R. Marinescu. Object-Oriented Metrics in Practice. First edition, Berlin: Springer Verlag, 2006.

[Maneva, 2010] N. Maneva, N. Grozev, and D. Lilov. A Framework for Source Code metrics. Proc. of the International Conference "CompSysTech'2010", Sofia, Bulgaria, pp.113-118.

[Munro, 2005] M. J. Munro, "Product Metrics for Automatic Identification of “Bad Smell” Design Problems in Java Source-Code", 11th IEEE International Software Metrics Symposium, 2005

[SVN, 2011] SVN - http://subversion.tigris.org/, last visited May 31, 2011.

Authors' Information

Nikolay GROZEV,

nikolay.grozev@gmail.com

Major Fields of Scientific Research: *Software Engineering, Software Architecture, Quality Assurance, Software Measurement*

Neli MANEVA, Assoc. Prof. PhD, Institute of Mathematics and Informatics, BAS, “Acad. G. Bonchev” Str. Bl. 8, Sofia 1113,

e-mail: neman@gbg.bg,

Major Fields of Scientific Research: *Software Engineering, Quality Assurance, Software Measurement*

Delyan LILOV, Musala Soft Ltd. 36 Dragan Tsankov blvd. Sofia, Bulgaria +359 2 969 58 21,

e-mail: delyan.lilov@musala.com

Major Fields of Scientific Research: *Project management, Software Engineering*