Information and Software Technology 57 (2015) 32–51

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A concern-oriented framework for dynamic measurements

Walter Cazzola [a,*], Alessandro Marchetto [b]

[a] Department of Computer Science, Universitá degli Studi di Milano, Italy
[b] Fondazione Bruno Kessler, Trento, Italy

ARTICLE INFO

Article history:
Received 30 July 2013
Received in revised form 19 August 2014
Accepted 20 August 2014
Available online 28 August 2014

Keywords:
Software measurements and metrics
Static and dynamic software artifact analysis
Software feature and concern

ABSTRACT

Evolving software programs requires that software developers reason *quantitatively* about the modularity impact of several concerns, which are often scattered over the system. To this respect, concern-oriented software analysis is rising to a dominant position in software development. Hence, measurement techniques play a fundamental role in assessing the concern modularity of a software system. Unfortunately, existing measurements are still fundamentally module-oriented rather than concern-oriented. Moreover, the few available concern-oriented metrics are defined in a non-systematic and shared way and mainly focus on static properties of a concern, even if many properties can only be accurately quantified at run-time. Hence, novel concern-oriented measurements and, in particular, shared and systematic ways to define them are still welcome. This paper poses the basis for a unified framework for concern-driven measurement. The framework provides a basic terminology and criteria for defining novel concern metrics. To evaluate the framework feasibility and effectiveness, we have shown how it can be used to adapt some classic metrics to quantify concerns and in particular to instantiate new dynamic concern metrics from their static counterparts.

1. Introduction

A *concern* is any consideration that can affect the implementation and maintenance of program modules [1]. In particular, *a concern is identified by portions of code not necessarily contiguous that contribute to implement such a concern*; the concern can be *selectively* exercised through *ad hoc* scenarios defined by, e.g., use cases or test units. A software requirement or functionality, for instance, is a concern while the dynamic counterpart is the execution of a requirement or functionality. As an example, the services provided to the user by a software system that controls an automated teller machine (ATM) are concerns.

Normally, the software is developed reasoning in term of the features[1] it must provide but the tangled nature of the resulting application forces the maintainer to reason quantitatively about their modularity to facilitate its maintenance. With the increasing relevance of concern-oriented programming, see, for example, the advent of aspect-oriented programming (AOP) [2] and feature-oriented programming (FOP) [3], there is an urge to revise existing metrics (as done by [4]) and to develop new ones supporting concern quantification against software variability. For instance, some studies [5,6] suggested that an increment to software modularity might correspond to: (i) an increment of undesirable couplings involving the realization of two or more concerns; and (ii) a decrement of the cohesion among the elements realizing a concern. *This kind of concern-specific design anomalies are key factors to decrease software maintainability.*

However, to provide an accurate characterization of how a concern affects a program is not a trivial task [4]. Many concerns are often tangled and scattered across a number of modules and, therefore, there is no direct traceability between a concern and the module boundaries [1]. The mapping between concern and code modules—i.e., "where the concern is implemented in the code"—is not always well-documented and well-preserved during the system design, implementation and maintenance. In such cases, the mapping between concerns and code modules can be inferred by static code analysis (to the static extent) and completed by dynamically exercising the concern, e.g., via test units (to the dynamic extent) [7]. As a result, concern-specific properties cannot be detected by applying conventional module-oriented metrics and proper variants of such metrics have been investigated in the literature, such as [4,8,9].

By analyzing the existing literature in the field of concern-oriented metrics, however, we observed two main limitations:

* Corresponding author.
E-mail addresses: cazzola@di.unimi.it (W. Cazzola), alex.marchetto@gmail.com (A. Marchetto).

[1] In the rest of the paper, *feature* and *concern* will be used as synonyms.

http://dx.doi.org/10.1016/j.infsof.2014.08.006
0950-5849/© 2014 Elsevier B.V. All rights reserved.

1. Existing metrics are not systematically defined, that is, there is a lack of shared frameworks or approaches that can support the systematic definition of concern-oriented metrics. In fact, to the best of our knowledge, there exists only one measurement framework (i.e., the one described in [9]) devoted to design and describe concern-oriented metrics; all the others frameworks available in the literature—such as [10,11]—only support module-oriented metrics, thus they cannot be reused as-is to define new concern-oriented metrics. Consequently, designers of measurement tools cannot rely on formal, systematic and shared terminology, set of notions and criteria to: define and describe concern metrics and systematically validate and compare them, e.g., with the existing ones. This leads to ambiguous and overlapping metric definition that hampers the adoption of concern metrics in academic and industry settings and the execution of empirical studies using these metrics in general.
2. Existing concern-oriented metrics are mainly static, i.e., they quantify statically-computable properties of a concern, as we have identified in a recent systematic study [9]. However, as happens in the case of software modules [10,11], some relevant properties of a concern can only be precisely discovered though the concern execution [4], such as dynamic coupling or cohesion. Static and dynamic metrics are hence complementary also at concern-level as well as at module-level. In fact, static metrics are conservative and can lose precision since they are based on static analysis of software artifacts (e.g., source code), while dynamic metrics are strongly tied to specific software executions, thus they can be more precise than the static ones but they can suffer of under-approximated results, i.e., the part of the system not executed is not considered in the metric computation.

This paper presents a contribution in this field by providing a concern-driven framework for defining and describing both static and dynamic metrics, at both module and concern levels. In particular, the presented framework extends and complements our measurement framework presented in [9] by capturing run-time properties that can be quantified for a concern and how they can be obtained. The framework is composed of a group of terms, notions and criteria for defining and comparing dynamic concern metrics beyond those for defining and comparing static concern metrics.

We evaluated the presented framework's feasibility and effectiveness in two ways. First, we conducted an experiment (Section 6) where some subjects (students) have used the framework to instantiate some dynamic concern-oriented metrics from their static or module-oriented counterparts; the goal of this experiment was to answer the research question: (RQ1): *"Can the framework be used to describe several concern-oriented metrics using a common and precise terminology and set of concepts?"*. Second, we reported on a case study (Section 7) where we used some dynamic and static concern oriented metric to measure a pool of open source applications; the case study has been carried out with the goal of answering to the research question: (RQ2) "*Are the dynamic concern-oriented metrics useful to predict the concern bug-proneness?*". This case study aimed at showing utility and effectiveness of such dynamic concern metrics for bug-proneness estimation.

The rest of the paper is organized as follows. In Section 2 we present a survey of existing maintainability measurements and describe their adaptation to quantify dynamic properties. Furthermore, we stress the relevance of dynamic measurement by examples. In Section 3 we analyze the specific characteristics of measuring concerns dynamically, that are in particular, concern mapping and triggering, as well as a tool supporting the identification of a concern and its components at runtime. We introduce the framework in Section 4 and the criteria composing it in Section 5. Section 6 provides an experimental evaluation of the proposed framework by metrics instantiation while Section 7 reports a study we conducted about the usage of dynamic measurements instantiated at concern-level through the presented framework. Finally, Section 8 summarizes the state-of-the-art about metric frameworks, and in Section 9 we draw our concluding remarks.

2. Towards dynamic concern measurement

To support dynamic concern-driven metrics definition and measurement we had to understand which properties and notions characterize a concern at run-time and whether it is worth measuring. Since the literature on dynamic concern measurement is scarce[2] we have looked at the literature about metrics (both at module and concern level) and dynamic properties of software systems for identifying such properties and characteristics. Therefore, to have a wide and comprehensive understanding of the concern's properties, we studied and adapted some existing static concern metrics and some well-accepted module-oriented metrics to quantify dynamic concern properties. Such an approach permitted to cover a larger amount of possible measurements and relevant properties that might otherwise be overlooked. Out of these findings, then, we defined a set of framework criteria that capture such properties and that make the framework complete and effective enough to describe existing and new dynamic concern-oriented metrics.

In the rest of this section we present the result of our investigation, in particular we show how the considered metrics have been adapted to the dynamic and/or concern-oriented context. We have classified the considered metrics according to their original characteristics as follows:

- Dynamic module metrics. These are dynamic module-driven metrics originally defined for object-oriented systems. They were adapted or extended to be applied to concerns as well.
- Static concern metrics. These are static metrics originally defined for concerns. They were adapted or extended to be dynamically applied.
- Dynamic concern metrics. These are dynamic metrics already defined for concerns that do not require any adaptation.

Table 1 summarizes the result of the literature review we conducted. The table shows the considered suite of metrics and it reports for each metric the original definition (column "Original Definition") present in the literature and the definition obtained from our adaptation (column "Modified Definition"). To complete the picture, in Table 2 we report the definition of those metrics that are already defined as dynamic and concern-oriented and therefore that do not need any adaptation in order to be considered.

The adaptation process is quite straightforward and relies on the adoption of the concept of *concern execution* that corresponds to the execution of the elements composing the concern that can be prodded by, for example, an *ad hoc* use case or test unit. If the considered metric is dynamic but not concern-oriented we mapped the subject and/or the target of the measurement to the concerns; whereas if the metric is already concern-oriented but not dynamic we have exclusively considered the events that occur during the execution. For instance, in *Concern Diffusion over Operations (CDO)* we look for components that participate in the concern

[2] To the best of our knowledge, [4] is the most relevant piece of work in this field by introducing *disparity*, *concentration* and *dedication* metrics.