3 M-Track: A Metric Tool Framework for Monitoring the Evolution of Object-Oriented Systems

Dharmalingam Ganesan
Jean-François Girard
Fraunhofer Institute for Experimental Software Engineering Sauerwiesen 6, 67661 Kaiserslautern, Germany
{ganesan,girard}@iese.fraunhofer.de

Abstract

This article reports about M-Track, a metric tool framework for tracking the evolution of Object-Oriented (OO) systems. It tracks the evolution using metrics reflecting cohesion, coupling, inheritance, and size. M-Track was applied for analyzing the evolution of a product family of systems in the domain of stock market.

3.1 Introduction

Maintaining the existing software effectively is an important activity for any organizations that develop systems that heavily depend on software. As software systems become older, changeability, understandability and testability start decreasing unless maintainers take active measures to prevent it.

One promising approach to control the maintenance of legacy OO software system is monitoring its evolution. It is well known that systems that adhere to the principles of low-coupling and high-cohesion are easier to maintain. But due to time to market pressure, these principles are often not followed, decreasing the maintainability of the system. On the other hand, without tools practitioners often only have a vague feeling about the degradation of coupling and cohesion between previous two releases. *M-Track helps addressing this problem.*

3.2 Applying M-Track

We use M-track's metrics to focus the attention of developers, designers and managers on the classes[1] with extreme metric values and extreme value changes. The idea is that such classes are more likely to cause problems than other. Many coupling and cohesion metrics have been proposed for OO systems in the literature. Many of them capture the same underlying concepts with more or less success depending on the system and its context. Because analyzing many partially redundant metrics requires too much effort from experts we select a subset of the available metrics. We used principal component analysis [Dun89] to select the subset of metrics that capture most of the underlying concepts.

3.3 Goals & Design of the M-Track Framework

The following are the major goals that drive us to the design the M-Track framework and the solutions we have selected.

- OO language Independent: The goal is to minimize the modification effort to apply M-track to a different oo language.
  This goal is achieved by decoupling the fact extraction from the metrics computation and by using a different fact extractor for each language. Each

[1]The metrics can also be applied at higher level of abstraction like namespace or packages.

fact extractor produces a standard fact representation (RSF) from which the metrics are computed.

- Extendable : M-Track should be extendable in order to introduce new metrics easily and also be able to customize the evolution analysis to support the needs of the analyst. We made M-Track easier to extend by separating the infrastructure needed by multiple metrics and by putting the specific part of each metric in its own module.
- Efficient and Scalable: The main target of M-Track is analyzing the evolution of large industrial systems. There scalability is the main concern.
  To achieve an efficient and scalable implementation, we applied three strategies. Firstly, we identified intermediate results needed by multiple metrics and made sure that they would be computed only once. Secondly, we preprocessed the relations in the fact based, so that indirect relations could be access directly. Thirdly we optimized the infrastructure to query the fact base.
- Portable: To offer short feedback cycle on the evolution and avoid delay, M-Track should run where the code is produced. Since different industry partners use different platform M-Track should be portable.
  To achieve this portability we implemented M-Track in Perl.
- Applicable to different level of granularity: In large industrial systems, analyzing coupling and cohesion metrics at different levels of abstraction helps to get a high-level overview, then to focus on the details of extreme cases.
  To support this analysis, we defined hierarchical version of the class metrics.

M-Track computes and visualizes OO metrics with the following steps.

1. Extract the containment information about files[2], classes and method as well the relations among them for the current and previous version of the system; then store these facts into RSF files.
2. Compute metric for both versions of the system.
3. Count the number of lines added, changed and deleted for each file appearing in both systems.
4. Prepare evolution report combining the metrics and the code change. Export the results as CSV format from which Microsoft Excel generate charts to provide an overview for the people analyzing the evolution.

3.4 Case Study and Lessons Learned

We applied M-Track to monitor the evolution of a product family from stock market domain implemented in Java [GVG04]. From the time where we started applying M-Track, we introduced new metrics and refined existing metrics to apply them at the package level. Our experience shows that new metrics can be easily introduced. Furthermore, metrics computation takes only few seconds for systems that contains more than 300 Kloc of code and with around 2000 classes in it.

One key lesson learned during this case study is that it is important to keep the presentation of the results flexible. As the developers, designers, and managers of a system perform multiple workshops using the metrics, they discover new ways to analyze the results. It is important to quickly adapt the reports and the visualization of the results to make the workshops most effective.

3.5 Conclusion and Future Work

This paper reports on a tool framework called M-Track, for monitoring the evolution of object-oriented systems. M-Track is customizable to the interest of industry partners for monitoring the evolution. This customizability was achieved by computing the metrics from a language-independent intermediate model, decoupling metrics computation from the metrics visualization, and also by implementing the entire framework in a portable language (Perl). Scalability is achieved by taking advantage of the commonality among the definition of the metrics itself.
This work will be extended in three directions in future.

1. Extending the M-Track to monitor dynamic aspects of the system.
2. Identifying relationships between static and dynamic measures.
3. Choosing an appropriate visualization approach for doing useful analysis on the static and dynamic measures at different level of abstraction.

References

[Dun89] G.Dunteman, "'Principal Component Analysis"', Sage Publication, 1989.

[GVG04] J.Girard, M.Verlage and D.Ganesan. "'Monitoring the Evolution of an OO system with Metrics: an Experience from the Stock Market Software Domain"', Submitted for publication

[2]For different languages higher abstraction are also captured (e.g. packages for Java, namespace for C++).