A Longitudinal Study of Slice–Based Metrics

Timothy M. Meyers and David Binkley
Loyola College in Maryland
Baltimore, Maryland
21210-2699, USA
{tmeyers,binkley}@cs.loyola.edu

Keywords: slicing, metrics, cohesion

Abstract

Slice-based software metrics were introduced more than a decade ago. However, to date they have lacked intensive empirical investigation due to a lack of mature tools. Slicing technology is now mature enough to allow an extensive empirical investigation. Three such investigations were undertaken. The first provides baseline values for slice-based metrics; these values act as targets for future development efforts. Second, two longitudinal studies showed that slice-based metrics quantify the deterioration of a program as it ages; this serves to validate the metrics. Finally, statistical comparisons of the metrics identified those metrics that provide similar views of a program and those that provide unique views of a program.

In all, 63 programs ranging in size from about 500 to over 150,000 lines of code were considered. Over 1.1 million lines from code from 22,651 procedures were analyzed with a total of 2,067,016 slices being taken.

In his original work on program slicing, Mark Weiser proposed using metrics based on program slicing to measure program cohesion and continuity. He informally presented several sliced-based metrics. Ott and Thuss formalized Weiser's metrics for *Tightness*, *Coverage*, and *Overlap*, and proposed two additional cohesion metrics: *MinCoverage* and *MaxCoverage*.

Of the three investigations into these five metrics, the longitudinal one proved to be most interesting. This study considered seven versions of the program `gnugo` ranging from `gnugo 1.2` through `gnugo 3.4`. Over this range, `gnugo`'s average module size exhibits general growth, with the only significant decline accompanying major version `gnugo 3.0`. This version is also interesting because it is the only instance of convergence between module size and the number of statements common to all of the module's computations. Consider an idealized target program in which each module contains a single "thought." Progress towards such a target would appear as a convergence between module size and the number of statements common to all of the module's computations. For `gnugo`, this convergence supports the notion that more thought and effort went into this major release as opposed to seemingly atrophic minor releases.

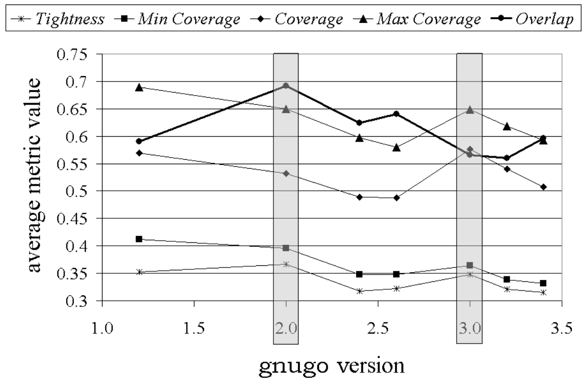

Figure 1: The longitudinal study of `gnugo`. Major releases are highlighted.

The graph of metric values for `gnugo` (Figure 1) shows a clear negative trend among all the metrics. *MinCoverage*, *Coverage*, and *MaxCoverage* are all in a state of constant decay with the exception of `gnugo 3.0`. Similarly, *Tightness* only experiences improvement for the major releases `gnugo 2.0` and `gnugo 3.0`. With the exception of *Overlap*, all of the metrics experience their maximum value within the first or second release and continue to fall thereafter. Thus, slice-based metrics capture and furthermore quantify the deterioration in cohesion and continuity that a program undergoes as it ages.

Conducting the three studies has suggested several areas for future work. For example, three follow-on empirical investigations have been suggested. The first will investigate the metric slice, a "meatier" slice that was proposed by Ott and Thuss. A second will consider the effectiveness of glue-token based metrics proposed by Bieman and Ott. Finally, analysis tools are now sufficiently mature for an in depth study of slice-based coupling between modules. Results from these future studies together with the results of this study serve to improve the understanding and therefore usefulness of slice-based metrics.