Empir Software Eng (2013) 18:177–180
DOI 10.1007/s10664-012-9212-1

EDITORIAL

Preface to the special issue on program comprehension

Giuliano Antoniol · Keith B. Gallagher

Published online: 30 May 2012

Welcome to the special issue of the 18th IEEE International Conference on Program Comprehension (ICPC), held in Braga, Portugal, 30 June–2 July 2010.

Bear with a small thought experiment: What does the following **C** function do?

```
float rt(float x1, float x2, float acc)
{
  float f, df, dx, val = 0.5*(x1+x2);

  while (fabs(f = g(val)) > acc)
    {
       df = g1(val);
       dx = f / df;
       val -= dx;
    }
  return val;
}
```

Suppose I say that `g` computes a polynomial and `g1` computes its derivative, and that `x1` and `x2` are boundary values.

Give up? It is a (ham-handed) attempt to use Newton's method to find a root of the polynomial `g`, to within accuracy `acc`. Without the right background, or even a helpful comment, it may take a while to *comprehend* what the function does, albeit

G. Antoniol (✉)
SOCCER Lab. DGIGL, École Polytechnique de Montréal, Montréal, QC, Canada
e-mail: antoniol@ieee.org

K. B. Gallagher
Florida Institute of Technology, Melbourne, FL, USA
e-mail: kgallagher@fit.edu

badly. Moreover, consider how your thinking about this code fragment would change with the simple comment: `Get root of g using Newton's method`.

Matters complicate quickly. For once one knows *what* the fragment does, the interesting questions follow. *Why* this technique at *this* place in the code? *Why* is the function `g` and its derivative hard coded. Or are they? Leaving the code aside, what about the implementation? Why this technique? Why not use the easliy available numerical recipe?

Suppose the comment was changed to `Get root of` **c** `using Newton's method`. Then question becomes *Which root?* And what changes about ones thinking if the comment says `Get square root of` **c** `using Newton's method`?

Now suppose that you have the executable and no source. In a quick-and-dirty study in my lectures, students could determine the fragment's specification in anywhere from 20 to 50 minutes. The accuracy parameter, `acc` clouded the issue; for large values of `acc`, the result seems chaotic.

Of course, this simple little thought experiment is about *Program Comprehension*. In the uncommented version, consider how long it might take to get to that "AHA" moment, and the frustration that ensues, a frustration we have all felt, when that moment occurs. And consider how your thinking changes based on the provided comments. The same kind frustration ensues when, using the executable, one realizes "is that all it does?"

So we argue that *Comprehension*, and in particular, *Program Comprehension* is central to any software engineering task. The software engineer, in any phase, must first comprehend the task.

How can this comprehension be done? Reading or Observing. And how can we as scientists help those readers and observers? Techniques for analyzing source and object abound, each with their own evangelist. It is our job, which we gladly assume, to look past the "hype-cycle" and empirically evaluate any claim that "comprehension is improved" using technique X.

As our community extends over the boundaries of the academic world, many of the issues come from software engineering professionals. This results in high-quality practical research with strong industrial case studies. While still a relatively young field, the program comprehension community is thriving and as a result there is now a substantial body of work that has been conducted in the area with important results.

The 18th edition of ICPC was characterized by strong papers and interesting discussions. Two of these discussions were sparked by two exceptional keynotes by Neil Maiden (2010) and Gail Murphy (2010). Neil highlighted how software engineering and program comprehension can benefit from creativity, how software engineering activities and program comprehension can be re-interporeted as as creative problem solving. Gail Murphy on the other hand highlights the importance of using the context to avoid developers information overload.

The invited papers for this special issue are revised and expanded versions of the 24 full papers accepted and presented at ICPC 2010. Based on ICPC review reports, presentations and discussions during the conference, we selected six papers as candidates for this special issue. These papers were subsequently extended with new material by the authors and subjected to two additional rounds of reviewing, resulting in five papers presented in this special issue.

We hope that readers will benefit from this special issue and through these papers gain useful insights into the domain of program comprehension. We would like

to extend our gratitude to all the authors who submitted papers to ICPC, to the members of the program committee and their additional reviewers for providing valuable feedback on the papers on time. Also, many thanks to those who reviewed the papers for this special issue. Their constructive feedback helped to shape the papers in this issue. Our thanks also go to Lionel Briand, editor of the Empirical Software Engineering Journal, for hosting this special issue.

We now briefly introduce the five papers of this special issue that touches a variety of subject in program comprehension ranging from the role of identifiers, to modeling software quality, to web privacy, to feature identification and software quality.

The first paper by Andy Zaidman et al. deals with classical program understanding questions in the context of new Ajax-enabled web applications. Ajax client server applications are highly dynamic and interactive. The paper reports empirical evidence that FireDetective, a tool tailored to address dynamic analysis at both the client (browser) and server-side, helps developers in the crucial program understanding activity.

Program identifiers have a paramount role in program understanding, the work by Binkley et al. empirically investigates the role of program identifiers in human comprehension and whether or not reading prose is different from reading code fragment. Results show that reading code source is different from reading and understanding natural language text and that the style of identifiers impact less experienced developers.

Very often developers are faced with the daunting task to locate fragment of code implementing domain abstractions or features (user observable functionality) in large software applications. In such a situation, techniques to support developers can be of great help. The work by Dit et al. empirically validate a novel, hybrid, feature location strategy inspired by data fusion. It uses static as well as dynamic information with web mining algorithm that improves existing approaches up to 87%.

Web privacy, how web systems store, protect and manage user data, is a growing concern. The work by Ralf Lammel and Ekaterina Pek investigates over thousands of web sites the correctness of policies declared in the domain specific language P3P (the W3C domain-specific language for privacy policies) and thus the possible actual use of user data. The paper concludes that P3P's approach to policy validation is too weak to ensure correct use of the language or declared policy enforcement. P3P usage is likely to decline and possibly was not seriously considered by organizations, for instance, results reported by clone detection show that there seem to exist many websites that do not even customize policies to their entity. Perhaps, as the authors suggest, many policies were never considered actionable or they were designed to be extremely permissive without usefully promising privacy.

Software systems are complex objects built and maintained by humans; understanding how collaborative works and social interactions impact software quality increases our understanding of software evolution. The paper by Nicolas Bettenburg and Ahmed Hassan studies the impact of social interactions on software quality. Authors model defects using predictor variables based on social information mined from open-source software repositories. The paper findings show that social information not only is useful but it complements more traditional code-based metrics and that a model based solely on on social information have a similar degree of explanatory power as more traditional models.

References

Maiden N (2010) Creativity in software engineering: a new research agenda? In: Proceedings of the 18th international conference on program comprehension (ICPC), p 1

Murphy G (2010) Context as an antidote to information overload. In: Proceedings of the 18th international conference on program comprehension (ICPC), p 1