Making Slicing Mainstream
How can we be Weiser?

Karl Trygve Kalleberg
Institutt for Informatikk
Universitetet i Bergen
PB 7800
N-5020 Norway
karltk@ii.uib.no

Tracy Hall
Department of Computer Science
Hertfordshire University
Hertfordshire, AL10 9LB
United Kingdom
t.hall@herts.ac.uk

Ran Ettinger
Programming Tools Group
Computing Laboratory
Oxford University
Oxford OX1 3QD
United Kingdom
rani@comlab.ox.ac.uk

2005-11-30

Abstract

By now, the concept of program slicing has been known in the research community for around 25 years. As a research topic, it has enjoyed a fair share of popularity, evidenced by the number of articles published on the topic following Mark Weiser's seminal paper. However, outside research circles, program slicing appears to be virtually unknown.

In this report, we take the premise that program slicing is both technically relevant, and has a sufficient theoretical foundation, to be applied in practice within the software industry. With this premise in mind, we ask ourselves, "what are the mechanisms by which we as a community could make program slicing mainstream"?

1 Introduction

The survey papers on slicing by Tip [Tip94], Binkley and Gallagher [BG96], and Binkley and Harman [BH04] tell us that program slicing has a rich and long history in the research community. While the original concept of static slicing was envisioned as a debugging and program comprehension aid, slicing has now gone in multiple directions, including dynamic slicing [KL88] and amorphous slicing [HBD03].

Despite its academic success, the industrial presence of program slicing is low, and this fact was the source of much debate at the Beyond Program Slicing seminar held at Schloß Dagstuhl, October 2005. Indeed, when asked, the seminar participants could name no existing debugger that includes slicing, though it was pointed out that for a number of years, a slicing patch against the GNU

Dagstuhl Seminar Proceedings 05451
Beyond Program Slicing
http://drops.dagstuhl.de/opus/volltexte/2006/486

Debugger existed, but that this has now fallen by the wayside and is no longer maintained.

In this report, we would like to go beyond the question we heard so often at the seminar (*"if slicing is so great, why is nobody using it?"*) and offer suggestions for how the slicing community can improve this situation. As with any marketing ploy, we take as a given that the concept and theory of slicing is relevant and applicable in the software industry. For our target audience – the slicing community – this should not be too controversial, considering the number of research papers describing possible applications of slicing produced by said community.

Our contribution towards the goal of mainstreaming slicing is to list and discuss a number of mechanisms which may be helpful in raising the awareness of program slicing. The list is a result of a brainstorming session by the authors during the Beyond Program Slicing seminar. It should be emphasized that not all the ideas are our own. The ones we have taken from other seminar attendees will be clearly marked.

The remainder of this report is organized as follows. In Section 2, we look at some other techniques and methodologies that have enjoyed mainstream popularity in the last 20 years of computing history and see if there are lessons to be learned from their stories. In Section 3, we present a (non-exhaustive) list of mainstreaming mechanisms inspired by other success stories, and ask how they might be applied to slicing. In Section 4, we show the results of a survey we conducted among the Beyond Program Slicing seminar attendees, and interpret the results in Section 5. In Section 6, we discuss what the attendees believe are the ways forward for achieving wider adoption of program slicing.

The report ends in Section 7, were we summarise and conclude our findings.

2 Successful Techniques and Methodologies

The software industry is full of successful techniques and methodologies that aid the production of software. One could imagine that the process of mainstreaming program slicing should follow an easy two step procedure: (1) Find a popular technique which is similar to slicing, and (2) investigate how that technique became so successful. It is beyond the scope of this report to offer any kind of taxonomy of development techniques and methodologies. We can, therefore, offer no such measure of similarity between techniques. However, from looking at the successful techniques described below, it is clear that their popularization shares many common mechanisms. We therefore believe that such common mechanisms are well worth considerating in the case of slicing.

The remainder of this section is devoted to brief discussions of some of the most popular contemporary methodologies and techniques for software construction. It is important to realize that we do not claim any form of equivalence between these techniques. The only thing they have in common, and the reason why we picked them, is that they are all highly popular.

Refactoring Refactoring is the reorganization of the internal structure of a program with the aim of improving design, without alternating the external behaviour of the program.

Improving programs as part of maintenance and evolution is well established. The contribution of refactoring is its collection and popularization of concrete recipies for code improvement, many of which have been built into modern development environments.

The reasons refactoring has become popular include: *attractiveness to developers* – refactorings offer concrete solutions to everyday problems encountered by developers; *tools* – the availiblity of refactorings inside modern development environments have made them instantly accessible to every developer; *books* – well-written cookbooks of refactoring recipies are accessible for many computer languages; *champion* – Martin Fowler has taken on the champion role of refactoring, by writing books, articles and also by talking directly to the developers of the popular IDEs.

Generative Programming Generative Programming is about bringing the benefits of automation to software development. It is a methodology for designing "families" of software products, so that members of such families, i.e. individual software programs, can be constructed quickly at the customer's request. Generative programming makes heavy use of models and automatic code generation.

Popularity of this approach may in part be due to *books* – the book Generative Programming [CE00] by Czarnecki and Eisenecker gave the technique a name, it has later been followed by Software Factories [GSCK05] by Greenfield et al of Microsoft; *portals* – on a related note, a web site called Code Generation Network [cod] carries interviews with researches and industry practitioners about the application of code generation in general; *commercial backing* – both the VisualStudio group at Microsoft and the developers of IntelliJ at JetBrains are investing heavily in tools releated to generative programming.

Formal Methods Using strict mathematical techniques to design programs and verify that they are correct according to a given specification is referred to as using formal methods. Reasoning about the correctness of a program involves relating program code to logic sentences and verify logically that the code implements the desired behavior.

Even though formal methods have a reputation in some circles of being an impractical technique for improving robustness and quality of software, we list it as a success story. Many high-profile, mission-critical software systems have been designed and implemented successfully using formal methods, including the software to drive the Paris Metro, the design of the Firewire protocol and the Playstation content protection scheme. A wide selection of books on various formal methods exists. Most universities and colleges offer special courses on the application of such techniques. Also, proof of correctness is common in any university level course on algorithms.

Agile Methods Agile methods is a methodology that aims at producing high-quality software in the face of changing requirements. The methodology emphasizes the importance of high quality source code and the individuals involved in writing it, and downplays the focus on tools, processes, plans and contracts.

There has been a marked rise in awareness of the tools and techniques that underlie agile methods in the later years. Over a dozen books have been written

on the topic of agile methods, spearheaded by authors like Kent Beck and Martin Fowler. The books act to complement and explain the agile manifesto [agi].

Unit Testing Unit testing is a companion technique to extreme programming and agile methods. The technique advocates writing tests of program functionality before you implement the code, thus forcing you to thing about *what* the interface and behaviour is supposed to be, before focusing on *how* it can be realized. The promise of unit testing is to catch regressions that may be introduced during code maintenance and evolution.

The unit testing technique has been made practical for developers by the free availability of small and robust unit testing frameworks, such as JUnit. These frameworks were designed by respected developers, who have since written books and fostered a community around their framework implementations.

Object Orientation Object-orientation enjoys its place as the dominant paradigm for constructing software. It is employed for constructing anything from embedded programs in mobile phones to the largest financial applications.

Over the years, the adoption of the object-oriented approach to designing and implementing software has been helped by the presence of good and relevant books on the topic; quality undergraduate courses in universities and colleges; solid development tools; many high-profile success stories as well as vocal and respected champions such as Bertrand Meyer and others.

From examining this list of success stories, we observe the presence of many possible mainstreaming techniques. In the following section, we will discuss many of them in greater detail.

3 Mainstreaming Mechanisms

This section lists a number of possible approaches to popularizing and mainstreaming program slicing. This is not a list of independent choices. On the contrary, there are clear relationships between some of the alternatives. For example, implementing better tools is in all likelihood a prerequisite for making slicing attractive to ordinary developers, but these tools may come from the industry itself if a business case for slicing could be made.

1. A Champion – A Slicing Tsar or Tsarina

Does the slicing community need a slicing champion?

By this we mean a person who takes on the job of advocating and lobbying for program slicing in various fora, such as industry conferences, magazines and journals, one who socalizes with industry practitioners and developers of established tools and methodologies.

2. A Slicing Consortium

Do we need a formal entity, like a consortium, to take on the job of defining and advocating slicing?

In the modeling community, there was a unification of several alternative modeling approaches and languages throughout the 90s into the now popular UML language, which is controlled by the Object Management Group. A similar organization, be it a foundation, consortium or other legal entity, could be

founded to garner industry support and contribution in the process of mainstreaming slicing.

3. A Slicing Manifesto

Is having a clearly written manifesto which proclaims the purpose, techniques and philosophy of slicing useful for popularizing slicing?

The availability of manifestos has historically been instrumental in many political as well as technical movements. The GNU Manifesto by Richard Stallman helped set a vision for the GNU project and the development of Free Software. The Agile Method community also has a manifesto that explains the philosophy behind their development methodology. A similar vision for slicing may help fix the idea in the minds of beginning practitioners.

4. More Industry Involvement

Will recruiting industry support be an effective way of reaching the mainstream?

Time and again, we see that some software companies decide to fully embrace a particular approach or technique, at least to get good PR. IBM did this with the philosophy of open-source a few years ago, Microsoft is pursuing service-oriented architectures (SOAs), Sun decided to really push Java, even through TV ads, IBM is persuing autonomic computing, which at present is mostly a vision. While slicing is not a philosophy nor a language, having an established industry mover may do a lot to raise awareness and produce the tools and know-how required for widespread industrial application.

5. More Empirical Evidence

Does slicing need more empirical evidence before people will start to apply it industrially?

A recent survey by Binkley and Harman [BH04] documents the current state of empirical evidence in favour of slicing, but does not offer unassailable evidence for the usefulness of slicing. More evidence would make it easier to sell slicing to software developers and organizations.

6. University and College Education

Will teaching slicing to undergrad students help slicing reach the "trenches" and from there gain wide acceptance?

In most undergrad programmes, courses on various programming paradigms are given, and for software engineering programmes, the popular development methodologies are also covered. By covering slicing in detail in the context of such courses, a new generation of slicing-aware developers may enter the industry and demand slicing tools. With time, this could possibly extend to include industrial training courses for practicing developers seeking to upgrade their skill set.

7. Prestigious Success Stories

Do we need to find and publicise cases where slicing has "saved the day"?

Most practitioners in the field only accept tested and tried techniques and methodologies, to minimize risk. Anecdotal evidence advocating one technique over another by established authorities account for a lot in the decision of which tools, techniques and methodologies to select for upcoming projects.

8. A Text Book

Will a definitive text book on slicing help establish slicing as useful and viable tool for developers?

From the previous section, we saw that most of the successes we considered all had one or several good, authorative references that explain central concepts

and techniques. No such book exists for slicing, and this may act as an impediment to adoption, simply because potential slicers have a difficult time in learning the technique. Additionally, basic introductory books such as "The Program Slicing Cookbook" or "Slicing for Dummies" may help get newcomers up to speed quickly.

9. A Discussion Forum

Can gathering the slicing community in a publicly visible discussion forum help attract newcomers?

Most communities have one or several hubs for discussion, be they mailing lists, web forums or newsgroups. When newcomers to slicing try to apply the tools and techniques gleaned from the existing literature, finding experienced assistance is difficult.

10. Widening of Application/Usefulness of Slicing

Should more effort be put into going beyond program slicing, and also slice other artifacts?

A nice demonstration of this was shown by Juergen Rilling, who had applied slicing techniques to program designs and other artifacts in the development process, with the aim of improving the understanding of the system as a whole.

11. Latching on to Established Mainstream Techniques

Can we ride the wave created by another success story?

Agile methods are associated with refactoring and unit testing which are both established by now. Much of the literature on refactoring and unit testing explains how these techniques complement object-orientation, thus creating an association between themselves and one of the most popular programming paradigms. Much of the work on slicing has been in the context of imperative languages, and slicing may therefore have missed the OO-train so far.

Andreas Zeller and his group made the case for riding the popularity of Eclipse by developing slicing tools that integrate well into the Eclipse Tools Platform.

12. Make It Popular to Developers

Should our target group be developers?

The reasoning for convincing developers is as follows: By educating developers on the benefits of slicing, they will demand slicing tools from their tool vendors, and argue the value of such tools to their management.

13. Make It Popular to Managers

Should our target group be managers?

By convincing management of cost savings, or other tangible benefit offered by slicing, it may be possible to attain a "buy-in" at the higher level in organizations, who will then dictate or recommend their staff to employ slicing techniques.

14. Better Tool Implementations

Will providing better tool implementations help make program slicing more widespread?

If we take the old adage *build it and they will come* to heart, the best way forward would be to construct one or several robust implementations of the most promising ideas offered in the slicing literature.

15. A Slicing Challenge

Is holding a grand slicing challenge of slicing a good way for raising awareness of the technique?

The Defence Advanced Research Project Agency (DARPA) in the United States has for the last few years held an annual Grand Challenge for autonomic navigation of robot vehicles. On a less grand scale, there is an annual International Conference on Functional Programming (ICFP) contest for producing the best working implementation for some particular problem. Perhaps a similar kind of challenge, perhaps even yearly, could be beneficial in the process of mainstreaming program slicing.

16. A Slicing Portal [Jens et al 2005]

Would the slicing community be served with a common point of presence on the internet, a portal?

This suggestion was put forth by Jens Knoop and his group at the seminar. The presentation proposed the creation of a web site in the style of `www.program-transformation.org`, only for slicing. Such a web site could contain tools, documentation, online discussion groups, project inventories, references to research people, groups and projects.

4 An Informal Survey

Based on the mainstreaming mechanisms described in the previous section, we conducted an informal survey among the seminar attendees to asses what they thought to be the most effective mechanisms, and which of the mechanisms they themselves were willing to contribute to. Each participant was asked to pick three different mechanisms and rank these in order of importance. Each participant was also asked to mark any and all mechanisms they were willing and felt capable of contributing to.

Each mechanism in Section 3 has a question. These were the questions given to the survey participants. In total, we received 30 responses which went into producing the Figures 1–3.

In Figure 1, we show the distribution of only the first choice, i.e. each person's first choice. From this we see that there is a clear bias towards having better tool implementation and more empirical evidence for the usefulness of slicing. If we weigh the three choices made by each person, we get the result shown in Figure 2. Here, we have assigned a weight of three to the top choice, a weight of two to the second choice and a weight of one to the third choice. We still see the preference for better tool implementations at the top, but now the more general goal of making slicing attractive to developers has made it to second place. Empirical evidence is now rated third. In Figure 3, we see the results from asking what the attendees were willing and able to contribute to. Again, producing tools is a clear first, with education a clear second.

5 Discussion

From the results in Section 4, we can see a marked preference in the community for a bottom-up approach to spreading awareness of slicing. The desired target group in industry is clearly the developer, as is evidenced by the three most highly rated choices, namely improved tools, popularizing slicing to developers and more empirical evidence.

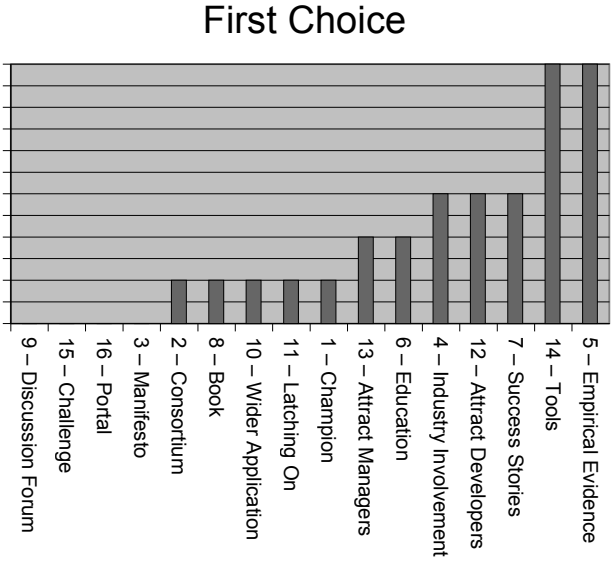

Figure 1: The distribution of the first choice of mechanism.

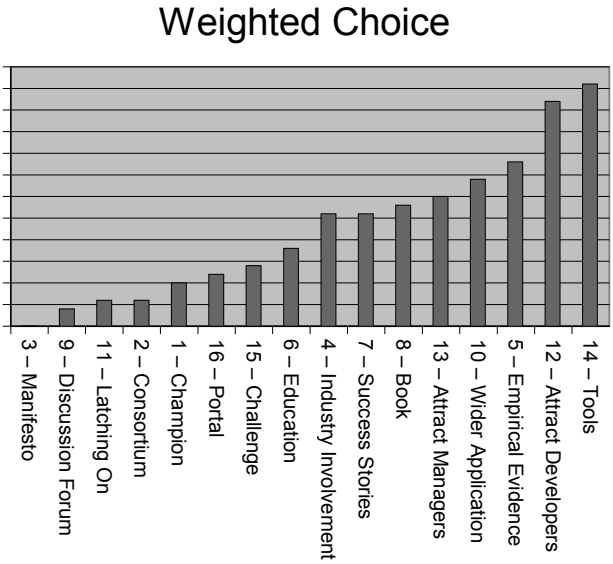

Figure 2: The distribution of a weighted choice of the three most important mechanisms. The first choice has three weights, choice two has two weights and choice three has one weight.

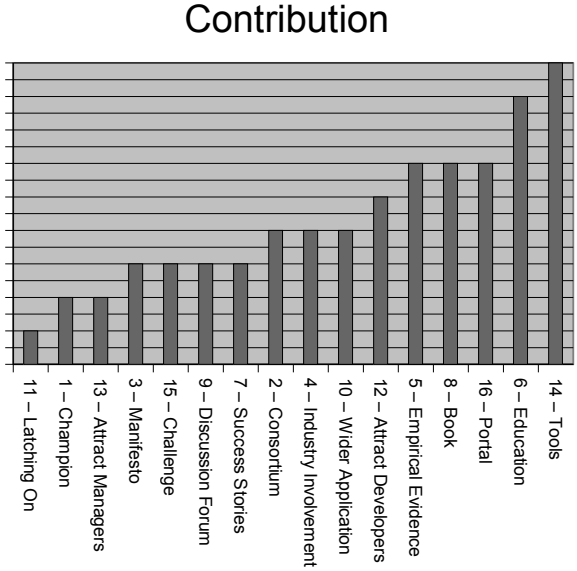

Figure 3: The mechanisms the attendees were most in contribution to.

There is clear correspondence between the first choice in Figure 1 and the weighted choice in Figure 2. Not suprisingly, tools and developers are rated as the most important mechanisms in both cases.

Figure 3 shows the distribution of the answers to the question of participation. The graph reflects the demographic of the attendees, which are all from academia. Contributing to better tool implementations and more empirical evidence are rated highest. A a second group of mechanisms share the third place in these ratings – a text book on slicing, education at college and university level, as well as a slicing portal.

As part of our survey, we added a question where the participants were asked to nominate a slicing champion. The responses to this questions made it clear that no such one person should be nominated.

6 Further Work

The results clearly indicate the direction in which the majority of the community is interested in taking, namely producing slicing tools for software developers.

We would like to point out that the construction of a slicing portal is probably the goal which is easiest to attain in the short term. In fact, Jens Krinke is already committed to searching for hosting facilities for one. Gathering the slicing community under `slicing.org`[1], would go a long way towards a common point of presence for the slicing community and potential slicers.

Another point we would like to make is that while there was little interest in finding a slicing champion, perhaps a slicing working group may be worth considering.

[1] Which is still not registered to anybody at the time of writing.

7 Conclusion

In this report we have presented more than a dozen mechanisms for mainstreaming program slicing. Based on this list, we have conducted a small and informal survey inside the slicing community about which mechanisms are believed to be most fruitful, and which mechanisms the members of the community are most likely to contribute to.

From the survey, we see a clear engineering bias, which favours the construction of tools and techniques targeted at software developers. This bias is evident both in what the participants rate as important and what they are willing to contribute to.

Acknowledgements The authors would like to thank Tibor Gyimóthy for joining our group towards the end of the Dagstuhl seminar. We would also to thank Barry Pekilis for many insightful comments on early drafts of this report.

References

[agi] The agile manifesto. http://agilemanifesto.org. Last visited: 2005-11-30.

[BG96] D Binkley and K Gallagher. A survey of program slicing. *Advances in Computers*, 1996.

[BH04] D. Binkley and M. Harman. A survey of empirical results on program slicing. *Advances in Computers*, 62, 2004.

[CE00] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., 2000.

[cod] Code generation network. http://www.code-generation.net. Last visited: 2005-11-30.

[GSCK05] Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, 2005.

[HBD03] Mark Harman, David Binkley, and Sebastian Danicic. Amorphous program slicing. *J. Syst. Softw.*, 68(1):45–64, 2003.

[KL88] B. Korel and J. Laski. Dynamic program slicing. *Inf. Process. Lett.*, 29(3):155–163, 1988.

[Tip94] Frank Tip. A survey of program slicing techniques. Technical report, Amsterdam, The Netherlands, 1994.