BUGS workshop 2005 Position Statement:

Issues in Deploying SW Defect Detection tools

David R. Cok
Kodak R&D Laboratories, Eastman Kodak Company
31 March 2005

In any technology, there is a large gap between a research prototype and a working, useful product; this is no less true with tools that embody complex, novel software technology. The various software defect detection tools available today, typically as some variation of open source software, exemplify this. The most successful are backed by a sizable research group with long-term research objectives that use the tool as a platform; the deploying of the tool requires considerable investment of time in careful coding, testing, documentation, and release procedures that does not necessarily benefit the academic research enterprise. Nevertheless, groups are often sufficiently passionate about the technology to invest the effort; however, this alone does not generate user demand for a product.

From an industrial software developer's point of view, a new tool must pass muster in two key user experience dimensions in order to be successful: usefulness and usability; furthermore a software development organization must have confidence in the tool supplier's longevity in order to adopt the tool.

Usefulness is a measure of the added value the tool supplies a user. It is not necessarily a measure of breadth of the tool, although tools such as IDEs that are meant to be working environments must be relatively complete. Indeed a tool that has all-encompassing claims may well generate suspicion rather than curiosity. Rather a defect detection tool must have these features:

It must be reasonably accurate in its output. From the user's perspective, the issue is not so much one of technical criteria like soundness and completeness; instead, the user's measure will be the fraction of false positives present and false negatives missed in the total output. False positives, in which a tool claims a code fragment to be a defect when the code is in fact valid, are noise to the user. The user must still assess (with the potential for error) each defect report and decide whether or not it is valid. Too high a fraction of useless reports will cause the user to see less benefit in the results of the tool overall. On the other hand, false negatives are defects that the tool fails to detect. The impact of false negatives can be mitigated if it can be clearly stated what types of defects the tool expects to address. False negatives within the domain of the tool, however, will reduce the trust the user has in the tool overall, since the user will need to invest duplicate manual effort in finding those errors.

Secondly*, the tool must address the bulk of the important defect areas for the user*. Importance can be measured in terms of user effort: important defect classes consume large fractions of the user's development and debugging time, because of the combination of the defect's frequency and the average effort to diagnose and correct each defect. Experimental case studies of actual programmer experience on large scale projects would be useful to determine which defect classes are the most problematic in the overall software engineering effort on typical projects.

And third, it must have time and space *performance and stability consistent with the benefit provided*.

Usability, or ease-of-use, describes the ease with which a user, particularly a novice user, can obtain useful results from a tool. There are a number of components to keep in mind:
- Out-of-the-box experience: how easy is it to install, configure and obtain results on a user example the very first time? Is the tool self-contained, or does one need to download and configure other tools first?
- Resource investment profile: how much time, money and effort must a user invest in order to obtain a small amount of initial benefit or invest incrementally to obtain increasing benefit? Can results be obtained on small portions of a large project? Can the tool be layered alongside of the user's existing working environment?
- Perspicuity: how clear are the reports provided by the tool? Do they actually save work? Can false positives, once evaluated, be annotated so that those false reports are not regenerated on subsequent invocations of the tool and the work of repeated defect report assessment avoided.

Finally, there is increasing willingness to adopt academic and open source tools that do not become part of a final commercial product. However, a software development organization has a need for **confidence** that the tool supplier will continue to support, correct, improve, and release the tool. This typically weighs against academic and open source tool suppliers and in favor of commercial tool vendors, even in the face of license fees, even though license fees place a considerable hurdle in the path of informal initial trial.