1 Introduction

Quality has been an elusive goal throughout the software industry. Despite large amounts spent on program design, development, testing, and maintenance, software has never reached the quality levels that are expected of computer hardware and many other manufactured products. Consequently, software defects are common and tolerated, regardless of the damage they may cause. The common assumption is that there simply is no cost-effective way to increase quality.

1.1 Inspection: The Key to Quality

The software engineering community has long known that software inspection is the key to lowering defect rates at minimal, short-term cost and with substantial long-term benefits. Inspection succeeds because it detects and removes errors early in the process, before the code reaches testing or deployment.

Nonetheless, thorough inspection is rarely practiced, and there are numerous barriers to effective implementation. For instance, inspection is labor intensive, tool support is minimal, and inspection is often regarded as one more bureaucratic barrier and delay.

1.2 The Move to Automated Inspection Tools

Powerful multi-language code inspection tools are now emerging. These tools can quickly locate many common programming faults, and these faults are exactly the ones that cause some of the most damaging defects. The best inspection tools are evolutions of proven Year 2000 Independent Verification and Validation (IV&V) tools, which have been invaluable in locating serious date bugs in code that was assumed to be Y2K compliant. The Y2K experience has decisively demonstrated the value, and even the requirement, of automated inspection. The challenge is to extend the successful Y2K techniques so as to detect other types of defects.

Automated code inspection can now be an integral, inexpensive part of the software process throughout the life cycle, achieving much higher quality levels and producing meaningful quality metrics. Software can be quickly inspected during development, testing, and maintenance after every significant change, or...
even with every compilation. Numerous defects will be detected and repaired immediately, adding efficiencies and quality throughout the software life cycle.

This paper will review inspection benefits, discuss the defects that automated tools can detect, describe the characteristics of the best tools, and show the significant returns from automated inspection.

2 Inspection, Quality, and Software Best Practices

The best software quality can be achieved by combining: 1) design inspection, 2) code inspection, 3) quality assurance, and 4) testing. Incorporating these four elements into a well-managed software project can yield the very highest quality as measured by the “defect removal rate” (DRR). The United States median DRR for all software is only 85%, whereas these approaches, used throughout the life cycle, have been shown to produce a median DRR of 99%. Put another way, 14 out of 15 defects can be eliminated before they are released as a part of production code, eliminating significant maintenance and failure costs.

Best practices that include inspection not only reduce defects during the development phase, but they also reduce bad fix injection problems during maintenance. Effective inspection can even enable “clean room management” so that all software changes adhere to verifiable quality standards.

2.1 Code Inspection Practices and Limitations

Code inspection is an old idea, going back at least to Fagan’s 1976 paper [3]. Since that time, any number of experiments and case studies with a wide variety of methodologies, mostly manual and labor intensive, have demonstrated how well inspection can work. An extensive bibliography can be found in [8].

Code inspection, as normally practiced, is a labor-intensive activity, often involving formal code reviews, structured walk-throughs, and similar techniques. The assumption has always been that the defects should be detected by programmers who should be carefully examining the code, designs, and documentation—inspection is often performed in a formal setting. While the results can be impressive, inspection may not be performed well or at all. Management often sees inspection as a resource and time drain, and programmers can feel constrained by the process formality. The sheer volume of the code involved is another constraint, as modern programs and systems often involve hundreds of thousands, or millions, of lines of source code.

---

1 This summary and the supporting data are from Capers Jones [9]. Jones has accumulated and analyzed software metrics from numerous projects.

2 The DRR is determined as a percentage of the total known defects in a software component at a point in time, where the defects are counted over the component’s life.