# Reducing Software Vulnerabilities – The Number One Goal for Every Software Development Organization, Team, and Individual

*Barti Perini, Stephen Shook, and Girish Seshagiri*
Ishpi Information Technologies, Inc.

## 1.0 Introduction

In this paper, we describe the measures that our software application teams collect, analyze, and report. These measures are proven to help them develop and deliver software with little or no vulnerabilities. For more than 15 years, the Ishpi Information Technologies, Inc. (DBA *ISHPI*) Advanced Information Services Division (*ISHPI* AIS) has consistently delivered software with an average of less than 0.27 defects per thousand lines of code (KLOC) across systems ranging from medium to large size of up to several hundred KLOCs. Using the estimate that 5% of defects result in vulnerabilities, *ISHPI* AIS software teams consistently deliver "sustainably secure software" with approximately 1 vulnerability in 100 KLOC. *ISHPI*'s business model is based on agile firm fixed price deliveries with a lifetime warranty against software defects found in production use. [1] [2]

## 2.0 Background

### 2.1 Principles Driving *ISHPI* Measures and Metrics

In our experience, starting with metrics does not produce favorable results. Instead, we started with the goals and then used the Goal-Question-Metric (GQM) paradigm to derive the metrics that would help us achieve the goals [3]. In 1992, *ISHPI* AIS established a business goal to "Improve profitability and customer satisfaction by delivering substantially defect-free products on predictable cost and schedule". It became apparent that, to predictably deliver substantially defect-free software, we needed to continually improve the quality of the development process.

Which leads to our first principle:

**Quality and reducing vulnerabilities should be the number one goal for every software team**

It is well known that some software defects result in vulnerabilities. To reduce vulnerabilities, software teams need to be focused on defect injection and removal throughout the software lifecycle. Teams need to be aware of where the defects are injected and where they are removed. Teams cannot rely on testing alone to find and remove software defects, including security vulnerabilities. Instead, teams need to focus on identifying and removing defects as early in the lifecycle as possible. Our experience has shown that many of the Common Weakness Enumerations (CWEs), such as buffer overflows, cross-site scripting, and failures to validate input values can be easily found and corrected in personal reviews and peer reviews of the source code. Improving quality by not relying on testing alone addresses many of the software security issues. [4]

Which leads to our second principle:

**Software teams should put the highest quality code into test**

Large systems are built by integrating small components developed by individual developers. If the components are of poor quality, the delivered product will have a large number of defects and vulnerabilities. Teams need to make sure individual developers take pride and responsibility for the quality of their work products. The developers should be trained in and supported by an agile disciplined personal development process that should enable collection, analysis, and reporting of defect injection and removal data through code complete. The developers need to make sure that the components are defect-free at completion of component development. This will make more time available to deal with defects and vulnerabilities in technology and system level stacks, which in turn significantly reduces total

development time. Teams should include defect data along with cost and schedule data in weekly status reports to development and customer management. The data collected needs to be both *precise and accurate*. Team members need to be trained in data collection and analysis so that they can effectively use these metrics to be self-managed and take timely action to meet personal and team goals.

Which leads us to the next principle:

**Unless data is collected and used by trained team members it won't be useful**

Software teams are often working to meet arbitrary and unrealistic delivery commitments imposed on them. Team members need to be trained in estimating sizes of software components and negotiate aggressive and realistic delivery commitments based on personal and team historical data that shows the relationship to size and effort. Data that clearly show the impact of disciplined practices, such as personal reviews and peer reviews in reducing rework and total development time, help the team and individual developers to make responsible delivery commitments.

Software teams are expected to make frequent deliveries of working software. Imposing time-consuming data collection of questionable value is counter-productive. *ISHPI* AIS software teams collect only size, task time, defect, and task completion data as shown in Table 1.

**Table 1. Base Team Measures**

| Measure | Unit |
|---|---|
| Size: | Lines of code |
| Task time: | Minutes |
| Defects: | Number |
| Task completion: | Yes/No |

In order for the resulting data to be precise and accurate, teams need to have a clear definition of the measures. For example:

- Size measurements need to be based on defined coding standards and line-of-code counting rules.
- Tasks need to be granular, and measurements for each task should track only time spent on it, excluding interruptions.
- Defects need to have standardized defect types based on orthogonal defect classifications.
- Task completion needs to be based on an agreed "definition of done" at the task level.

From the precise and accurate base measures, the "vital few" performance metrics are derived to help self-managed teams consistently deliver substantially defect-free software on predictable cost and schedule. In turn, management is assured that the organization's business model of firm fixed price with guaranteed quality, results in customer satisfaction and company profitability.

Which leads to the final principle:

**Software teams must be focused on the "vital few" metrics and not the "trivial many"**

## 2.2 Summary of Management Practices for Producing Secure, Defect-Free Software

Our experience has shown that jelled, self-directed teams consistently collect, analyze, and report the "vital few" data and deliver software that is substantially free of defects and vulnerabilities on predictable cost and schedule. Management bears the responsibility to:

- Staff projects with team members who are formally trained in estimating, planning and tracking, and measuring and managing quality.
- Start each project right with a cohesive team having a shared vision that makes disciplined commitments with the assistance of a coach.

- Support the teams in collecting, analyzing, and reporting product and process data—size, effort, schedule, and quality (defects injected and removed at each step in the process)—providing them the data needed to make decisions, and improving continuously through causal analysis and postmortems.
- Ensure team members are trained to conduct peer reviews of all design and code artifacts in order to put the highest-quality code components into test, striving for 90 – 100% defect-free components with zero cybersecurity vulnerabilities. Ensure the review checklists incorporate specific checks for the types of vulnerabilities identified within the "Open Web Application Security Project (OWASP) Top Ten and Common Weakness Enumeration (CWE/SANS) Top 25 Most Dangerous Programming Errors".
- Ensure independent penetration testing is performed.
- Require teams to report status weekly with precision and accuracy, monitoring planned versus actual numbers of defects throughout the software development lifecycle, enabling the team to proactively manage the quality.

## 3.0 Specific Measures and Metrics to Improve Quality and Reduce Vulnerabilities

We discuss the vital few performance metrics that really matter and help software teams manage the software work by managing quality. Table 2 shows data collected for the vital performance metrics at the component, increment, and project level. The data from projects across the organization are used to derive *ISHPI* AIS organizational performance metrics as shown in Table 4 ("Results and Objective Evidence" section below). Table 3 contains the operational definitions for these metrics.

These metrics help individuals at the component level, the team at the increment and project level, and the process group at the organization level to guide the course of action towards the achievement of the goals, or to help to evaluate the result of the actions.

The metrics identified as leading indicators in Table 2 are proactively monitored to provide an early indication of whether the strategy is being implemented successfully to build a secure, quality product. Corrective actions are taken as needed based on these leading indicators. These corrective actions have a significant impact on the lagging indicator metrics. The lagging indicators measure the results of the practices used by the individual, team, and organization. Lessons learned from both positive and negative outcomes are analyzed and used for continuous improvement of the individual, team, and the organization. [5] [6]

**Table 2. Vital Few Performance Metrics Tracked**

| Performance Metrics | Leading Indicator? | Lagging Indicator? | Organization | Project | Increment | Component |
|---|---|---|---|---|---|---|
| Planned vs. actual size | Y | | ✓ | ✓ | ✓ | ✓ |
| Planned vs. actual effort | Y | | ✓ | ✓ | ✓ | ✓ |
| Planned vs. actual schedule | Y | | ✓ | ✓ | ✓ | ✓ |
| Planned vs. actual earned value | Y | | | | ✓ | |
| Planned vs. actual defect profile (e.g., personal review yield, peer review yield, defect densities, etc.) | Y | Y | ✓ | ✓ | ✓ | ✓ |
| Total Cost of Quality (COQ) | Y | Y | ✓ | ✓ | ✓ | |
| First Time Right (in acceptance test): number of changes with no acceptance test defects | | Y | ✓ | ✓ | ✓ | |
| Acceptance test defect density in delivered code | | Y | ✓ | ✓ | ✓ | |

**Table 3. Operational Definitions**

| Term | Definition |
|---|---|
| % schedule deviation | Actual schedule divided by planned schedule (by months) times 100 minus 100 |
| Acceptance test defect | Defect found during the acceptance test of the product. This defect may be identified by the customer as well as the *ISHPI* team. Acceptance test period starts after the completion of system test of the product and ends when the code is deployed to production. |
| Cost of Quality (COQ) | COQ = (effort in appraisal tasks + effort in prevention tasks + effort in failure tasks) / Total effort towards the commitment (including project management) x 100<br><br>Appraisal tasks: Personal reviews, peer reviews, and first-time test execution<br>Prevention tasks: Training, postmortems, and causal analysis<br>Failure tasks: Analyzing and fixing defects found in reviews and testing |
| Defect density | The number of defects identified in a product divided by the size of the product component, expressed in standard measurement terms for that product (e.g., 1000 lines of new and changed code) |
| Earned value | Percentage of effort for tasks completed divided by total effort of the tasks |
| First Time Right (in acceptance test) | Deployed software changes, accepted by the customer the first time, with no further rework |
| Peer review yield | The percentage of the total defects that are found and removed in peer review |
| Personal review yield | The percentage of the total defects that are found and removed in personal review |

### 3.1 Decision Process Based on these Measures and Metrics

*ISHPI* AIS teams use measures and metrics in a multitude of ways to manage their work and meet project goals and customer objectives. Secure software that is free from vulnerabilities starts with responsible team commitments—whether for an individual software component, a sprint, or a longer-term timeline. When teams are under pressure to meet what turns out to be an impossible commitment, they often fall victim to taking short-cuts, which degrade quality, create technical debt and rework, and open the door to security vulnerabilities. When team members don't understand the relationship between time and size, it is hard to make commitments they can fulfill. Therefore, individuals and teams need to keep a historical record of their estimates and actuals, enabling them to make aggressive but realistic commitments for future efforts.

The data is not reviewed in isolation—it is always used in combination, and with a focus on the bigger picture. Data falling outside the expected range is not necessarily a cause for alarm: Such data is taken as a trigger to dig deeper, perform more analysis to understand the circumstances, and make decisions that mitigate risks and address issues.

**Planned versus Actual Size, Planned versus Actual Effort, and Planned versus Actual Schedule:**

During component postmortems, team status meetings, and increment postmortems, the team reviews the size, effort, and schedule deviation data. Based on the data, the team makes a judgement about need for corrective action. If deviations are significant, then the team assesses the causes and potential impact on schedule commitments for the committed scope. If correction action is required, some possible actions include renegotiating the schedule, deferring functionality, and adding staff.

The team also reviews size, effort, and schedule deviation data during project postmortems. The team identifies lessons learned based on the causes of the deviation and submits improvement suggestions to be applied to future increments and projects.

**Planned vs. Actual Earned Value:**

During team status meetings, the team reviews earned value deviation data and makes a judgement about the need for corrective action. If deviations are significant, then the team assesses the causes (e.g., quality issues) and potential impact on schedule commitments for the committed scope using earned value projections and "what if" scenarios enabled by our process performance model. If correction action is required, some possible actions include strengthening peer review practices, renegotiating the schedule, deferring functionality, and adding staff.

**Planned vs. Actual Defect Profile:**

During team status meetings, the team reviews the defect profile for components that have completed unit test. In particular, the team reviews for component design and code: personal review yields, peer review yields, personal review defect densities, peer review defect densities, and unit test defect densities. In addition, the team reviews the projected defect counts and defect densities for downstream steps (e.g., entering system test, entering acceptance test), estimated by the quality projection model using the project performance to-date. The team uses the data to determine the need for corrective action. If the team determines that corrective action is required, some possible actions include:

1. Strengthening peer review practices
2. Conducting another peer review (perhaps with a different or additional reviewers)
3. Re-reviewing unit test cases
4. Performing additional unit testing

Before each increment delivery for acceptance test, the team reviews the integration and system test results. The team uses the data to confirm that the product is ready to be delivered to the customer. If team analysis of the data reveals warning signs, the team may identify single components that seem to be particularly worrisome, initiate additional rounds of testing, or conduct targeted peer reviews to proactively address quality issues before they reach the customer.

During increment and project postmortems, the team reviews the defect profile for the completed products. The team identifies lessons learned based on the causes of quality issues (as well as quality successes) and submits improvement suggestions to be applied to future increments and projects.

**Total Cost of Quality (COQ):**

During team status meetings and postmortems, the team reviews the planned, actual, and projected COQ tables and charts: appraisal + prevention, failure, and total. After reviewing the data, the team makes a judgment as to whether the project is on course to be completed with a COQ that is too high and thus requiring corrective actions. If the team determines that corrective action is required, some possible actions include:

1. Proactively monitoring and analyzing the factors that affect the appraisal + prevention, and failure COQ. These factors include:
    a. For appraisal COQ: Slow review rates, slow test execution, and poor product quality (poor quality product slows down the review process)
    b. For prevention COQ: Training tasks to address new technologies, training new team members, and time spent in causal analyses and postmortems
    c. For failure COQ: Depends on the number of defects found in review and test activities. *ISHPI* AIS data show that there is a reasonable correlation between failure costs and the defect injection rate (total defects found through component/document development divided by the development effort)
2. Conducting causal analysis of selected failure outcomes and identifying action items
3. Re-evaluating after corrective actions have been put into place

**First Time Right (in Acceptance Test):**

During increment and project postmortems, the team reviews the number of features or changes that were incorporated into the product "first-time-right". The team analyzes whether the outcome was within the expected range, and the team identifies lessons learned and submits improvement suggestions to be applied to future increments and projects for both positive and negative outcomes.

**Acceptance Test Defect Density in Delivered Code**

During increment and project postmortems, the team reviews acceptance test defect density. The team identifies lessons learned based on the causes of quality issues (as well as quality successes) and submits improvement suggestions to be applied to future increments and projects. The team performs formal root cause analysis on any individual defects (or related groups of defects) that escaped to the customer in order to identify specific improvements that would have prevented the defect from escaping.

## 4.0 Results and Objective Evidence

In industry, a very high percentage of cyber-attacks (more than 90% in some studies) are due to hackers exploiting application software defects, and a majority of software applications do not pass even rudimentary tests of known vulnerabilities [7].

By comparison, our metrics and approach were proven successful on several contracts with the Government. An example is the modernization of a system containing personally identifiable information for a federal agency that one of our teams delivered ahead of schedule with extremely high quality and met stringent security standards. During independent penetration testing conducted before and after delivery to the customer, no cybersecurity vulnerabilities were found in the code base of more than 700,000 lines of code. The system has had zero down-time due to software code defects and vulnerabilities since production release in September 2011. Annual FISMA audits by the government report zero non-compliance issues. $(ISC)^2$ recognized the *ISHPI* AIS contribution by awarding the Government Industry Security Leadership Award (GISLA).

This project is one of five projects included in an SEI Technical Report focusing on specific security- and safety-critical outcomes. The report includes a discussion of how these projects could provide potential benchmarks for ranges of quality performance metrics (e.g., defect injection rates, removal rates, and test yields) that establish a context for determining very high quality products and predicting safety and security outcomes. [8]

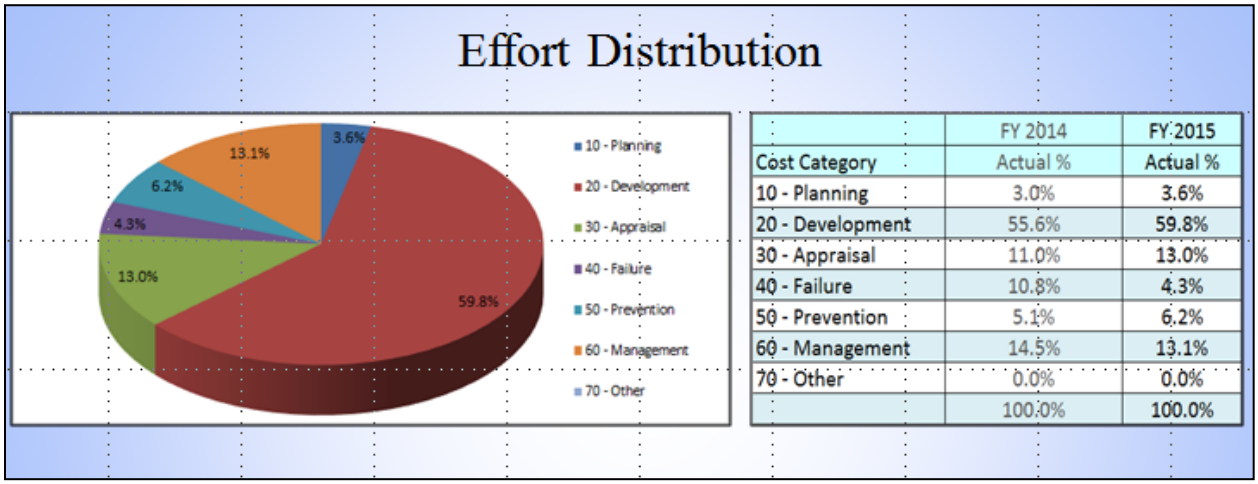
During the past 15 years, our teams have achieved results far superior to generally acknowledged industry averages for schedule, cost, and quality performance, as indicated in Table 4. *ISHPI* AIS results for schedule deviation is now at 3.3%, significantly below the generally-accepted industry average for schedule deviation, which is in excess of 50%. Our defect density in the delivered products is less than 0.27 defects/KLOC, which extrapolates to approximately 1 vulnerability/100 KLOC. Our Total Cost of Quality is less than 34%, compared to the industry average of over 50% Total Cost of Quality, with a failure cost of less than 10%.

**Table 4. Industry, Best-In-Class, and *ISHPI* AIS Average Performance Metrics**

| Performance Metrics | Industry Average | Best in Class Average | *ISHPI* Average |
|---|---|---|---|
| % Schedule deviation [9], [10], [11] | > 50% | 6% | < 4% |
| % of defects removed prior to system test [12], [14] | < 60% | 70 – 95% | > 85% |
| % of development time in rework fixing system test defects [13], [15] | > 33% | 2.7 – 10% | < 10% |
| Acceptance test defect density in delivered code (defects per 100,000 Source Lines of Code) [12] | > 100 | 55 | < 27 |
| Cost of Quality [16] | > 50% | 29% | < 34% |
| % of components with zero post-unit test defects | Data not available | Data not available | > 60% |
| % of deployed features or changes that were incorporated into the product "first-time-right" | Data not available | Data not available | > 94% |

In industry, costs related to cyber security are rising exponentially. Consequently, more than 50% of operation and maintenance (O&M) spend now goes to corrective maintenance (fixing bugs). Current practices rely on testing as the principal defect removal method and, as a result, customers spend months in acceptance testing, even for modest sized software products.

By comparison, Figure 1 shows the typical effort distribution of two recent *ISHPI* AIS maintenance contracts performed during Fiscal Year (FY) 2014 and FY 2015. This figure demonstrates that over 50% of the effort was used towards product development, compared to less than 5% used towards failure. This results in significantly lower costs for maintenance. Our operations and maintenance (O&M) cost is 5% of the overall cost of product development. Table 5 shows that, of the effort spent for change requests, more than 70% was spent on adaptive and perfective changes, compared to less than 30% spent on corrective-type changes.




| | FY 2014 | FY 2015 |
|---|---|---|
| Cost Category | Actual % | Actual % |
| 10 - Planning | 3.0% | 3.6% |
| 20 - Development | 55.6% | 59.8% |
| 30 - Appraisal | 11.0% | 13.0% |
| 40 - Failure | 10.8% | 4.3% |
| 50 - Prevention | 5.1% | 6.2% |
| 60 - Management | 14.5% | 13.1% |
| 70 - Other | 0.0% | 0.0% |
| | 100.0% | 100.0% |

**Figure 1. Effort Distribution for Operations and Maintenance Contracts**

**Table 5. Effort by Type of Change Request**

| Type of Change Request | % of Effort |
|---|---|
| Corrective | 29.54% |
| Adaptive and Perfective | 70.46% |

## 5.0 Conclusion

We have presented key measures and metrics that our software teams collect, analyze, and report. They are based on sound principles borne out by industry experience: that quality and reducing vulnerabilities should be the number one goal for every software team; that teams should put the highest quality code into test; that the data must be collected and used by trained team members; and that teams must be focused on the "vital few" metrics. These measures and metrics are a key element of the overall set of software practices that *ISHPI* AIS has developed and refined over the past 24 years and used with proven best-in-class results to produce and deliver substantially defect-free software with very few security vulnerabilities.

## 6.0 References

1. Shull, Forrest. "A Lifetime Guarantee." *IEEE Software* 30.6 (2013): 4-8. Print.
2. Software Engineering Institute. *2012 Year In Review*. Rep. Software Engineering Institute, 2012. Web. 19 July 2016. <http://www.sei.cmu.edu/library/assets/annualreports/2012_YIR.pdf>.
3. Basili, Victor, Gianluigi Caldiera, and H. Dieter Rombach. *The Goal Question Metric Approach*. White Paper. N.p.: n.p., 1994. University of Maryland. Web. 19 July 2016. <http://www.cs.umd.edu/~mvz/handouts/gqm.pdf>.
4. Seshagiri, Girish. "'If It Passes Test, It Must Be OK', Common Misconceptions and The Immutable Laws of Software." *CrossTalk* May/June (2014): n. pag. *CrossTalk*. U.S. Department of Defense, May-June 2014. Web. 19 July 2016. <http://www.crosstalkonline.org/storage/issue-archives/2014/201405/201405-Seshagiri.pdf>.
5. Seshagiri, Girish. "If We Eliminate the Monthly Status Report, What Do We Replace It With?" Agile in Government. 26 Feb. 2015.
6. Seshagiri, Girish. "Performance Metrics That Matter: Eliminating Surprises in Agile Projects." Software Solutions Conference 2015. 16-18 Nov. 2015.
7. Jarzombek, Joe. "How to Write Risk Management and Cyber Resilience Requirements into Contracts." CISQ Cyber Resilience Summit. Reston, VA. 15 Mar. 2016.
8. Woody, Carol, PhD, Robert Ellison, PhD, and William Nichols, PhD. *Predicting Software Assurance Using Quality and Reliability Measures*. Rep. no. CMU/SEI-2014-TN-026. Software Engineering Institute, Dec. 2014. Web. 19 July 2016. <http://resources.sei.cmu.edu/asset_files/technicalnote/2014_004_001_428597.pdf>.
9. *Chaos Manifesto 2013*. Rep. The Standish Group International, 2013. Web.
10. Davis, Noopur, and Julia Mullaney. *The Team Software Process (TSP) in Practice: A Summary of Recent Results*. Rep. no. CMU/SEI-2003-TR-014. Software Engineering Institute, Sept. 2003. Web. 19 July 2016. <http://www.sei.cmu.edu/reports/03tr014.pdf>.
11. McAndrews, Donald R. *The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices*. Rep. no. CMU/SEI-2000-TR-015. Software Engineering Institute, Nov. 2000. Web. 19 July 2016. <http://www.sei.cmu.edu/reports/00tr015.pdf>.
12. Jones, Capers. "Achieving Software Excellence." CrossTalk Jul/Aug (2014): n. pag. CrossTalk. U.S. Department of Defense, July-Aug. 2014. Web. 19 July 2016. <http://www.crosstalkonline.org/storage/issue-archives/2014/201407/201407-Jones.pdf>.
13. Humphrey, Watts S. *Winning with Software: An Executive Strategy*. Boston, MA: Addison-Wesley, 2002. Print.

14. Jones, Capers. “Software Quality in 2013: A Survey of the State of the Art.” *Namcook Analytics*. Namcook Analytics LLC, 18 Aug. 2013. Web. 23 Feb. 2016. <http://namcookanalytics.com/wp-content/uploads/2013/10/SQA2013Long.pdf>.
15. McHale, James. “A Brief Survey of the Team Software Process.” *SEI Technologies Forum*. Software Engineering Institute, 24 Oct. 2011. Web. 19 July 2016. <http://www.sei.cmu.edu/webinars/view_webinar.cfm?webinarid=18744>.
16. Jones, Capers. “Evaluating Agile and Scrum with Other Software Methodologies.” *InfoQ*. C4Media, 20 Mar. 2013. Web. 19 July 2016. <http://www.infoq.com/articles/evaluating-agile-software-methodologies>.