Fully Employing Software Inspections Data

Forrest Shull[1,*], Raimund L. Feldmann[1], Carolyn Seaman[1,3], Myrna Regardie[1], Sally Godfrey[2]

[1] *Fraunhofer Center for Experimental Software Engineering Maryland, College Park, MD 20740*

2 NASA Goddard Space Flight Center, Greenbelt, MD 20771

3 University of Maryland Baltimore County, Baltimore, MD 21250

[*] corresponding author: fshull@fc-md.umd.edu / phone: +1-240-487-2904 / fax: +1-240-487-2960

Abstract

Software inspections provide a proven approach to quality assurance for software products of all kinds, including requirements, design, code, test plans, among others. Common to all inspections is the aim of finding and fixing defects as early as possible, and thereby providing cost savings by minimizing the amount of rework necessary later in the lifecycle. Measurement data, such as the number and type of found defects and the effort spent by the inspection team, provide not only direct feedback about the software product to the project team, but are also valuable for process improvement activities.

In this paper, we discuss NASA's use of software inspections and the rich set of data that has resulted. In particular, we present results from analysis of inspection data that illustrate the benefits of fully utilizing that data for process improvement at several levels. Examining such data across multiple inspections or projects allows team members to monitor and trigger cross project improvements. Such improvements may focus on the software development processes of the whole organization as well as improvements to the applied inspection process itself.

Keywords

Formal software inspections, measurement and analysis, process monitoring and improvement, experience-based approach, management by data, NPR 7150.2

1 Introduction and Motivation

A long history of experience and experimentation has produced a significant body of knowledge concerning the proven effectiveness of software inspections. A substantial portion of this body of knowledge has been gained at NASA Centers or on NASA systems (e.g. [11], [12], [21]).

As a result, software inspections have become an integral part of today's Verification and Validation (V&V) activities for software development projects. The underlying technology is well established, and incorporated into the standard software development procedures of many organizations. For example, software inspections are included in the mandatory NASA Procedural Requirements for Software Engineering (NPR 7150.2), issued by the Office of the Chief Engineer [14].

Despite this successful record, experiences gathered from conducting training courses and hosting technical exchanges have shown us that some teams still have problems applying inspections in practice. Although metrics collection and analysis need not be heavyweight to provide useful levels of insight, development teams are often running hard to meet the next technical deliverable and find it hard to get time for such activities. It can even sometimes be difficult for projects to get the resources to keep a sufficient number of inspections in place at all, es-

pecially when developers see them as disconnected from their normal day-to-day development activities. Our research work with NASA has focused on addressing these difficulties while providing better results (in terms of defect detection effectiveness and efficiency) for the effort spent on inspections.

An important factor in obtaining better results is the use of information gained from an inspection to better understand the outcomes of the inspection; to gain insight into what is working well or could be improved with the inspection process itself; and to provide information that can be used to make decisions about where to best allocate effort on projects. In this paper, using data from NASA projects, we illustrate the use of inspection metrics for improvement activities at each of these levels. Moreover, we show that even relatively light metrics collection can help achieve these goals, if a mechanism exists for comparing the parameters of a given inspection against existing models or guidelines. We will also consider tool support that makes these activities more feasible for projects.

1.1 The software inspection process

Data and experience from many years and many types of organizations have shown that a properly conducted inspection can remove between 60% and 90% of existing defects [23]. Other key benefits of using inspections for reviewing software products are:

- A well documented and technically sound approach
- Improvement in quality by identifying faults and omissions
- Cost savings through fault detection in early phases and before delivery
- Avoiding repetition of common defects by clearly defining them and educating team members
- Improved team communication

Figure 1 illustrates an overall process model for software inspections. Although many variants exist, the phases in this model have been defined based on years of practical experience. This model incorporates process recommendations made by seminal works on software inspection ([7], [8]). Although different teams have some leeway to tailor the parameters of how they implement each step, in training courses with NASA teams we have found it helps to have explicit steps that remind the team to:

- Select an appropriate excerpt from the technical documentation to be reviewed; select the appropriate team and mix of expertise to conduct the review; plan and schedule resources so that the process can be conducted; abort inspections of documents that are not ready, to avoid spending effort without commensurate benefit. (Planning)
- Provide needed background and/or technical information to the team members, if needed, to ensure that all reviewers have the information they need to conduct a useful inspection. (Overview)
- Allow time for individual reviewers to read and become familiar with the technical product under inspection, and begin finding issues that may need to be reworked. (Preparation)
- Review the document together and come to consensus, as a team, on the list of issues that should be improved. (Inspection Meeting)
- Optionally, allocate time to discuss *how* to fix those issues or investigate outside the meeting whether certain issues require fixing. (Third Hour)
- Allow time for the rework to be completed by the author. (Rework)
- Explicitly close out the issues by verifying that they have been fixed and none have been missed – to ensure that the time spent on the inspection has not been wasted. (Follow-up)

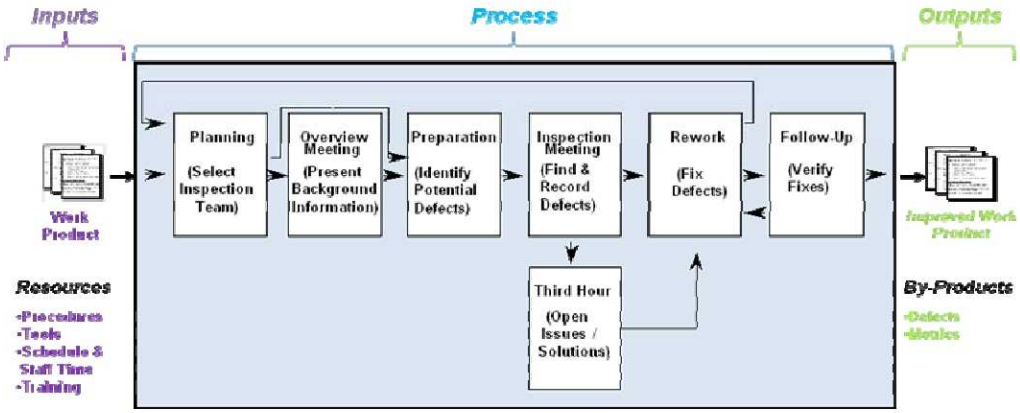


Figure 1: Software Inspection Process

Looked at as a whole, such a process requires not only a work product to inspect, but also a procedure that keeps the team on track, time to be allocated in the schedule for the team to perform the inspection, training (in the sense that team members need to have agreement on what is in- and out-of-scope for the inspection, and a shared sense of how to productively investigate these issues). As an output, if the process is followed an improved document is created. However, teams sometimes fail to appreciate the benefits to be gained from examination of other information that exists as a by-product of the inspection process: the defects themselves, and metrics that describe aspects such as how much effort was required to find those defects, how they were distributed, or how many inspectors were involved. These data can be central to improvement efforts, as they shed insight not only on the inspection process itself, but also on the overall health (defect density) of the development work.

1.2 Why improve?

Despite the numerous and well-documented benefits it can be difficult to implement effective software inspections in projects. There is a learning curve involved, meaning that it takes some time for developers to understand how to effectively find defects on their own in software work products. Inspections can be *perceived* as a heavyweight process that does not address the real issues of concern to the team. And, for moderators who are trying to set up inspection processes and get people to take part, there is little guidance on what types of people are important to get involved and how much direction to give them.

However, while the process structure in Figure 1 has been shown to suit just about any kind of inspection, there are many variations in how the individual steps can be instantiated, and the wide range of software being developed across all of the NASA Centers poses an interesting challenge regarding how to provide for each team a detailed and effective process. A feasible solution has been to provide generic inspection processes that yield good results when first applied, but that can be tailored or improved based on the constraints of each project. Monitoring the practice once it is in place helps address the learning curve by helping the team understand where effort is being spent and whether the return is worth the investment. This feedback can help the team focus on how to improve their implementation or the process itself to be more effective. Monitoring can also help provide better guidance to inspectors about which problem areas to focus on during the review.

Figure 2: Primary areas for improvement of software inspections

For example, previous studies, run both by ourselves [1] and by independent researchers [[2], [5], [13], [16]], had shown that an inspection approach known as Perspective-Based Inspections (PBI) resulted in improved defect detection effectiveness. PBI does not apply "out of the box"; there are many choices that need to be made about which sets of technical expertise to include, which quality issues to focus on, and how much detailed guidance to provide, that must be made for each project. However, PBI does provide a general strategy that has proven effective and a mechanism for how that strategy can be better tailored and improved for each team's specific needs.

The ability to improve the inspection approach in context was demonstrated in work funded by the NASA Technology Infusion program, in which we worked with teams across several NASA Centers to tailor, implement, and evaluate the perspective-based inspection approach. These projects gave us experience with tailoring the general approach to teams building software for space station monitoring, satellite scientific instrument control, and flight software, as well as for independent verification and validation of scientific mission software, among others. (Some of these experiences have been made available in the public literature [6], [18].) These experiences show that improving this inspection approach within each of these contexts provided multiple benefits for these teams, including improved ability to find major defects and improved efficiency.

1.3 How to improve?

For software inspections, there are three aspects that improvement activities should focus on. These areas are summarized in Figure 2.

Providing feedback to the project team about the outcome of each inspection should always be a concern. While we typically do not know at the time of the inspection the true number of real defects in the document (and hence, how effective a particular inspection has been), if a baseline of similar inspections and their results has been compiled we can at least compare the results to the baseline to see if it is out of the ordinary. An unexpectedly high or low number of defects being detected should cause the inspection moderator to double-check whether the inspection process was applied adequately.

The second aspect focuses on the inspection process itself. As indicated in the general inspection process (see Figure 1) there are certain degrees of freedom. For instance: project-specific entrance criteria should be used in the planning phase to ensure that the work product is ready for inspection; it is not always mandatory to conduct an overview meeting; the types of issues that inspectors should focus on

during their preparation will vary depending on the type of work product and the team's quality goals. Depending on the specific situation of a project team or the overall maturity of the organization, there is room for improving, or fine-tuning the process over time. These actions may provide some direct benefits to the project team, if the improvements can be identified and implemented during the life-cycle of the actual project. Otherwise such inspection process improvements are a first step towards cross-project improvements.

The last aspect fosters on organization-wide process improvement activities employing software inspections. To this end, the outcome of software inspections may be used to trigger or monitor organization–wide process improvement activities as for instance required by the CMMI® [4]. Most of the time, the focus is placed on trying to make better use of the gained inspection data across all projects.

1.4 Structure of this paper

The remainder of this paper is organized as follows: First, we give a short overview of the background and existing work for software inspections (Section 2). The focus is placed especially on existing work at NASA. We will describe how the collected inspection data is used for providing project feedback and improving overall inspection process by providing guidelines to the inspectors. In Section 3 we then will discuss how the collected inspection data can be used to move from project specific improvements to cross-project improvements. Section 4 will describe tools to support these cross project improvement activities and how they can be integrated into the project work. Finally, we conclude in Section 5 and sketch an outlook for future work.

2 Software Inspection Research at NASA

In many cases, data collected from an inspection is used to assess the quality of the current project. Hence, typically only the number of detected defects are recorded. However, the collection of additional measurement data is strongly recommended for the feedback that can be gained for more effective inspection planning and execution. By the early 1990s, researchers led by Dr. John Kelly at NASA's Jet Propulsion Laboratory (JPL), had started to systematically track and analyze key inspection metrics. Data on effort spent, on inspection team size, and on the size of the inspected document were used to detect patterns in the variables that led to more effective inspections. Based on hundreds of inspections, initial guidelines to help moderators in planning and executing the inspection process were formulated [11]. The guidelines focused on the inspection control metrics, that is, the parameters over which the inspection planner has direct influence. Modifying the values of these parameters is the mechanism by which an inspection moderator can affect the outcome of a given inspection. These parameters include:

- **Team Size:** The number of reviewers involved in the inspection. Teams which are too small are likely to lack important perspectives on the document, while teams which are too large are more likely to experience dynamics that make members less likely to participate effectively.
- **Page Rate:** The number of document pages that the inspectors examine per hour of the inspection meeting. The maximum page rate will depend on the type of document. Giving a team too much material to look through

will invariably result in a more superficial inspection, while giving too little material often runs the risk of leaving out important connections to other parts of the system.

A 2001 report [22] that characterized the state-of-the-practice of software inspections at NASA by means of interviews across multiple NASA Centers found that there were many recognized benefits of performing inspections. However, it also identified causes for concern, including that, in the face of schedule pressures, many projects found it difficult to keep the inspection process in place with an appropriate degree of formality. In response to this, we began experimenting with a particular approach to inspections, the Perspective-Based Inspection approach (PBI) [24]. PBI augments the process model described in Section 1 with additional guidelines that focus on defining the defect types targeted by the inspection from the point of view of stakeholders in the work product being inspected, and having each inspector represent a single stakeholder and use a scenario to actively work through the information.

The specifics of the software inspection approach can clearly impact the success of the inspection. In our previous work, we had demonstrated that the Perspective-Based approach was effective in some circumstances at improving the effectiveness of an inspection, increasing by up to 30% the amount of defects that were found by both individual inspectors and teams of inspectors [1]. In the period of 2000 – 2005, in work originally funded by a NASA SARP grant and later continued as multiple research infusion projects, we continued to work with this approach and adapt it to the needs of real project teams from multiple NASA Centers. Obviously, projects have specific needs and constraints that can be met with process improvement activities. We were able to show that the approach could be successfully tailored to the needs of diverse NASA project groups [[17], [6], [18]].

One of the issues complicating this work with multiple projects was that, in many instances, a solid measurement baseline was not available against which the results of a Perspective-Based inspection could be compared. We have explored ways in which the results of a given inspection could be compared to baselines from the wider industry [25] or against simulated results of other process variants [19], but clearly for use on NASA projects, the ideal would be a baseline composed of similar projects within the Agency. In addition to being useful to researchers, such a baseline would be useful to NASA personnel as well to provide decision support for their own process choices.

3 From Project Improvement to Organizational Improvement

Our recent work has focused on creating such inspection baselines, and using them to facilitate decision making ranging from the inspection process used by a single project to larger questions of organization scope.

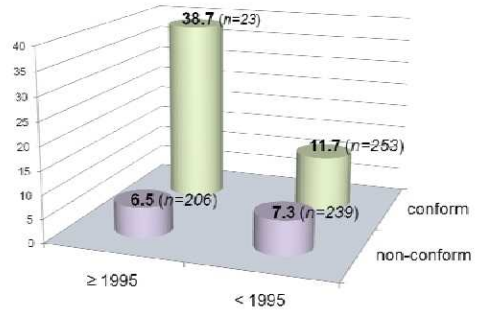


Figure 3: Influence of "*Team Size*" on the average number of overall defects found

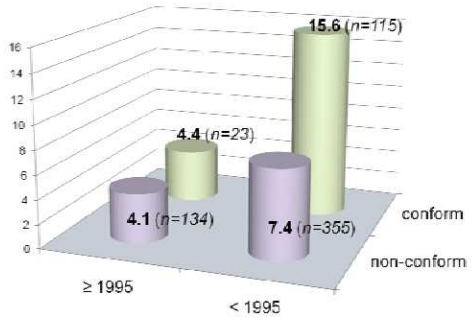


Figure 4: Influence of "*Page Rate*" on the average number of overall defects found

To do this, we worked to collect and unify inspection data from across NASA into a single dataset. The goal was to use and compare historical inspection data with data collected by recent programs. Such a centralized measurement database is the starting point for constant evaluation of existing guidelines and models.

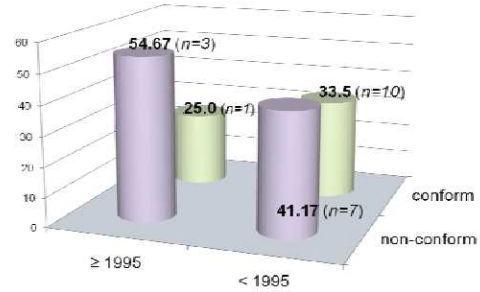
Unfortunately, not all data sets used the same defect classification schema. Aggregation of the different taxonomies and heterogeneous inspection data sets was necessary [20]. We started out by using a common defect classification schema: The Orthogonal Defect Classification as described in [3]. By following a defined process [20] of adopting, splitting, and merging the defect categories in the measurement data using different classification schemes, we were able to create a unified defect classification scheme into which all the others could be mapped. This allowed us to then aggregate all of our inspection data into a base with a total of 2,529 inspections from 81 projects across five NASA Centers. This database uses three unified defect taxonomies (see appendix for details) for inspections focusing on:

- Requirements
- Design and Source Code
- Test Plans

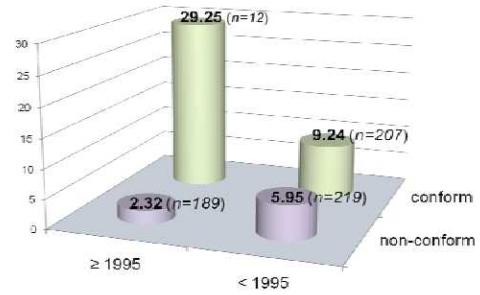
This combined dataset allowed us to study several improvement aspects at various levels, described in the subsections below.

3.1 Feedback on a single inspection

(a) Requirements



(b) Design and Source Code



(c)Test Plans

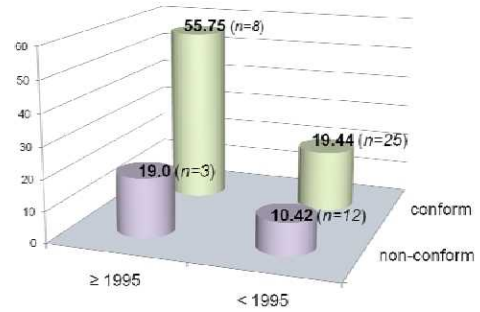


Figure 5: Influence of "*Team Size*" on the average number of overall defects found for different types of documents

We investigated whether we could use our Agency-wide dataset to test the guidelines originally proposed by Kelly for use on contemporary projects. We divided the dataset into data describing inspections in 1994 and earlier (the projects that would have been going on at the time the guidelines were originally formulated) and those from January 1, 1995, and later (the contemporary dataset). Some of our findings are displayed in Figure 3 and Figure 4. As can be seen, the average number of overall defects found when conforming to the suggested "*Team Size*" (Figure 3), is significantly higher than that seen on non-conforming projects. This holds true for both the historical measurement data lected in 1994 or earlier) as well as in the more contemporary ones (1995 and later). Figure 4 displays the same information for the suggested "*Page Rate*".

We then tested whether these lines should vary for different classes of inspection, for instance, for different types of documents. The initial results for the fluence of the control metric "*Team Size*" are displayed in Figure 5. The results regarding the defects that can be expected to be found clearly varied between the different document types. Please note that these are provisional results owing to the relatively small number of data points in some of the categories at this time. At this point we note that although the guidelines still seem to be useful rules of thumb for contemporary projects, fewer and fewer projects seem to be able to remain in conformance with them. Ongoing work is aimed at investigating whether the bounds can be relaxed, so that we can give guidelines that more projects may find it possible to conform to. As well, we are investigating the extent to which the guidelines themselves vary by type of project.

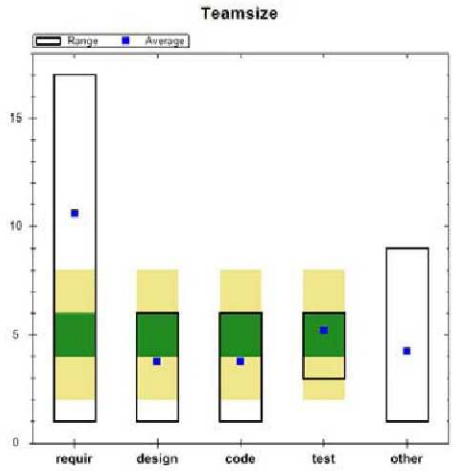


Figure 6: Team Size analysis. Recommendations for optimal values for increasing inspection effectiveness are shown in green, with the slightly less optimal range shown in yellow. The range of data from inspections at one NASA Center are represented with the black boxes; the mean value in this context in each category is represented with a dot.

Hence, these guidelines can serve as a baseline against which to give feedback on an individual inspection. We cannot say that every inspection should conform to these guidelines, of course; in many cases there are excellent reasons for non-conforming. Inspections of particularly complex portions of the system, for instance, may require a higher than average number of inspectors to cover the technical areas of interest. However, moderators can find that comparing their inspection parameters to the guidelines is a useful exercise, in that when values are outside the range found in the guideline, the moderator needs to verify that there is a sound reason for planning the inspection in this way.

3.2 Continuous monitoring and improvement of the inspection process

Once guidelines, such as those for the control metrics on "*Team Size*" and "*Page Rate*", have been defined, they should be monitored and periodically re-evaluated. This is due to the fact that there might be changes in the overall process or the context in which the technology is used (e.g., change in programming language or better qualifications of development team). In many cases, as the number of inspections from within the specific context of interest grows, better guidelines that are specific to the context can be formulated.

The charts in Figure 6 show the up-to date guidelines resulting from our research for the "*Team Size*" control metric, broken out by the type of work product being inspected. (The "other" category contains a mix of many different document types, so we have no guidelines explicitly formulated.) Overlaid on these displays are boxes representing the range of values seen in actual data from one NASA Center, with the dot representing the mean value for the control metric from that Center.

One of the things we recommend Centers and projects to do periodically is to look at their aggregated data compared to the guidelines. In this case, we can see that values for the "*Team Size*" metric from this Center generally track well with the guidelines, except in the case of requirements, where larger than recommended teams seem to be the norm. While this analysis cannot point to issues that necessarily *should* be changed, it should cause inspection planners to double-check whether there is a valid reason for departing from the recommendations. Especially in the case of requirements documents, it can be the case that many stakeholders need to be represented in the review to account for all the diverse technical areas that have some bearing on what set of requirements may in fact be feasible to build. Discussions with the personnel collecting data at this Center revealed that this reasoning was exactly the cause for the discrepancy. In such cases, rather than changing project behavior, it might be more worthwhile to create a tailored set of guidelines that are more focused on the needs in this particular context.

However, the same graph also demonstrates that a recurring problem in this environment was a tendency to create inspections of design, code, and test work products with fewer inspectors than recommended. This comparison might serve as a useful reminder to inspection planners to take the extra effort in formulating larger teams.

3.3 Supporting organization-wide improvements

Collecting and analyzing software inspection data in a centralized inspection measurement database is not only beneficial for improving the organization's inspection process. The data collected as part of the ongoing software inspections are also a valuable resource for process improvement activities and organizational learning. Focusing on defect distribution, the defect classification schema used in our inspection database allows for a comparison between software inspections (in early lifecycle phases) and testing activities (in late lifecycle phases). The research goal here is to monitor and improve the overall effectiveness of the verification and validation strategy.

By taking a closer look at the defect distribution data, which can be easily collected as a by-product of the inspections and testing processes, it becomes possible to identify issues in the overall software development process applied in the organization. Such analyses then may trigger organization-wide process improvement activities. Furthermore, these activities help to satisfy key CMMI requirements at different maturity levels [4], for example:

- **Level 2:** Measurement and analysis (MAS)
- **Level 3:** Verification (VER) and Validation (VAL)
- **Level 4:** Quantitative project management (QPM)

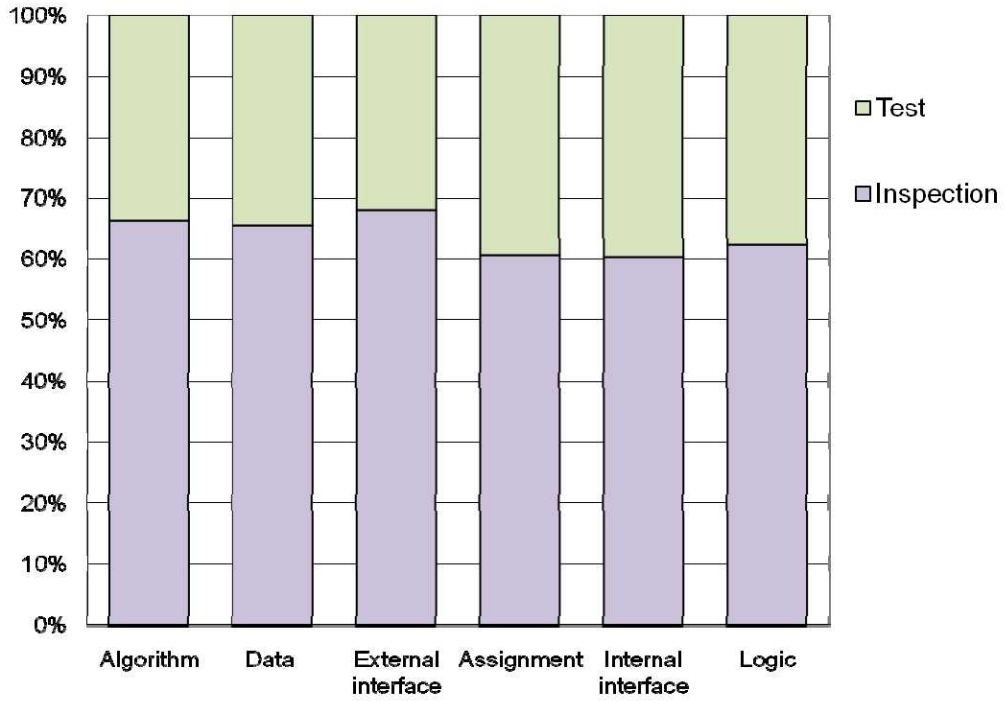


Figure 7: Defects found by inspections (in early life cycle phase) and testing (in late life-cycle phase

The results of these activities can then be used to help answer the following questions from the viewpoint of a development team:

- *If I choose to apply inspections, what are the implications for the effort required to be spent on other non-optional activities, like system testing?*
- *Can I make an informed decision about what type or how many inspection or testing activities to apply, based on the expected defect profile of a project?*

For example, a preliminary comparison between the defect types found as part of the software inspection process and testing activities in one set of NASA projects is displayed in Figure 7. The overall results can be summarized as follows:

- In the eight projects under study, on average 64% of the defects in the software were removed through inspections
- More than 60% of the defects were removed by inspections in each defect type
- Inspections were nearly equally effective across all defect types
- Inspections performed best on finding external interface defects
- If we considered the number of defects remaining after inspections as an approximation of the testing effort required, an analysis shows that testing effort could have been reduced by 2/3 on the projects studied

4 Necessary Tool Support

To make optimal use of the data collected in the course of a software inspection, a number of analyses need to be performed. Such analyses should be completed in a timely matter to provide quick feedback to the developers of the current project, and to allow process improvement experts to contact the inspection team if they

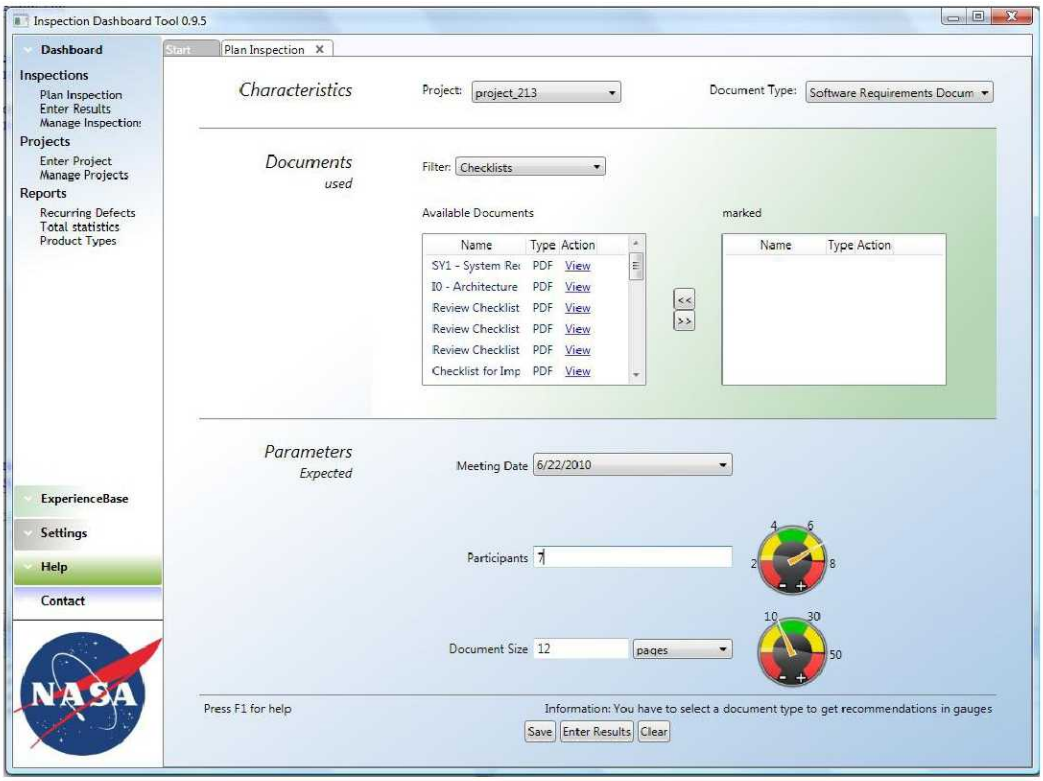


Figure 8: "Inspection Planning" screen of the Inspection Dashboard tool

have further questions regarding their work. To support such a quick analysis of the inspection data, effective tool support is needed. Such a tool can be a standalone solution, or preferably, a tailored version that is integrated into the overall inspection and software development process and coupled with the organization's experience base. There are two primary user types of such a tool: the developers who plan and conduct actual inspections, and the Software Process Improvement (SPI) group focusing on maintaining and evaluating the current guidelines and models.

4.1 Dashboard tool for planning support and initial feedback

To support our research work, we have developed an initial prototype Inspection Dashboard tool that leverages an organization's existing knowledge regarding software defects to improve strategies for software quality. The complex analyses and models built from our centralized inspection measurement database are hidden behind a tool interface providing customized and context-sensitive recommendations based upon a project's domain, maturity, and other factors. The Dashboard tool automates many of the analyses described in the prior sections. Among others, it provides feedback to the project teams on how well a planned inspection meets historical targets for the control metrics and how the actual outcomes (e.g., the number of defects found) can be compared to the historical guidelines. Users are not only able to plan, store, analyze, and display inspection data for their current project, but also contribute to the overall inspection measurement database to allow improvement of the inspection process across the organization.

Developers can use the tool to plan an inspection, enter the results and gain feedback on the performance of the inspection compared to similar past in-

Suggested Resource: Inspections Experience Base

(http://fc-md.umd.edu/EB/startup)

This on-line source provides a central repository for checklists and other documents that are useful for inspections. The web site is a by-product of a recent NASA SARP research initiative focused on improving the overall inspection process.

Materials in the repository come from projects and trainings from across NASA. Features include:

- Ability to search by type of material (i.e., checklists, forms, and tools & resources)
- Ability to filter by the type of artifact to be inspected (i.e., requirements, architecture, design, code, or hardware)
- Search for materials developed at a specific Center

For instance, click on "Checklists" under "Tools & Resources" on the left side menu to access several checklists for conducting inspections. Other documents found on the web site are commonly used defect classifications (see "Defect classifications" entry under "Tools & Resources" on the left side menu), as well as spread sheets for collecting measurement data (click "Forms & Templates" in the left side menu).

To share some of your own collected inspection materials with other NASA projects, you can simply add them into the collection by clicking on one of the links in left side menu under "Suggest new documents".

If you have any questions regarding the inspection experience base, please do not hesitate to contact Dr. Forrest Shull via e-mail at fshull@fc-md.umd.edu.

spections. Thereby, the tool offers support for the "*Planning*" and "*Inspection Meeting*" steps of the Software Inspection Process (see Figure 1).

In the "Planning" step, developers use the input screen as displayed in Figure 8. Initially, developers enter general information such as the project name and the document type to be inspected. Based on these inputs the tool then suggests documents (e.g., checklists or defect reporting sheets) that can be reused for this task. The tool accesses an Inspection Experience Base (EB), which is also publicly available through the FC-MD web server (http://fc-md.umd.edu/EB). The Inspection EB allows inspection planners to benefit from the past experiences of other projects that have applied inspections. Specifically, the EB provides access to inspection materials, such as checklists and forms that can be used by teams planning new inspections. These materials are organized according to the type of work product for which they can be applied and the types of projects that have applied them in the past. Teams planning new inspections will be able to see what resources might be appropriate for their context and whether they have been applied at NASA or in another context. Other useful information related to software inspections, such as definitions, defect taxonomies, related literature, or existing tools are also available and up-dated periodically.

Finally, the Developer enters the planned date of the inspection meeting, the number of participants, and the document size. As soon as the document size and number of participants are entered, the tool uses the underlying control metrics to provide immediate feedback to the developer. The dashboards next to the fields with the entered values indicate whether or not the planned parameters are within the suggested range (green area) or above or below the acceptable values (red and yellow areas to the right and left).

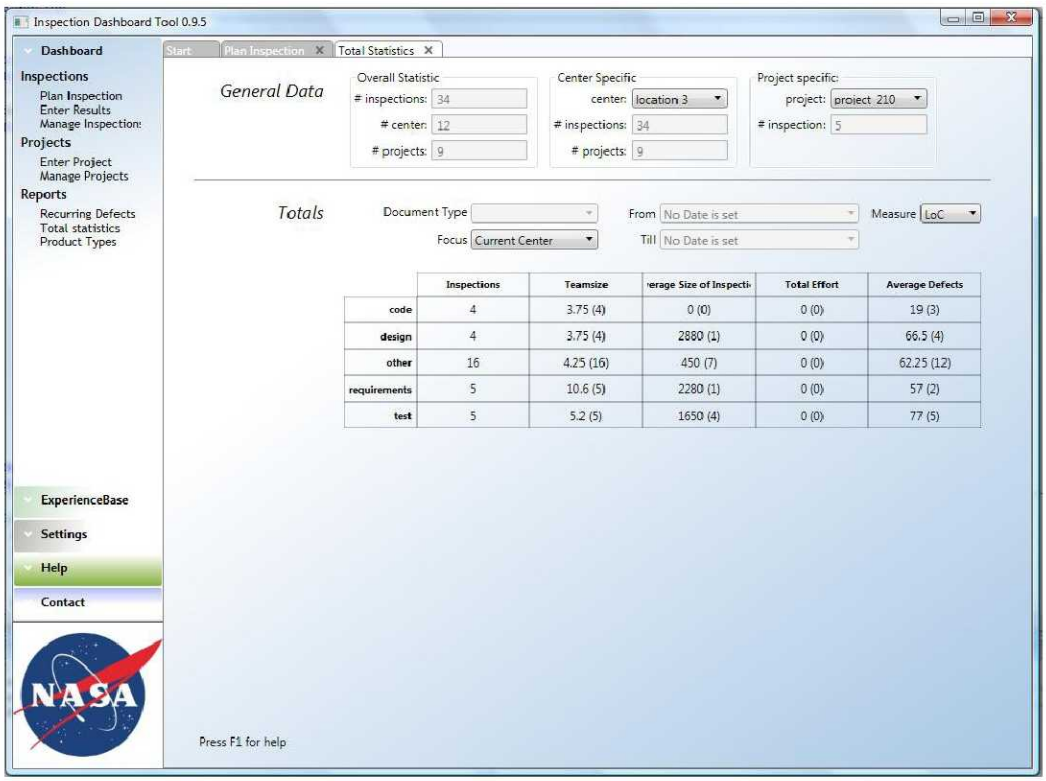


Figure 9: "Total Statistics" overview screen of the Inspection Dashboard tool

At the end of the inspection meeting, the developers can use the tool to enter the findings for the inspections. As discussed above, projects have used a wide range of different terminologies and data collection mechanisms to collect their defect data. Since a tool that asked projects to adapt to new vocabulary and new mechanisms is less likely to be used than one that adapts to the existing project-specific decisions, we offer a number of options for users to import their current defect reports into the dashboard and get feedback. The mappings between different data types, e.g., between the project-specific defect taxonomy and the unified ones we have adopted for "*Requirements*", "*Design and Source Code*", and "*Test Plans*" (see appendix), are done automatically and only the project-specific terminology is shown to the user. Again, immediate feedback is available to the user, whether the inspection meets the guidelines or not.

The SPI group members can use the tool to analyze the collected inspection measurement data to validate, up-date, and improve the overall processes. To support the SPI team in their tasks, our Inspection Dashboard tool provides specific information, such as the "Total Statistics" screen displayed in Figure 9. Using the filter options (e.g., to select a specific time-frame, project, or Center), the tool automatically provides feedback on the entries in the inspection measurement database which meet the selected criteria. This easy and fast feedback mechanism can, for instance, be used to validate the existing guidelines.

4.2 Tool support for other steps of the inspection process

Many tools are available to support developers in performing steps of the Software Inspection Process (see Figure 1). Among them, we would like to point out the ISPIS framework [[9], [10]] and the ISI tool [15]. These tools support users in

conducting various inspection process steps that are currently not covered by our Inspection Dashboard tool. As part of the support offered, these tools help in collecting the measurement data and formulating the inspection reports.

The **ISPIS framework** helps to assign, schedule, track, and coordinate the different inspection activities. In the "*Planning*" step, it supports developers by assigning and distributing the different review materials and tasks (i.e., perspectives and checklists) to the concrete developers. As part of the "*Preparation*" step, each developer can then use the tool to report the potential defects. The defect lists are compared and analyzed by the tool. Defects that are recorded by several inspectors are ranked differently than ones that are only reported by a single inspector. This consolidated ranking can be used as a starting point for the "*Inspection Meeting*" step. The final meeting results as well as the different data points collected during the process can be automatically consolidated by the tool. Aggregation of the different measurement data points from each inspector is also supported by the tool. Such a consolidated list, for instance, of the total effort spent, can then be used as an input to our Inspection Dashboard tool. Hence, the ISPIS framework fills a gap in the current tool support.

The **ISI tool** actually supports the inspectors in reading and inspecting code documents. It uses rules to slice and highlight the code to be inspected according to specific checklist points. For instance, if a checklist explicitly asks for a review of variable declarations and initializations, the tool will highlight all lines of code that have such initial assignments and declarations of variables. The inspectors thereby are more likely to find all occurrences of variables in the code and can especially focus on each of the specific check-list items. The tool further helps to generate a basic inspection report based on the checklist items inspected.

By supporting the inspectors in their work, tools such as ISI and ISPIS in addition collect and aggregate measurement data and information, which are a valuable input for the inspection improvement activities described in this paper.

5 Conclusions and Future Work

When estimating the overall cost and benefits of software inspections, organizations often only take into account the direct project costs and the estimated savings based on avoided rework and recalls. However, by fully employing all of the collected measurement data, one has a powerful tool at hand for overall process improvement and monitoring activities. In general, the cost of inspections should not only be judged in the context of a single project, but also be considered as a valuable input for the overall organization and its continuous improvement and learning efforts across projects.

Software inspection data enable an organization to control and optimize their overall development processes and help identify systemic problems. The classification of defects, for instance, allows for a comparison between software inspections and testing activities, while at the same time monitoring and improving the overall V&V effectiveness.

In this paper we detailed such cross project activities. The presented results are based on historic and contemporary inspection measurement data collected throughout NASA. Ongoing improvement initiatives are studying possible other benefits from employing the collected software inspection data on a regular basis as part of the daily software development activities – which then will be beneficial for the project teams as well as the overall organization. In particular, we are continuing to investigate how to improve the guidelines we offer as starting points to

project teams in various contexts – for example, we would expect that the guidelines we offer to safety-critical missions would be different from those we offer to projects with less constrained quality requirements. We are also looking at whether we can apply the same basic approach, with different guidelines, for inspections in the realm of system engineering artifacts.

Acknowledgments

This work was sponsored by NASA grant NNG05GE77G, "Full-Lifecycle Defect Management Assessment." The authors wish to thank our collaborators at NASA who helped us compile this data set from across many projects and Centers, without whom this work would not have been possible.

References

[1] Basili, V., Green , S., Laitenberger, O., Lanubile, F., Shull, F., Soerumgaard, S., Zelkowitz, M. (1996) The Empirical Investigation of Perspective-Based Reading. Empirical Software Engineering: An International Journal 1(2):133–164

[2] Ciolkowski, C., Differding, C., Laitenberger, O., Muench, J.(1997) Empirical Investigation of Perspective-based Reading: A Replicated Experiment. International Software Engineering Research Network (ISERN), Technical Report ISERN-97-13

[3] Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus, D.S., Ray, B.K., Wong, M.-Y. (1992) Orthogonal Defect Classification–A Concept for In-Process Measurements. IEEE Transactions on Software Engineering 18(11):943–956

[4] Chrissi, M.B., Konrad, M., Shrum, S. (2007) CMMI® Second Edition – Guidelines for Process Integration and Product Improvement. SEI Series in Software Engineering, Addison Wesley Professional

[5] Conradi, R., Mohagheghi, P., Arif, T., Hegde, L.C., Bunde, G.A., Pedersen, A. (2003) Object-Oriented Reading Techniques for Inspection of UML Models—An Industrial Experiment. European Conference on Object-Oriented Programming (ECOOP'03), Darmstadt, Germany

[6] Denger, C., Shull, F. (2007) A Practical Approach for Quality-Driven Inspections. IEEE Software 24(2):79–86

[7] Fagan, M.E. (1976) Design and code inspection to reduce errors in program development. IBM Systems Journal 15(3):182–211

[8] Gilb, T., Graham, D. (1993) Software Inspection. Addison-Wesley Publishing Company

[9] Kalinowski, M., Travassos, G.H. (2004) A computational framework for supporting software inspections. Proceedings of the 19th International Conference on Automated Software Engineering (ASE04), Linz, Austria, IEEE Computer Society

[10] Kalinowski, M., Travassos, G.H. (2004) ISPIS: A framework supporting software inspection processes. Proceedings of the 19th International Conference on Automated Software Engineering (ASE04), Linz, Austria, IEEE Computer Society

[11] Kelly, J.C., Sherif, J.S., Hops, J. (1992) An Analysis of Defect Densities Found During Software Inspections. Journal of Systems Software 17(2):111–117

[12] Kolkhorst, B.G. (1992) Space Shuttle Primary Onboard Software Development: Process Control & Defect Cause Analysis. IBM Corporation Technical Report, Houston, Texas, pp 1–15

[13] Laitenberger, O., El Emam, K., Harbich, T. (2000) An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective-based Reading of Code Documents. IEEE Transactions on Software Engineering 27(5):387–421

[14] NASA Procedural Requirements7150.2 (2009) Subject: NASA Software Engineering Requirements. Available at http://nodis3.gsfc.nasa.gov/displayDir.cfm?t=NPR&c=7150&s=2. Accessed 30 Jun 2009

[15] Nick, M., Denger, C., Willrich, T. Experience-Based Support for Code Inspections. In Althoff, K.-D., Dengel, A., Bergmann, R., Nick, M., Roth-Berghofer, T. (Eds.): Professional Knowledge Management, Third Biennial Conference, WM 2005, Kaiserslautern, Germany, pp. 121–126

[16] Porter, A., Votta, L. (1998) Comparing Detection Methods for Software Requirement Inspections: A Replicated Experiment using Professional Subjects. Empirical Software Engineering: An International Journal 3(4):355–379

[17] Pressburger, T., Hinchey, M., Feather, M.S., Markosian, L. (2006) Infusing Software Engineering Technology into Practice at NASA. 2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06), pp.89–100
[18] Pressburger, T., Markosian, L. (2004) Software Engineering Research/Developer Collaborations in 2004 (CI04): Final Report. Available at: http://ti.arc.nasa.gov/m/pub/806h/0806%20(Pressburger).pdf Accessed 30 Jun 2009
[19] Rus, I., Shull, F., Donzelli, P. (2003) Decision Support for Using Software Inspections. 28th Annual NASA Goddard Software Engineering Workshop (SEW'03), pp 3
[20] Seaman, C., Shull, F., Regardie, M., Elbert, D., Feldmann, R.L., Guo, Y., Godfrey, S. (2008) Defect Categorization: Making Use of a Decade of Widely Varying Historical Data. Proceedings of the second ACM–IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM08), Kaiserslautern, Germany, The Association for Computing Machinery, New York
[21] Selby, R.W. (1990) Empirically based analysis of failures in software systems. IEEE Transactions on Reliability 39(4):444–454
[22] Shull, F., Bachman, J., Van Voorhis, J., Larsen, P. (2001) "Lessons Learned Report for the 'State-of-the-Art Software Inspections and Reading' Initiative." Deliverable to the NASA OSMA SARP, Available from http://sarpresults.ivv.nasa.gov/DownloadFile/31/11/Lessons%20Learned%20Report.doc
[23] Shull, F., Basili, V.R., Boehm, B., Brown, A.W., Costa, P., Lindvall, M., Port, D., Rus, I., Tesoriero, R., Zelkowitz, M.V. (2002) What We Have Learned About Fighting Defects. Ottawa, Canada, IEEE. 8th International Software Metrics Symposium, pp 249–258
[24] Shull, F., Rus, I., and Basili, V. R., (2000) "How Perspective-Based Reading Can Improve Requirements Inspections," IEEE Computer 33(7):73–79
[25] Wohlin, C., Aurum, A., Petersson, H., Shull, F., Ciolkowski, M., (2002) Software Inspection Benchmarking - A Qualitative and Quantitative Comparative Opportunity. In Proceedings IEEE International Symposium on Software Metrics (METRICS02), Ottawa, Canada, pp. 118–130

Appendix

1. Requirements defects

Defect Type	Definition
clarity	A problem in the wording or organization of the document that makes it difficult to understand.
completeness	A missing requirement or other piece of information.
compliance	A problem with compliance to any relevant standard.
consistency	Two or more statements in the document that are not consistent with each other, e.g., requirements that are mutually exclusive.
correctness	Any statement in the document that is incorrect.
testability	A requirement that is not stated in a way that makes it clear how it can be tested.
other	Anything that does not fit any of the above categories that is logged during a requirements inspection.

2. Design/code defects

Defect Type	Definition
algorithm / method	An error in the sequence or set of steps used to solve a particular problem or computation, including mistakes in computations, incorrect implementation of algorithms, or calls to an inappropriate function for the algorithm being implemented.
assignment / initialization	A variable or data item that is assigned a value incorrectly or is not initialized properly or where the initialization scenario is mishandled (e.g., incorrect publish or subscribe, incorrect opening of file, etc.)
checking	Inadequate checking for potential error conditions, or an inappropriate response is specified for error conditions.
data	Error in specifying or manipulating data items, incorrectly defined data struc-

	ture, pointer or memory allocation errors, or incorrect type conversions.
external interface	Errors in the user interface (including usability problems) or the interfaces with other systems.
internal interface	Errors in the interfaces between system components, including mismatched calling sequences and incorrect opening, reading, writing or closing of files and databases.
logic	Incorrect logical conditions on if, case or loop blocks, including incorrect boundary conditions ("off by one" errors are an example) being applied, or incorrect expression (e.g., incorrect use of parentheses in a mathematical expression).
non-functional defects	Includes non-compliance with standards, failure to meet non-functional requirements such as portability and performance constraints, and lack of clarity of the design or code to the reader - both in the comments and the code itself.
timing / optimization	Errors that will cause timing (e.g., potential race conditions) or performance problems (e.g., unnecessarily slow implementation of an algorithm).
other	Anything that does not fit any of the above categories that is logged during an inspection of a design artifact or source code.

3. Test plan defects

Defect Type	Definition
clarity	A problem in the wording or organization of the document that makes it difficult to understand.
completeness	A missing test case or other piece of information.
compliance	A problem with compliance to any relevant standard.
correctness	Any statement in the document that is incorrect, including incorrect expected output for a test case.
testability	An infeasible test case (e.g., one too costly to test).
redundancy	Test cases or other information that is not necessary because it appears more than once.
other	Anything that does not fit any of the above categories that is logged during a test plan inspection.