Fraunhofer USA, Inc

Center for Experimental
Software Engineering,
Maryland

Full Life-Cycle Defect Management Assessment

Initial Inspection Data Collection Results and Research Questions for Further Study

Work Performed under SARP 2007 Research Initiative

Prepared by

Dr. Forrest Shull, Mr. Raimund L. Feldman, Mr. Ralf Haingaertner, Ms. Myrna Regardie, Dr. Carolyn Seaman

Contact:

{fshull, rfeldmann, rhaingaertner, mregardie, cseaman}@fc-md.umd.edu

Keywords:

NASA / SARP 2007 / Inspections / Results and Questions

July 2007

REVISION HISTORY

Date	Version	Status	Reason for Change
07/31/2007	1.0	Initial version	New document

TABLE OF CONTENTS

1. Background and Motivation

It is often the case in software projects that when schedule and budget resources are limited, the Verification and Validation (V&V) activities suffer. Fewer V&V activities can be afforded and moreover, short-term challenges can result in V&V activities being scaled back or dropped altogether. As a result, too often the default solution is to save activities for improving software quality until too late in the life-cycle, relying on late-term code inspections followed by thorough testing activities to reduce defect counts to acceptable levels. As many project managers realize, however, this is a resource-intensive way of achieving the required quality for software.

The "Full Life-cycle Defect Management Assessment" Initiative, funded by NASA's Office of Safety and Mission Assurance under the Software Assurance Research Program, aims to address these problems by:

- Improving the effectiveness of early life-cycle V&V activities to make their benefits more attractive to team leads. Specifically, we focus on software inspection, a proven method that can be applied to any software work product, long before executable code has been developed;

- Better communicating this effectiveness to software development teams, along with suggestions for parameters to improve in the future to increase effectiveness;

- Analyzing the impact of early life-cycle V&V on the effectiveness and cost required for late life-cycle V&V activities, such as testing, in order to make the tradeoffs more apparent.

This white paper reports on an initial milestone in this work, the development of a preliminary model of inspection effectiveness across multiple NASA Centers. This model contributes toward reaching our project goals by:

- Allowing an examination of inspection parameters, across different types of projects and different work products, for an analysis of factors that impact defect detection effectiveness.

- Allowing a comparison of this NASA-specific model to existing recommendations in the literature regarding how to plan effective inspections.

- Forming a baseline model which can be extended to incorporate factors describing: the numbers and types of defects that are missed by inspections; how such defects flow downstream through software development phases; how effectively they can be caught by testing activities in the late stages of development.

The model has been implemented in a prototype web-enabled decision-support tool which allows developers to enter their inspection data and receive feedback based on a comparison against the model. The tool also allows users to access reusable materials (such as checklists) from projects included in the baseline. Both the tool itself and the model underlying it will continue to be extended throughout the remainder of this initiative.

As results of analyzing inspection effectiveness for defect containment are determined, they can be shared via the tool and also via updates to existing training courses on metrics and software inspections. Moreover, the tool will help satisfy key CMMI requirements for

the NASA Centers, as it will enable NASA to take a global view across peer review results for various types of projects to identify systemic problems. This analysis can result in continuous improvements to the approach to verification.

The remainder of the document is organized as follows: Section 2 details our process while our current results are provided in Sections 3 and 4. Based on our findings, some of the capabilities have already been incorporated into a dashboard-type tool developed as part of this project (see Section 5). Finally, an overview of research questions to guide further work is in Section 6. Section 7 lists the references cited. Appendix A provides detailed information about the original source data. Appendix B includes a reference list of acronyms, terms, and definitions used throughout this preliminary results report.

2. Process

2.1. Approach

The main approach used was to work with several Centers, discuss the purpose of the SARP research initiative, and obtain access to existing inspection data collected by projects (which in various contexts can be stored on paper, using the InSpec tool, in the eRoom collaborative environment, within an MS Access database, or using spreadsheets). We were interested in data from recent projects as well as historical data, such as that collected by the Software Engineering Laboratory (SEL) at NASA GSFC. The Fraunhofer Center Maryland (FC-MD) team looked at the product types inspected in the data received (e.g., requirements documents, design documents, source code, test plans), defect types (logic, external/internal interface, initialization,…), defect severity (major, minor), inspection effort related data (meeting length, team size, preparation time, total inspection effort, size of product inspected), and characteristics about the project (e.g., flight SW, development language, safety critical, NPR class of project, in-house/contracted out, etc.).

Data models were built for each source as the data became available, and then mappings were created so that data from different contexts, but related to inspections of the same work product, could be compared across organizations. These mappings were used in our attempts to build models for individual projects which can be combined to provide insight within each NASA Center and across the entire agency. Initial attempts have been made to combine Center data and provide a NASA agency view, but it may be too early to do so for some types of data, in particular data on defect types, which seems to vary widely. Other areas are more promising, for instance, at an agency level, it may be possible to combine all inspection meeting length data collected to date and provide some general guidance on "typical" or "recommended" inspection meeting length to help ensure inspection effectiveness.

2.2. Data Protection

A key concern expressed by most Centers we approached for data access is the sensitivity of the data and how it could be used or inadvertently misused to make a Center or project look bad. On the other hand, Centers were interested in having the FC-MD analyze their data and build decision support models for them, which is time-consuming for projects and Centers to do. In many cases the appropriate data for such analyses has been collected at the Center, but not been previously analyzed in depth. The FC-MD team negotiated and signed Memorandums of Understanding (MOUs) that describe fair use of and necessary protections for the data, in order to gain access.

Data protection and privacy has been a major consideration in the design of our prototype support tool (See Section 5). For example, we have made a design decision to incorporate roles and security levels for users of the tool so that, for example, detailed project data will only be accessible to members of that project team. Organizational-level persons such as SEPG members would be able to look at all the data for the organization and Center, unless otherwise restricted. The decision support tool would control this through permissions. The data in the underlying models will be rolled together, but not attributed to any individual projects, organizations, or Centers. FC-MD will use translation tables to hide the identity and protect the source data.

2.3. Source Data

To date, we have gained such access to data from three separate groups (referred to as data sources A, B, and C), with several more in progress. The data includes data on 1334 inspections with 6110 defects reported. This data has been used to build some preliminary models by project and by Center. Analysis has been performed in an attempt to build a generic set of models that can later be used across Centers or at least for an individual project to compare the effectiveness of their inspections with how other projects, Centers, or the Agency are doing. In the long term the purpose would be for the decision support tool to be able to look for systemic problems and provide guidance on potential means to mitigate the specific problem.

2.4. Analysis of Source Data

The source data was first analyzed by product type. Some projects inspect project plans, requirements documents, test plans, design documents, source code and some only inspect a subset of these. With respect to defect data, some projects categorize defects into a very large number of defect types by product and some only indicate whether the defect was major or minor. After looking at a wide variation of defect types for each product type and their corresponding definitions, the FC-MD recommended an initial set of defect types to be collected by teams going forward. The set of defect types and definitions are provided in Section 3.1.

Although these particular defect types have been chosen for cross-Center analysis, the tool will still allow projects to collect their original defect types and input that data directly into the tool. The tool will translate the data into the recommended set of defect types and compare with the baseline set of models. It is expected that the baseline set of models will be reviewed periodically and revised, if relevant. For projects not already collecting inspection data the intention would be to introduce the recommended set of defect types. Also, in time, as more data is collected, it is envisioned that the recommended set could be refined. The tool is designed so as to make this type of change relatively easy.

Other types of inspection-related data (e.g., effort, number of inspectors, etc.) have also been analyzed, again in an effort to build baselines and recommendations for use across the Agency. The preliminary results of this analysis are presented in Section 4.

3. Analysis Results of Inspection Defect Type Data

We examined inspection defect data for several purposes. First, we wanted to understand how defects were classified at different Centers. Second, one of our goals was to develop a defect classification scheme that accommodated the defect classification practices used in our different data sources and that could be used to guide inspection data collection going forward. Finally, we wanted to determine what types of defects tended to be found in inspections in general, and if the distribution of defect types varied among our data sources.

In examining the data from the different sources, we found that different names were used for similar artifacts that were inspected. One of our data sources reported data on unit design and another data source reported data on both architectural design and detailed design. For our analysis, we have combined all of these into the design category. Similarly, in another case, a data source distinguished between software requirements and subsystem level requirements. For our analysis, we have combined both requirements types into one requirements category. Using this type of analysis we currently have four inspection product categories: requirements, design, code, and test. It is anticipated that others will be added in the future, if warranted.

In the sections that follow, we first present our proposed defect classification, grouped into the four inspection product categories mentioned above, as well as how our defect categories map to the defect categories used in the Centers. Then we present the various analyses conducted to investigate the distribution of defect types.

3.1. Proposed FC-MD Inspection Defect Types by Product

Below is a simple listing of the defect categories that resulted from our analysis, for the different work products that are inspected.

Requirements
clarity
completeness
compliance
consistency
correctness
testability
other

Design/Code
algorithm/method
assignment/initialization
checking (i.e. error handling)
data
external interface
internal interface
logic
non-functional defects
timing/optimization
other

Test Plans
clarity
completeness
compliance
correctness
redundancy
testability
other

We found that the categories that make sense and that are useful for analysis of design and code inspections are not helpful for requirements and test plans, therefore these work products each have different defect categories. The defect classifications for requirements and test plans are almost identical, with most of the defect types relating to common characteristics of natural language documents. The only difference between the two classifications is that the requirements defect type “consistency” is replaced by the test

plan defect type "redundancy". Consistency, we believe, is not normally a crucial aspect of test plans that would raise defects. However, redundant test cases or test schemes would be a possible source of defects raised in an inspection of test plans. In forming these categories, we began with an industry standard, Orthogonal Defect Classification (ODC) [1], and then added categories from data currently collected from Centers that could not be mapped to the existing ODC categories. We attempted to achieve a compromise between facilitating the mapping of existing data and preserving the standard categories from ODC. This is an initial categorization that is expected to evolve over time as additional data sources are incorporated.

3.2. Inspection Defect Type Definitions

The tables below present definitions of the defect types presented in the previous section.

Table 1. Requirements inspection defect types

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.

Table 2. Design and Source Code inspection defect types

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 structure, 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.

Defect Type	Definition
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.

Table 3. Test Plan inspection defect types

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	Test case may not be testable because it is infeasible (e.g., 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.

3.3. Defect Type Translation Tables and Mapping Rationale

Tables 4, 5, and 6 below depict how the original source data inspection defect types were mapped to the proposed FC-MD defect types by product type inspected. The tables also provide some rationale for the mapping, where it is not straightforward. In some cases, the original source did not include any defects that mapped to one of the proposed defect types. In those cases, the FC-MD defect type is excluded from the table for that data source.

Table 4. Mapping for Requirements inspection defect types

Defect Types from Data Sources	Mapped to FC-MD Proposed Defect Types	Rationale
Data Source A		
None		
Data Source B		
completeness	completeness	straightforward

Defect Types from Data Sources	Mapped to FC-MD Proposed Defect Types	Rationale
reliability	completeness	The assumption is that problems with reliability usually have to do with something missing, e.g., a check for an error condition.
data usage	completeness	The assumption is that problems with data usually have to do with something missing, e.g., a missing data definition.
interface	completeness	The assumption is that problems with interfaces usually have to do with something missing, e.g., a missing specification of input or output data.
maintainability	completeness	The assumption is that problems with maintainability usually have to do with something missing, e.g., sufficient documentation or separation of concerns.
performance	completeness	The assumption is that problems with performance usually have to do with something missing, e.g., a performance requirement.
clarity	clarity	straightforward
level of detail	clarity	An inadequate level of detail implies a problem of clarity.
correctness	correctness	straightforward
compliance to standards	compliance	straightforward
traceability	compliance	It is assumed that there are standards about what needs to be traceable to what, so a lack of traceability could be seen as a standards compliance problem.
consistency	consistency	straightforward
testability	testability	straightforward
functionality	other	"Functionality" defects cannot be cleanly assigned to a single existing category.
feasibility	other	There is no other category that would cover feasibility problems in requirements.
Data Source C		
None		

Table 5. Mapping of Design and Code inspection defect types

Defect Types from Data Sources	Mapped to FC-MD Proposed Defect Types	Rationale
Data Source A		
Initialization	assignment/initialization	straightforward
Data value or structure	data	straightforward
Logic/control structures	logic	straightforward
Computational	algorithm/method	Definition of "algorithm/method" category includes computational issues.
Internal interface	internal interface	straightforward
External interface	external interface	straightforward
Data Source B		
anomaly management	checking	These terms are believed to be synonyms.
performance	timing/optimization	Performance is seen as a sub-concept of optimization.
data; data usage	data	straightforward
control	logic	These terms are believed to be synonyms.
computation; accuracy	algorithm/method	Definition of "algorithm/method" category includes computational issues; it is assumed that most accuracy problems are problems with the algorithm being employed.
interface (half); linkage	internal interface	Data Source B does not distinguish between internal and external interface defects, so we have elected to split the interface category evenly between the two. "Linkage" is assumed to mean linkages between different parts of the system, which is akin to internal interface problems.
interface (half)	external interface	Same as above
compliance to standards; portability; maintainability; clarity; functionality	non-functional defects	All these Data Source B categories are assumed to not affect external behavior of the system, thus can be considered non-functional.
other; qualify; modularity	other	We are not sure what "qualify" defects are. Modularity does not fit into any other category.
completeness; consistency; correctness	assignment/initialization; logic; algorithm/method; data; internal interface; external interface	We believe that design or code defects labeled as completeness, consistency, or correctness could be defects of any of these six types. We first distributed the "three c's" defects evenly among the six types, but that resulted in these defects dominating the dataset, and a nearly equal distribution of defects over these types as a whole. So our

Defect Types from Data Sources	Mapped to FC-MD Proposed Defect Types	Rationale
		final decision was to distribute the "three c's" defects over the six types according to the distribution of defects in these six types in the rest of the Data Source B dataset.
Data Source C		
optimization	timing/optimization	These terms are believed to be synonyms.
unset	assignment/initialization	We believe that "unset" refers to unset variables or other data items, i.e., things that have not been initialized properly.
program logic	logic	These terms are believed to be synonyms.
usability	external interface	Usability is one type of external interface problem.
Coding Standard; Clarify; Suggestion	non-functional defects	All these categories are assumed to not affect external behavior of the system, thus can be considered non-functional.
other	other	straightforward

Table 6. Mapping of Test Plan inspection defect types

Defect Types from Data Sources	Mapped to FC-MD Proposed Defect Types	Rationale
Data Source A		
Half of all "data value or structure", "initialization", "internal interface" and "logic/control structures" defects	completeness	These four categories were the only 4 used for test plan inspections in the Data Source A database. We do not have enough information to make a complete mapping, but we believe that all defects in these 4 categories correspond to actual functional defects, and so are not appropriate for categories such as clarity, compliance, etc.; they are more concerned with non-functional issues. So we decided to split the defects in these 4 categories between the "completeness" and "consistency" categories.
Half of all "data value or structure", "initialization", "internal interface" and "logic/control structures" defects	correctness	Same explanation as above.
Data Source B		
completeness	completeness	straightforward
reliability	completeness	The assumption is that problems with

Defect Types from Data Sources	Mapped to FC-MD Proposed Defect Types	Rationale
		reliability usually have to do with something missing, e.g., a check for an error condition.
data usage	completeness	The assumption is that problems with data usually have to do with something missing, e.g., a missing data definition.
interface	completeness	The assumption is that problems with interfaces usually have to do with something missing, e.g., a missing specification of input or output data.
maintainability	completeness	The assumption is that problems with maintainability usually have to do with something missing.
performance	completeness	The assumption is that problems with performance usually have to do with something missing, e.g., testing a performance requirement.
clarity	clarity	straightforward
level of detail	clarity	An inadequate level of detail implies a problem of clarity.
correctness	correctness	straightforward
consistency	correctness	If two statements in the test plan are inconsistent, then one can assume that one of them is incorrect.
compliance to standards	compliance	straightforward
traceability	compliance	It is assumed that there are standards about what needs to be traceable to what, so a lack of traceability could be seen as a standards compliance problem.
consistency	consistency	straightforward
testability	testability	straightforward
functionality	other	"Functionality" defects cannot be cleanly assigned to a single existing category.
feasibility	other	There is no other category that would cover feasibility problems in the test plan.
other	other	straightforward
Data Source C		
None		

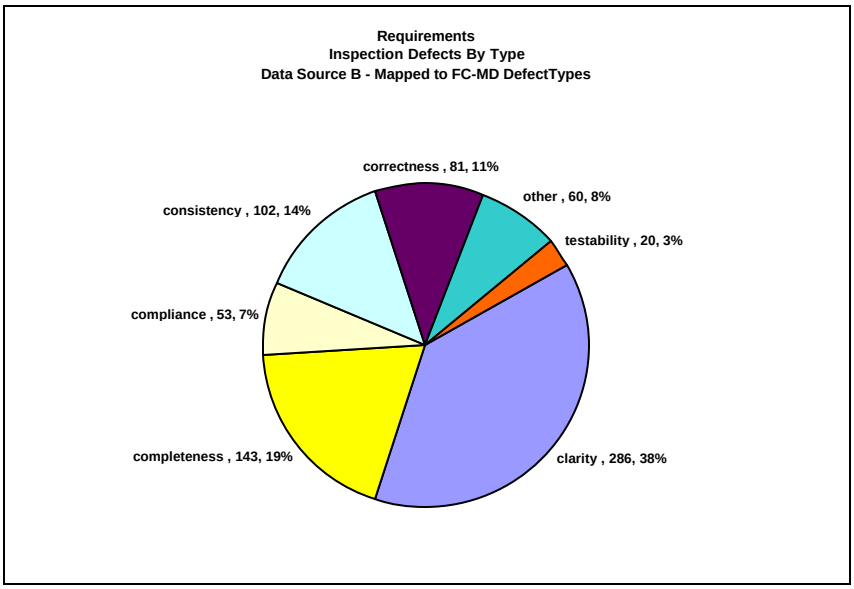
3.4. Inspection Defect Type Models by Work Product Type

This subsection contains the models formulated by analyzing the historical inspection data obtained and analyzed to date. It is expected as more data is received from Centers currently collecting data on inspections the newer data will be used to validate the baseline models. The detailed defect type data by product inspected from each data source along with the rationale for building the models is presented in the subsequent subsections below. The data presented in these sections has already been mapped to the FC-MD defect types. The original data along with the mapping can be found in Appendix A.

3.4.1. Requirements

Only data source B provided data on requirements inspections. The distribution of data on requirements defects is presented in Figure 1. Clarity and completeness constitute most of the defects found.

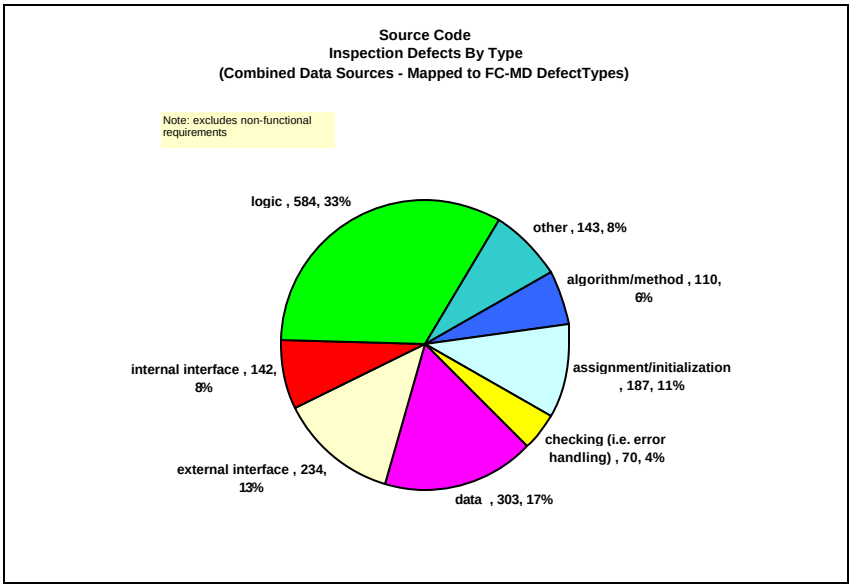
Figure 1. Data Source B - Requirements inspection defects



3.4.2. Source Code

All data sources contributed data on defects found during inspections of code artifacts. The distribution of the data for two of the data sources is dominated by non-functional defects (71%, 41%) (e.g., conformance to standards, performance, maintainability, etc.) and was omitted from the 3rd data source. Therefore, this category of defects is omitted in Figure 2 for the sake of comparing the defects across data sources.

Figure 2. Combined Data Sources - Source Code inspection defects



Data on source code inspections from data source A was divided into two categories: newly written source code and modified source code. The data from these two categories are shown in Figures 3 and 4. Note that the data from data source A did not include a "non-functional defect" category. While this data represents only one of our data sources, it is interesting to note the differences between Figures 3 and 4. A much larger proportion of the defects in modified source code are data-related and internal interface defects, as compared to the newly written source code. This makes some intuitive sense: One might suppose that categories such as algorithm and logic would be less likely to change during modification of a source code unit, than would issues related to data and component interfaces that are likely to be misunderstood by developers reusing code that was written elsewhere.

Figure 3. Data Source A - Source Code inspection defects

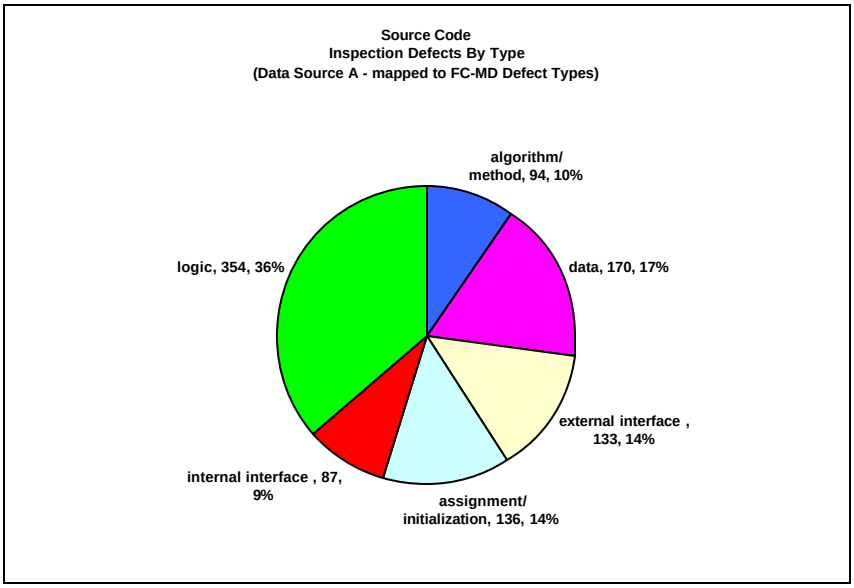
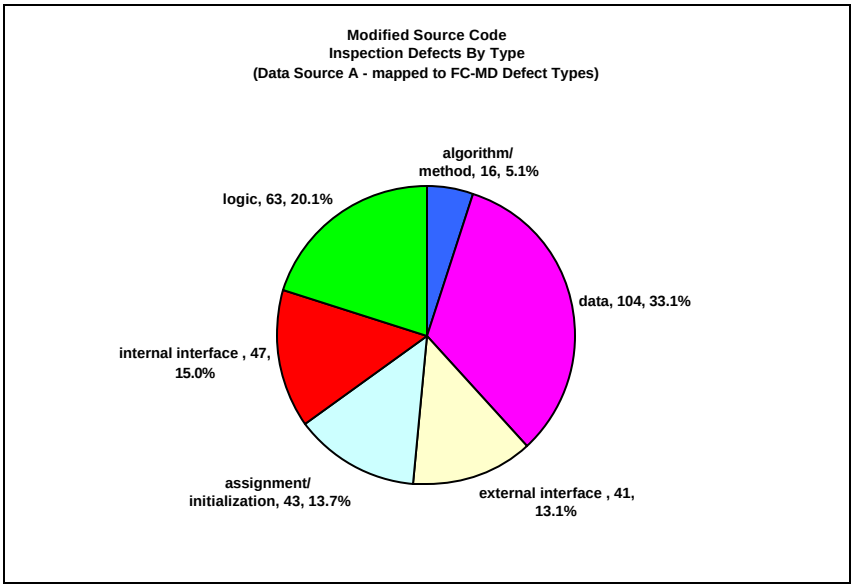
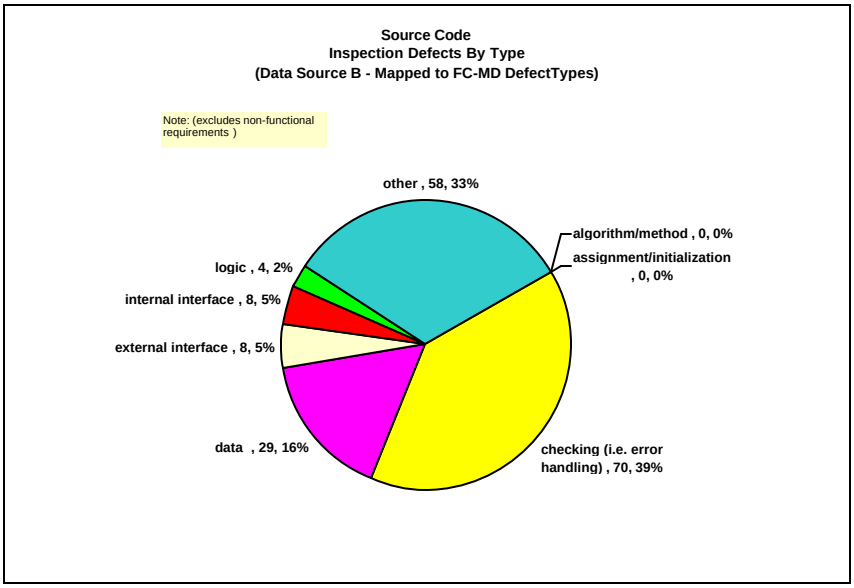


Figure 4. Data source A - Modified Source Code inspection defects



Data Source B did include a non-functional defects category, which constituted 71% of the defects reported. For this reason, the non-functional defect type has been excluded from Figure 5, which shows the distribution of the remaining 29% of the defects reported.

Figure 5. Data Source B - Source Code inspection defects



Data source C, like Data source A, also separated its data on source code inspections into two categories, one for modified source code and one for new source code. The defect data from these two categories are shown in Figures 6 and 7. As for data source B, a large percentage (41%) of source code defects from data source C were non-functional defects, so they have been excluded from the charts in Figures 6 and 7 so that the distribution of the remaining defects can be seen more clearly. Comparison of these two charts shows that most defects in modified source code were not classified into any defect category (i.e., they were "other" defects) while the new source code defects are dominated by the "logic" category.

Figure 6. Data Source C - Modified Source Code inspection defects

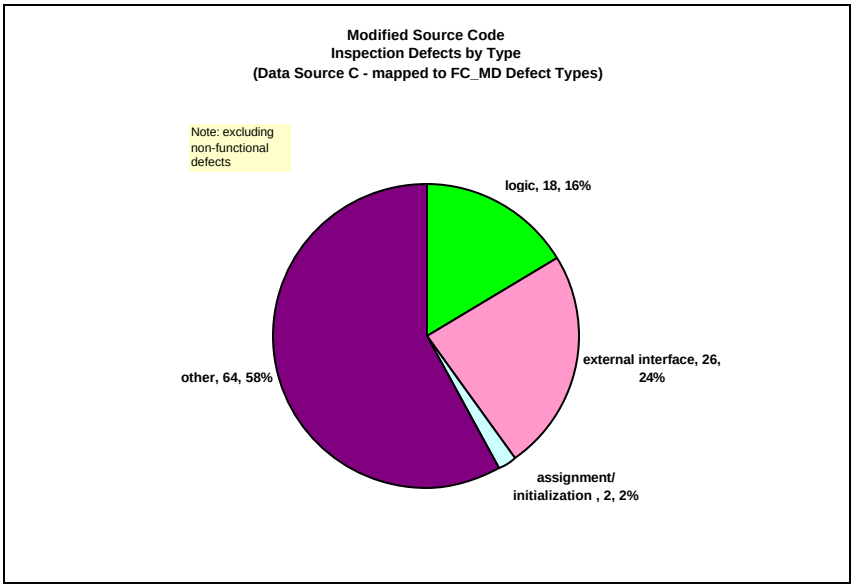


Figure 7. Data Source C - Source Code inspection defects

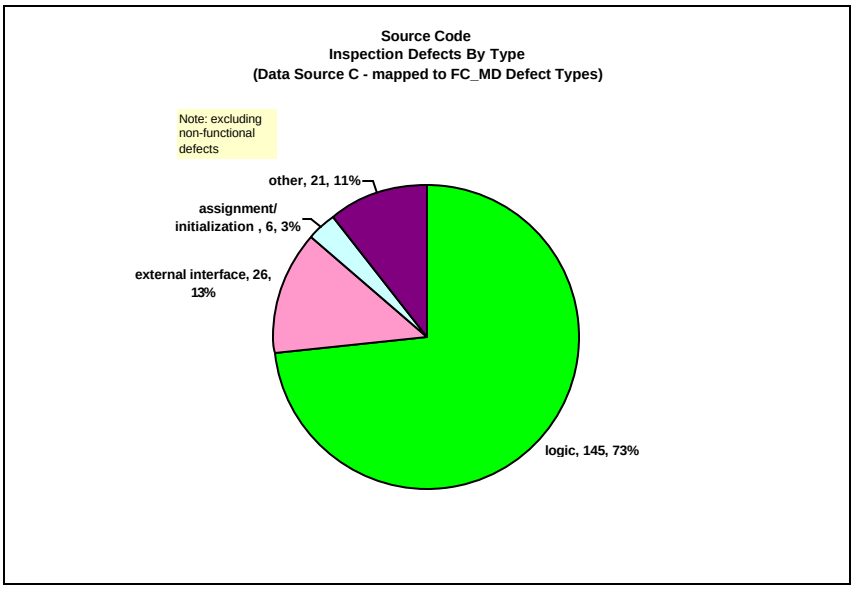


Table 7, below, compares the types of defects found in source code inspections from all data sources. The table shows that there are many more source code defects (1288) reported by data source A, compared with (177,308) the other data sources. There were also many more products inspected and reported by data source A (1643) compared with the other data sources (199 and 690). When combining all of the source code-related data, data source A's source code defect profile will tend to overshadow the contributions from the other data sources as previously shown in Figure 2, above. Scaling by size (e.g., number of products inspected) is one way to deal with this issue. Another item to note when looking at the varied profiles of source code inspection defect types is that for data source C there are no defects of type "data" reported. The "data" type defects may have been incorporated in the "other" category.

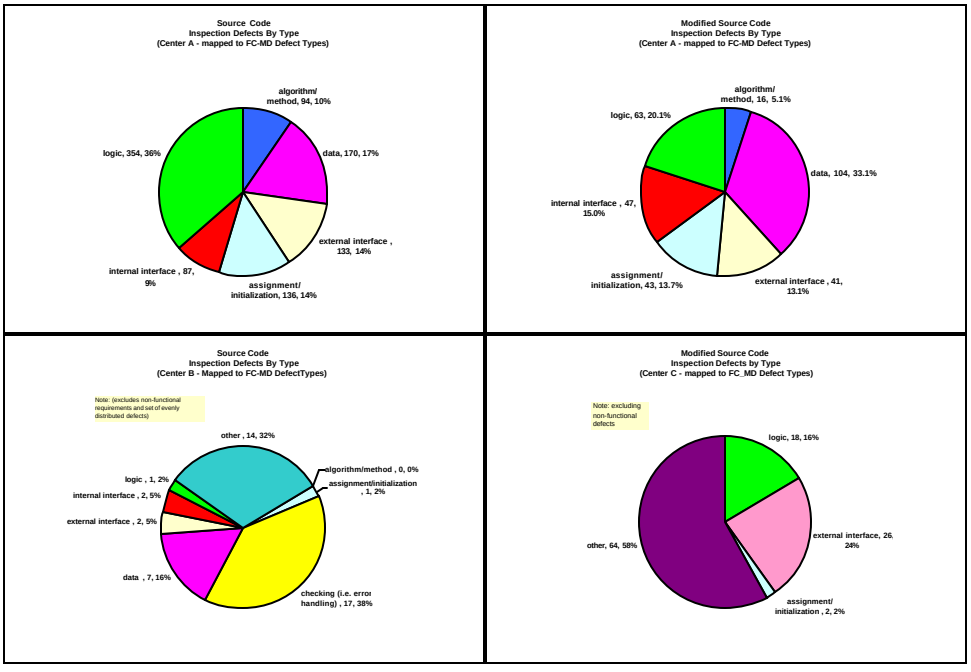
Table 7. Comparison of source code inspection defects by Data Source

Inspection Type	Defect Type	Data Source A (# defects)	Data Source B (# defects)	Data Source C (# defects)	Total (# defects)
Source Code	algorithm/ method	110	0		110
	assignment/ initialization	179	0	8	187
	checking (i.e., error handling)	0	70	0	17
	data	274	29		281
	external interface	174	8	52	228

	internal interface	134	8		136
	logic	417	4	163	581
	other	0	58	85	99
	Total	1288	177	308	1639
Inspection Type	Defect Type	Data Source A (% defects)	Data Source B (% defects)	Data Source C (% defects)	Total (% defects)
Source Code	algorithm/ method	8.5%	0.0%	0.0%	6.2%
	assignment/ initialization	13.9%	0.0%	2.6%	10.5%
	checking (i.e., error handling)	0.0%	39.5%	0.0%	3.9%
	data	21.3%	16.3%	0.0%	17.1%
	external interface	13.5%	4.7%	16.9%	13.2%
	internal interface	10.4%	4.7%	0.0%	8.0%
	logic	32.4%	2.3%	52.9%	32.9%
	other	0.0%	32.6%	27.6%	8.0%
	Total	100.0%	100.0%	100.0%	100.0%

Figure 8 shows the charts based on the data in Table 7 depicting the comparison of the types of defects by data source. As can be seen, there are large discrepancies in the data across Centers. Our next task is to explore whether some of these discrepancies can be explained by different attributes, such as size or type, of the projects represented in these data sets (as described in our research questions for future work in Section 6).

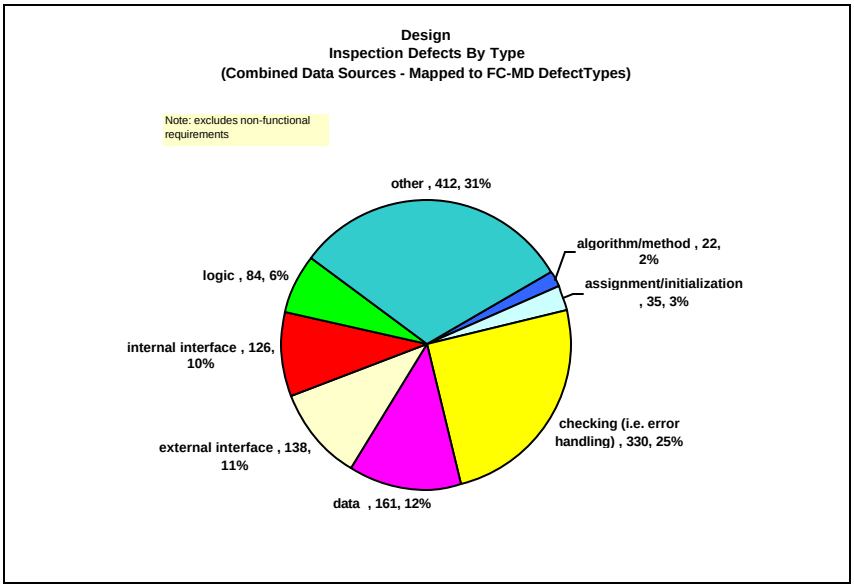
Figure 8. Comparison of Source Code inspection defects by Data Source



3.4.3. Design

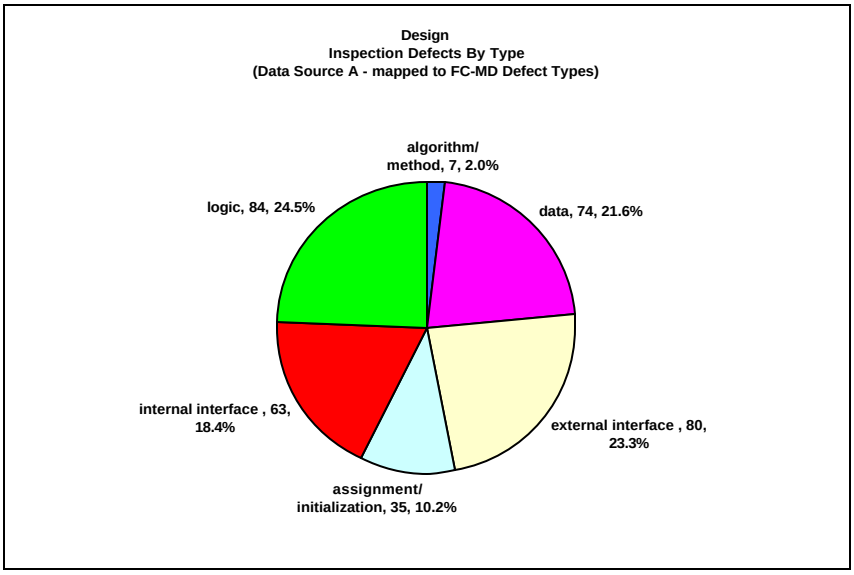
As with source code defects, a significant number of design defects (44% of the total) fell into a single category, “non-functional defects,” and so have been excluded from the chart in Figure 9 to better see the distribution of the remaining defects. A large number of the remaining defects are in the “other” category.

Figure 9. Combined Data Sources - Design inspection defects



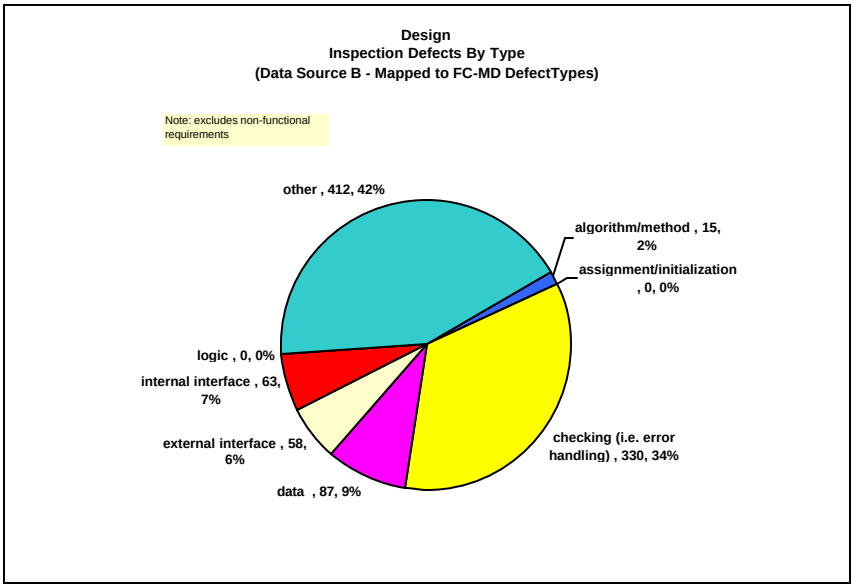
The distribution of design defects from Data source A are shown in Figure 10. Data source A did not use a “non-functional defects” category, or an “other” category.

Figure 10. Data Source A - Design inspection defects



Data Source B did include a non-functional defects category, which constituted 45% of the defects reported. For this reason, the non-functional defect type has been excluded from Figure 11, which shows the distribution of the remaining 55% of the defects reported, of which 42% are in the "other" category.

Figure 11. Data Source B - Design inspection defects



Data Source C supplied no data from design inspections.

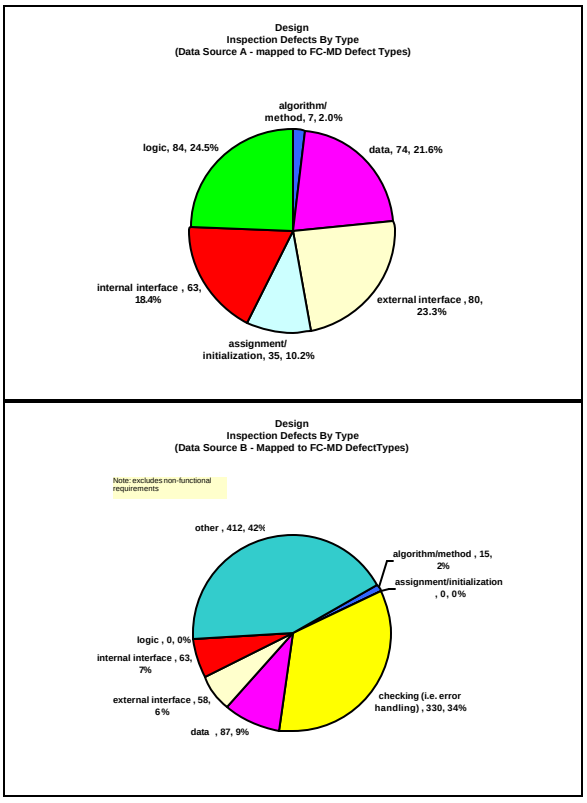
Table 8, below, compares the types of defects found in design inspections from all data sources, and Figure 12 shows the same information graphically. For the most part there are large discrepancies in the data across data sources. Once again, our next task is to explore whether some of these discrepancies can be explained by different attributes, such as size or type, of the projects represented in these data sets (as described in our research questions for future work in Section 6).

Table 8. Comparison of design inspection defects by data source

Inspection Type	Defect Type	Data Source A (# defects)	Data Source B (# defects)	Data Source C (# defects)	Total (# defects)
Design	algorithm/ method	7	15	0	22
	assignment/ initialization	35	0	0	35
	checking (i.e., error handling)	0	330	0	330
	data	74	87	0	161
	external interface	80	58	0	138
	internal interface	63	63	0	126
	logic	84	0	0	84

	other	0	412	0	412
	Total	343	965	0	1308
Inspection Type	Defect Type	Data Source A (% defects)	Data Source B (% defects)	Data Source C (% defects)	Total (% defects)
Design	algorithm/ method	2.0%	1.5%	0.0%	1.6%
	assignment/ initialization	10.2%	0.0%	0.0%	2.7%
	checking (i.e., error handling)	0.0%	34.2%	0.0%	25.2%
	data	21.6%	9.0%	0.0%	12.3%
	external interface	23.3%	6.0%	0.0%	10.6%
	internal interface	18.4%	6.5%	0.0%	9.6%
	logic	24.5%	0.0%	0.0%	6.4%
	other	0.0%	42.7%	0.0%	31.5%
	Total	100.0%	100.0%	0.0%	100.0%

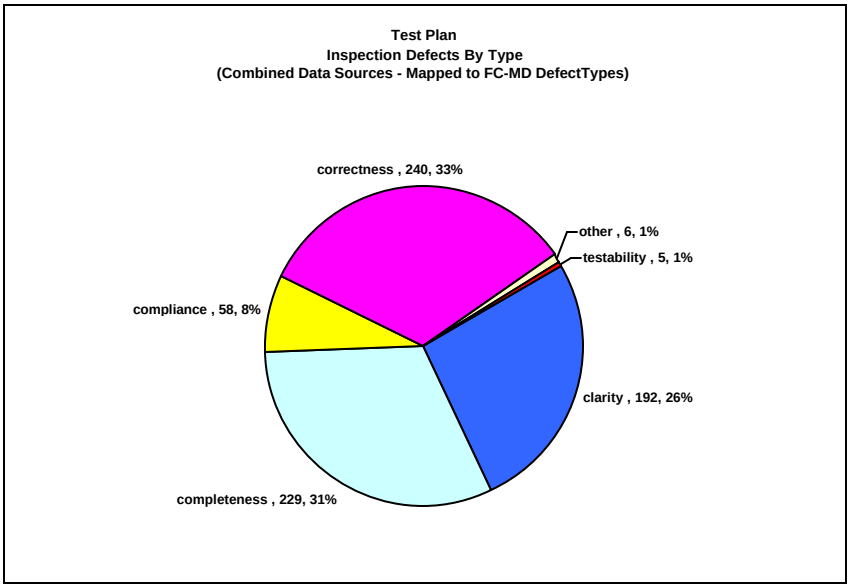
Figure 12. Comparison of Design inspection defects by Data Source



3.4.4. Test Plans

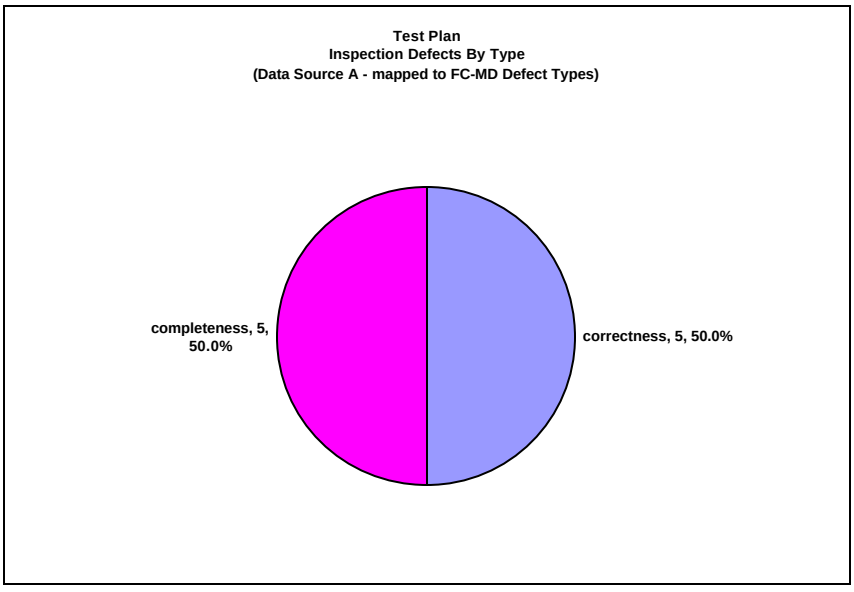
Correctness, completeness, and clarity dominated the distribution of defects found in inspections of test plans in the combined data set, which can be seen in Figure 13.

Figure 13. Combined Data Sources - Test Plan inspection defects



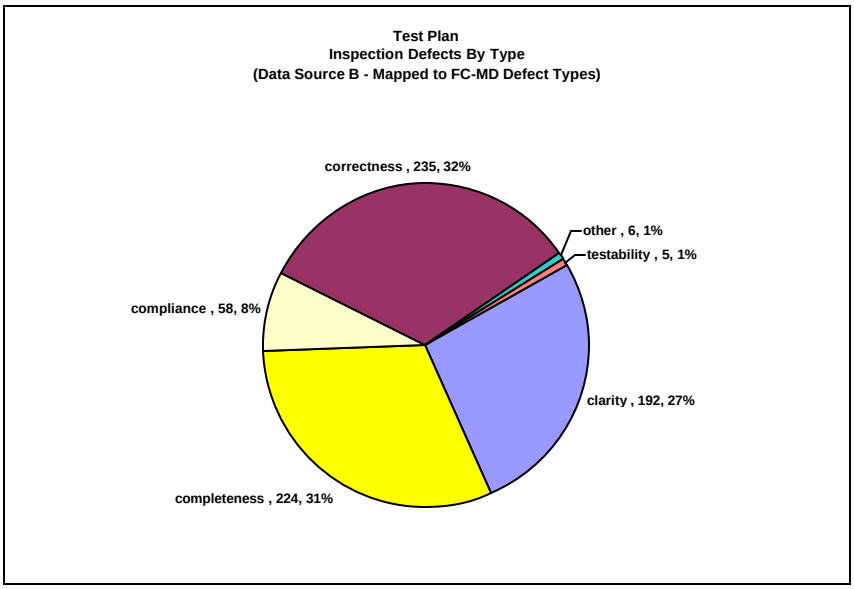
The test plan defect data from data source A fell into just two categories, as shown in Figure 14.

Figure 14. Data Source A - Test Plan inspection defects



Data Source B's test plan defect data, shown in Figure 15, is much more diverse.

Figure 15. Data Source B - Test Plan inspection defects



Data Source C supplied no data on test plan inspections. For Data Source A and B the results are widely different making it difficult to combine the data. Our future work will be to explore the reasons more fully by different attributes, such as size or type, of the projects represented in these data sets (as described in our research questions for future work in Section 6).

4. Inspection Effort, Size, and Defect Summary Models

A number of parameters are relevant for analyzing inspections and modeling the impact on a project's defects. Independent parameters (those under the control of the inspection planner) include the effort invested towards an inspection, the number of participants, the phase of development in which the inspection was conducted, the checklist used for the inspection, and document size. Other influencing factors are quality of the inspected artifact and the type of the project, which can be characterized by several criteria like safety, criticality, etc. Dependent variables include number of defects, type of defects, efficiency of finding those defects during an inspection and savings achieved by conducting an inspection. This section contains a description of the approach used to analyze existing effort and size type data as well as the status of analyzing the data collected from Data Source A, B, and C and some next steps for continuing this research initiative refining the summary effort, size, and defect models.

4.1. Types of data collected

The following types of data have been collected so far:

- Team size per inspection
- Size per inspection (pages for documents, SLOC for source code)
- Time to find defects (total hours per inspection/number of defects per inspection)
- Time to fix defects
- Major defects per page (or SLOC)
- Minor defects per page (or SLOC)
- Total defects per page (or SLOC)

4.2. Modeling Approach

The basic approach used so far has been to analyze the types of data (see immediately above) from each data source (A, B, C). The source data has not yet been combined and analyzed in depth because the data sources are from different environments and initially appear to be widely different. The initial analysis examined the mean and variability of the dataset for each product type inspected. As more data is provided in the future it is expected that these baselines will be refined. Also more analysis will be done that will allow for comparisons across projects, Centers and Agency. To provide some insight to our analysis approach and status so far, the next sub-section includes the initial baseline models from one of the data sources.

4.3. Sample Effort, Size, and Defect Summary Baseline Models

The Table below is a sample of the types of modeling that have been done under this research initiative. This table depicts the initial baseline models derived from Data Source B. In addition, the corresponding Figures follow the table. These Figures can also be viewed using the prototype tool, which is described in more detail in Section 5.

Table 9. Summary Baseline Models

Product Type	Size Per Inspection		Team Size Per Inspection		Time (Person Hrs.) to Find Defects		Time (Person Hrs.) to Fix Defects	
	Average	Range	Average	Range	Average	Range	Average	Range
Requirements Document	28.0 pages	(6 to 64)	6.5	(5 to 8)	35.4	(16 to 83.5)	19.8	(0.0 to 81)
Design Artifacts	34.8 pages	(6 to 102)	5.2	(2 to 8)	20.4	(1.8 to 59)	10.1	(0.0 to 47)
Code	42.9 SLOC	(8 to 100)	4.7	(3 to 7)	15.6	(4.5 to 46.8)	3.7	(0.0 to 22)
Test Artifacts	35.6 pages	(9 to 78)	4.6	(2 to 7)	20.2	(4 to 46)	10	(1 to 30)
Other	9.2 pages	(5 to 20)	5.3	(4 to 7)	15.6	(5.6 to 25.2)	20.4	(1.2 to 51)

Product Type	Total Defects / Page		Major Defects / Page		Minor Defects / Page	
	Average	Range	Average	Range	Average	Range
Requirements Document	1.6	(0.3 to 3.8)	0.3	(0.0 to 1.2)	1.3	(.1 to 2.7)
Design Artifacts	0.6	(0.0 to 2.2)	0.2	(0.0 to 1.2)	0.5	(0.0 to 2.2)
Code	0.4	(0.0 to 2.4)	0.04	(0.0 to 0.3)	0.4	(0.0 to 2.1)
Test Artifacts	0.7	(0.1 to 2.6)	0.3	(0.0 to 0.8)	0.4	(0.1 to 2.0)
Other	2.6	(0.7 to 5.4)	0.8	(0.0 to 2)	1.8	(.5 to 3.6)

Some issues to note in the following figures include:

- There are significant differences in the data ranges and means from one work product to another, for all parameters examined. For this reason, we treat the data for different product types separately in all of our analyses, as was reflected in this document.
- One interesting distinction was that for the team size parameter, the mean value for requirements documents was higher than for other document types and the range was fairly small. This corresponds to the intuition that more persons have a stake in the description of the system to be built, as defined by the requirements, and hence more people need to be represented in inspections of these documents.
- The data seem to show the difficulty of V&V in the early life-cycle phases, where it is more difficult to produce clear models of the system. Smaller documents were reviewed in the early life-cycle phases (requirements and design, as opposed to code) but more time was required by find and fix defects and a higher defect density was found.

Figure 16. Size data per Inspection by Product type

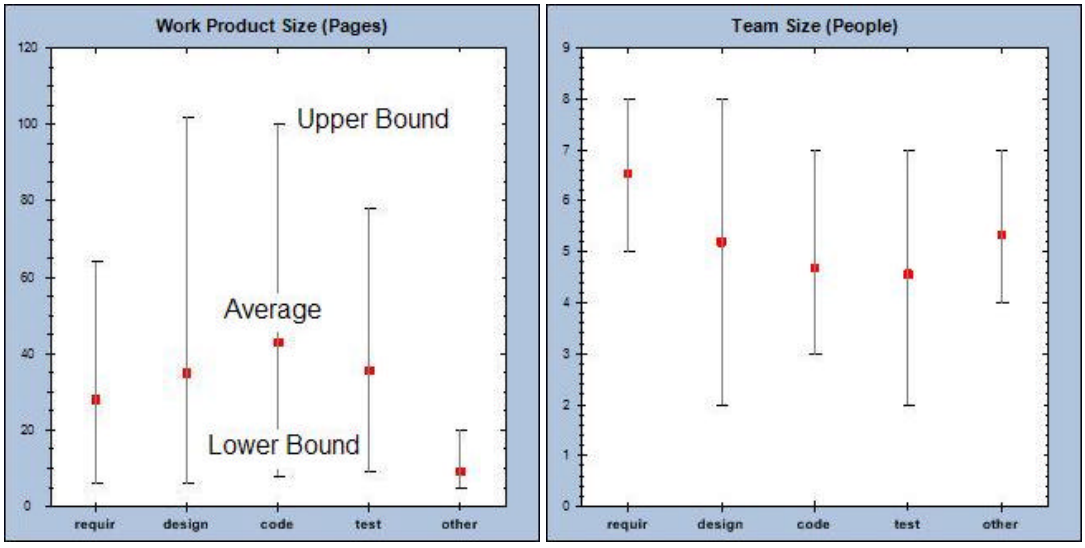


Figure 17. Effort data per defect by Product type

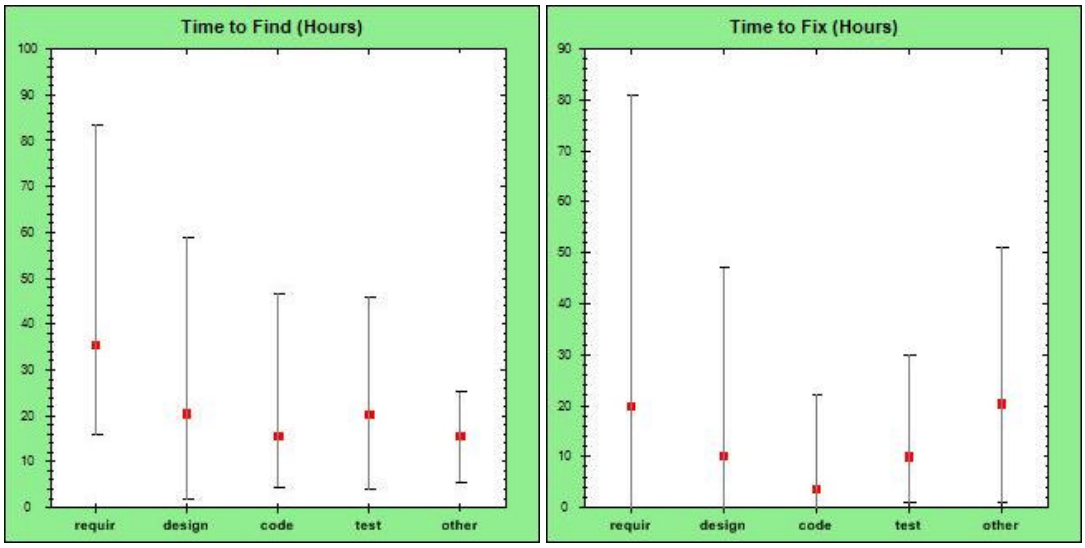
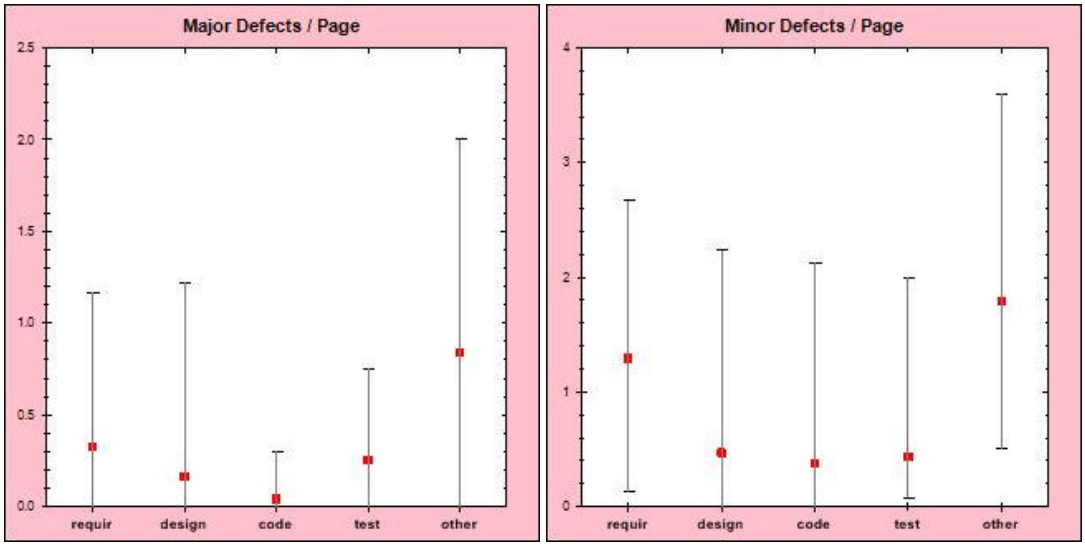


Figure 18. Defects found per size unit by Product type



4.4. Some Next Steps for Refining Effort, Size, and Defect Baseline Models

Future related work in this area will include:

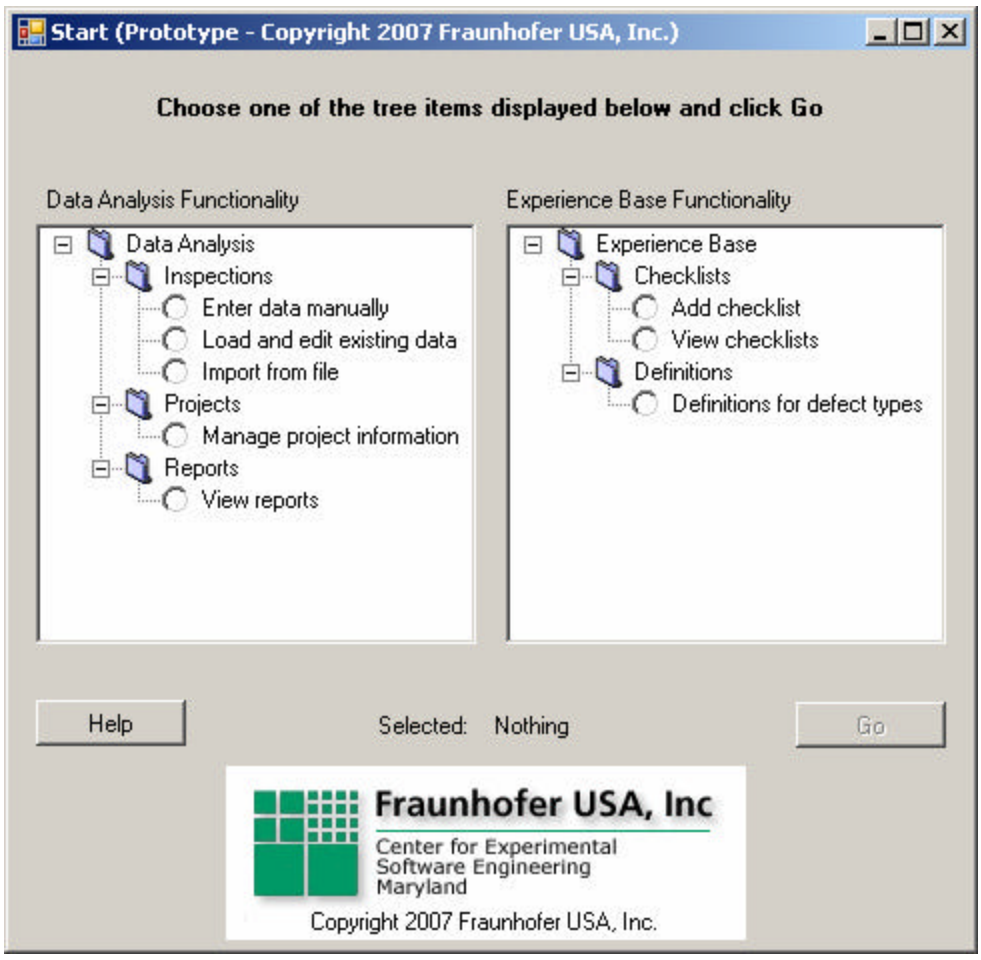
- building similar models with the other data we have received so far;
- continuing to compare new models with the recommendations, to determine whether the best practices reported in the literature hold in this environment and for contemporary programs;
- combining data across Centers and analyzing those models in more depth;
- collecting additional data from the same Center and validate/ refine the initial baseline models;
- obtaining feedback on the models and prototype tool and refine accordingly.

5. Prototype Inspection Tool Highlights

A number of analyses need to be performed on the different inspection datasets. To support these activities we developed a prototype tool called *Inspection Dashboard*. Our tool now automates many of the analyses described in the prior sections. The tool therefore supports the dissemination of the inspection results and offers a basis for detailed analysis and future improvements.

In the following sections, we will describe some highlights of the Dashboard tool. The tool is currently developed at the Fraunhofer Center, Maryland. For detailed documentation of the tool and its full functionality we refer the interested reader to [2].

Figure 19. The start-up screen of the Dashboard prototype



As illustrated in Figure 19, the Dashboard tool offers two groups of functions:

1) A number of Data analysis and handling capabilities (see functions in the left tree structure of the start-up screen in Figure 19), and

2) Experience Base (EB) capabilities (see functions in the right tree structure in the start-up screen in Figure 19).

5.1. Data Analysis Functionality

This set of functions form the main part of the Dashboard tool. With the offered functionality, users can store, analyze, and display inspection data. Dashboard offers several ways to input the collected data of conducted inspections. Whether users enter the collected data manually, or import them from an already existing file (in MS Excel™ format), the inspection data is systematically stored in the tool's own database.

The database of the Dashboard prototype initially has been seeded with several sets of historic inspection data from NASA. This data is not only used to test and demonstrate the tool's functionality, it further can serve as an initial baseline against which data of newly conducted inspections can be compared.

Note that in all following example screenshots of the dashboard tool only historic datasets together with other test datasets are illustrated. By design, these datasets are different from the project specific inspection data discussed in previous sections of this report.

Note that an additional feature of the tool is the ability to display recommended ranges for many of the key parameters, which are drawn from the existing literature and training courses. An important part of our future work is to examine whether there is sufficient evidence in our database to support these recommendations for NASA projects, and if not, whether we can produce more relevant, tailored recommendations.

Figure 20. Example: Project Reporting of the Dashboard prototype

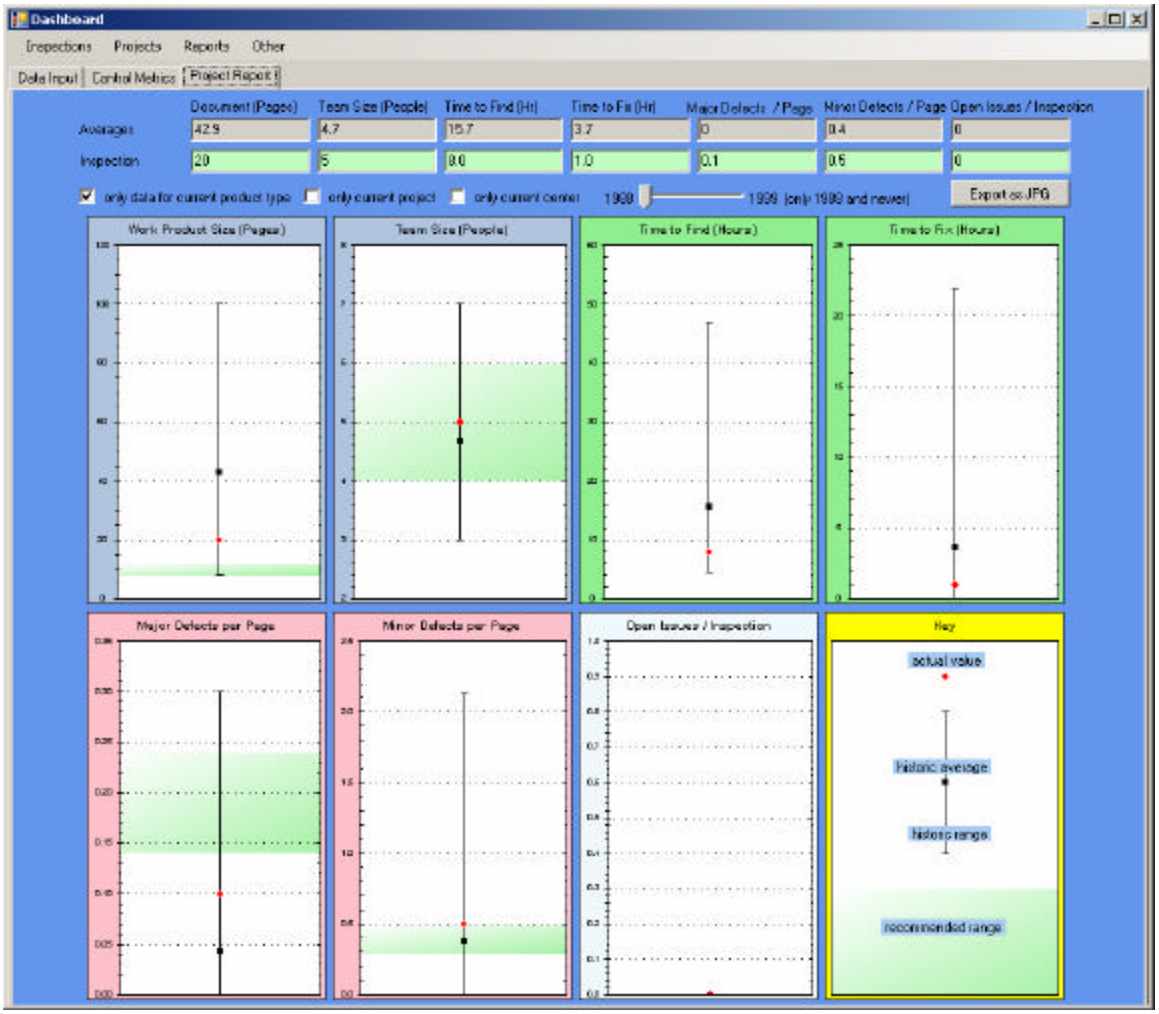


Figure 20 illustrates some of the reporting and display capabilities of the Dashboard prototype. Project specific inspection data can be displayed and compared to other projects. By using the different checkboxes and sliders in the upper part of the screen, users can choose against which existing inspection datasets in the Dashboard database the current project is compared. The current data point is displayed in contrast to the range of the selected inspection data as well as the average value of the selected datasets. Recommended ranges are also shown (in green) for comparison to incoming data. The complete screen can be captured and exported as a separate JPG file to be saved for future reference, or for inclusion in external reports.

Figure 21. Example: Control Metrics of the Dashboard prototype

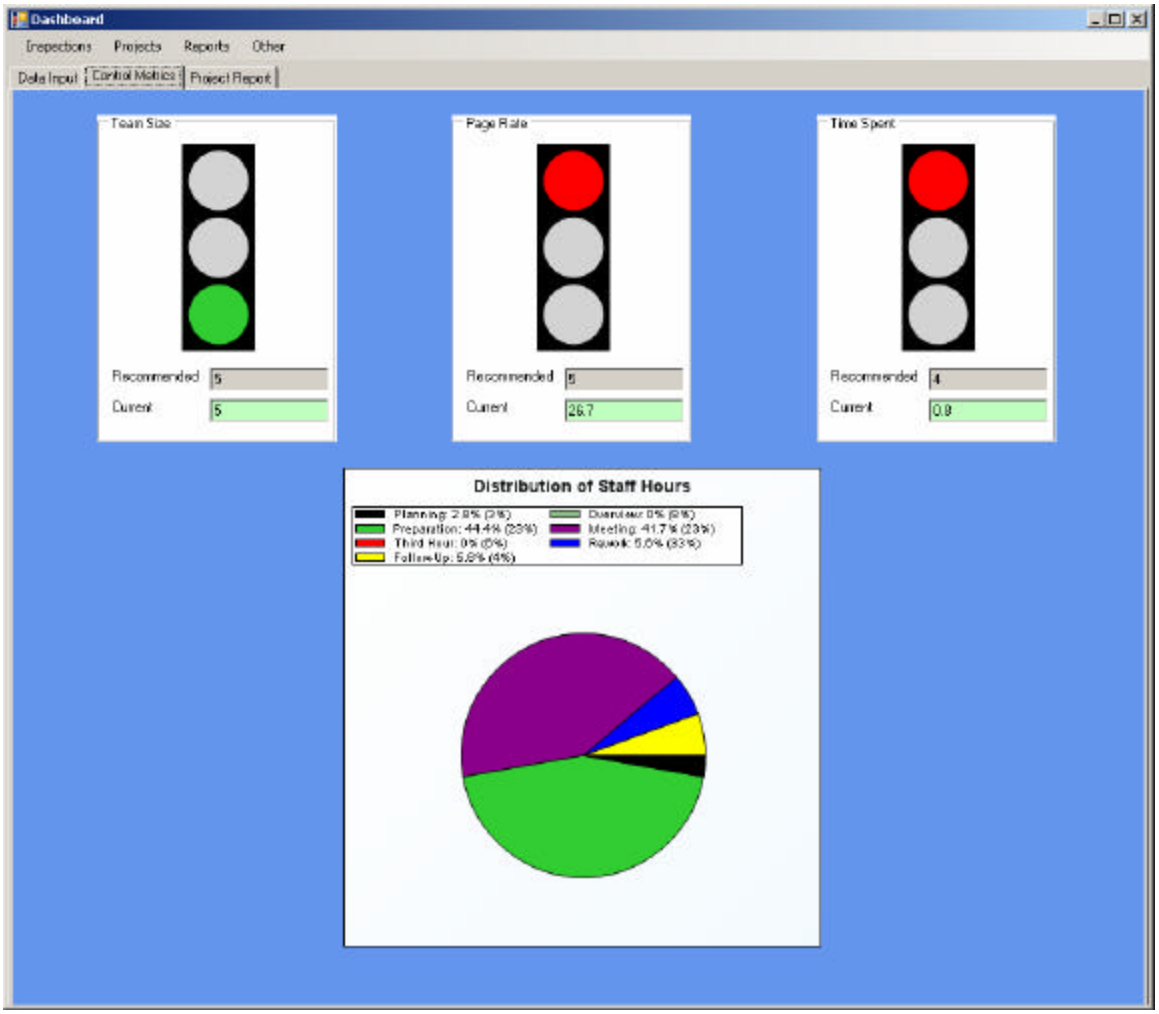


Figure 21 provides an example of the control metrics capability of the Dashboard prototype. The different traffic lights indicate whether the three major parameters that the inspection planner can control on a given inspection - "Team Size", "Page Rate", and the "Time Spent" - are within the predefined range of recommended values. In Figure 21, the current "Team Size" of 5 matches exactly with the recommended value of 5 team members. Hence, the traffic light is displayed in green. At the same time, the current values for the "Page Rate" and "Time Spent" are well above the recommended limits. Therefore, a red traffic light is displayed. Currently, the Dashboard prototype uses a set of control metrics based on historic inspection data, which we will evaluate against contemporary NASA inspection data during.

5.2. Experience Base Functionality

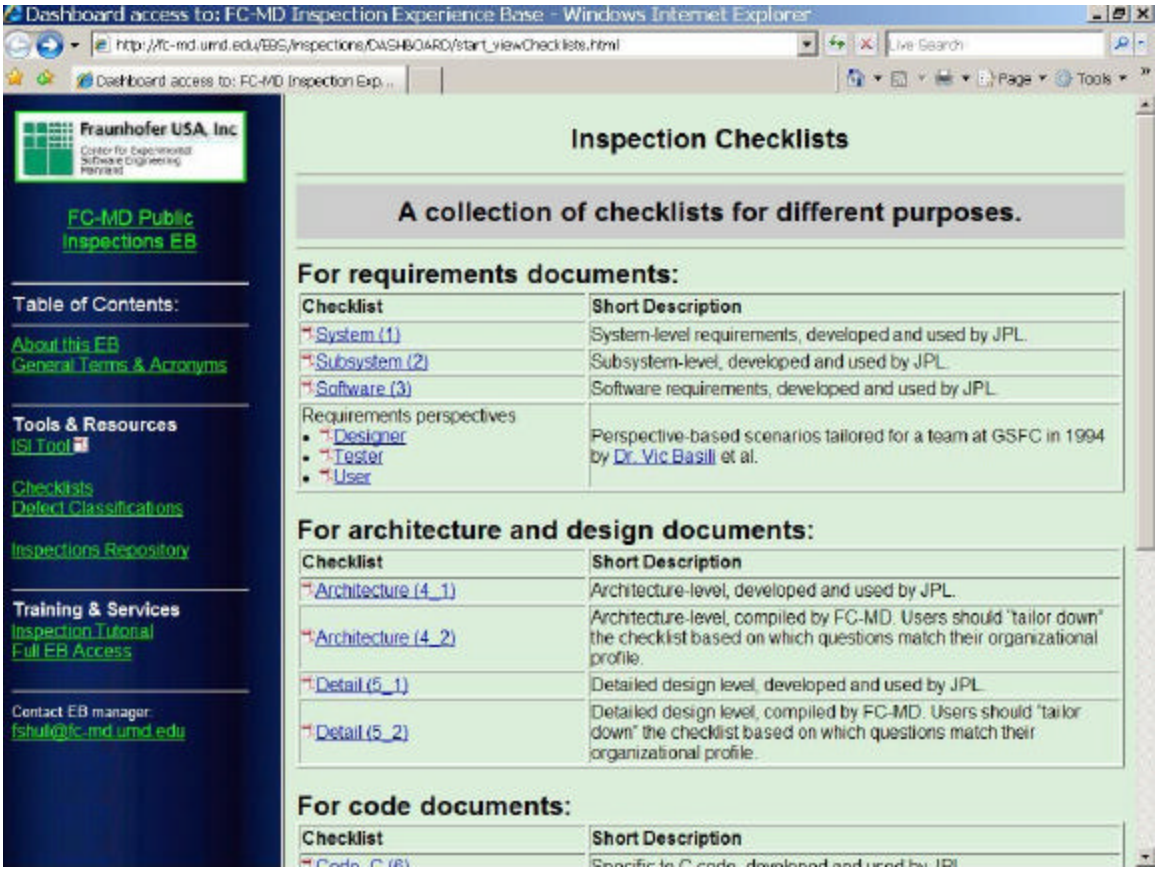
Besides the specific data storage, analysis, and display functionalities, which we highlighted in the last section, the Dashboard prototype also provides access to an Inspection Experience Base (EB).

The Experience Base will allow inspection planners to benefit from the past experiences of other projects that have applied inspections. Specifically, the EB will:

- Provide 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 come from elsewhere. Figure 22 shows one view of a user browsing for a checklist to use in a new inspection.

- Provide other related content that is updated periodically, such as definitions, defect taxonomies, related literature, etc.

Figure 22. Example: FC-MD Inspection Experience Base



6. Research Questions for Further Study

As emphasized earlier, this model is a preliminary one that will continue to be refined and updated as we gain access to additional data sources. So far it has allowed us to characterize key parameters in the inspection process (such as inspection preparation effort, inspection meeting length, inspection rate, type of document, inspection team size, and type and number of defects found).

In future work on this initiative we need to refine this model via the following analyses:

Research Goal 1: Characterize the effects of inspection process used, inspection preparation effort, inspection meeting length, inspection rate, type of document, and inspection team size on defects found (number, severity, defect types) and rework effort over a set of projects.

Questions:

A. What is the effect of parameters that can be controlled in an inspection (metrics i through vi, below) on inspection outcomes (metrics vii through x)?
B. Are there any interaction effects of metrics i through vi on any of the metrics vii through x? (E.g.: Does inspection rate have an effect that varies for different types of documents?)
C. Are there any combinations of project characteristics (e.g. project size, project criticality, etc.) for which some of these effects hold or do not hold?
D. For inspections for which metrics i through vii are within the recommended range, are significantly better values of metrics vii through x observed? If not, can ranges be defined for which this is true?

Metrics:

i. inspection process used
ii. inspection preparation effort
iii. inspection meeting length
iv. inspection rate
v. type of document
vi. inspection team size
vii. number of defects found
viii. severity of defects found
ix. defect types of defects found
x. rework effort

Research Goal 2: Characterize the relationship between the inspection process and the test process with respect to effort expended and defects found over a set of projects.

Questions (NOTE: These questions are basically a cross product of [characteristics of defects found in inspections, size of inspection effort, inspection rework effort] X [characteristics of defects found during test, test effort, test-related rework effort]):

A. Do differences in the defects found (number, severity, defect types) during inspections have an effect on the defects later found (number, severity, defect types) during test on the same project?
B. Do differences in the defects found (number, severity, defect type) during inspections have an effect on the amount of test effort expended later on the same project?
C. Do differences in the amount of the inspection effort (preparation effort, meeting length, inspection rate, size of inspection team) have an effect on the defects later found (number, severity, defect types) during test on the same project?
D. Do the answers to any of the above questions depend on the phase in which the inspection takes place (i.e. the document inspected)?

Metrics:

i. Number of defects found per inspection
ii. Severity (major/minor) of defects found per inspection
iii. Defect type of each defect found per inspection
iv. Inspection preparation effort
v. Inspection meeting length
vi. Inspection rate
vii. Size of inspection team
viii. Test effort

7. References

[1] R. Chillarege et al.: “Orthogonal Defect Classification—A Concept for In-Process Measurements,” IEEE Trans. Software Eng., vol. 18, no. 11, 1992, pp. 943–956.

[2] R. Haingaertner: Master Thesis. Mannheim University of Applied Science, Mannheim, Germany & Fraunhofer USA Inc., Center for Experimental Software Engineering, Maryland, scheduled October 2007.

Appendix A: Original Source Data By Product and Defect Type

Table A1. Data Source A detailed defect data

Modified Source Code		
Data Source A	Number of Defects	Mapped to FC-MD Defect Types
Computational	16	algorithm/ method
Data value or structure	104	data
External interface	41	external interface
Initialization	43	assignment/ initialization
Internal interface	47	internal interface
Logic/control structures	63	logic
Total	314	
New Source Code		
Data Source A	Number of Defects	Mapped to FC-MD Defect Types
Computational	94	algorithm/ method
Data value or structure	170	data
External interface	133	external interface
Initialization	136	assignment/ initialization
Internal interface	87	internal interface
Logic/control structures	354	logic
Total	974	
Design		
Data Source A	Number of Defects	Mapped to FC-MD Defect Types
Computational	7	algorithm/ method
Data value or structure	74	data
External interface	80	external interface
Initialization	35	assignment/ initialization
Internal interface	63	internal interface
Logic/control structures	84	logic
Total	343	

Test Plan		
Data Source A	Number of Defects	Mapped to FC-MD Defect Types
Computational	0	half to correctness and half to completeness
Data value or structure	3	
External interface	0	
Initialization	1	
Internal interface	2	
Logic/control structures	5	
Total	11	

Table A2. Data Source B detailed defect data

Requirements		
Data Source B	Defects	FC-MD mapping
Clarity	250	clarity
Completeness	69	completeness
Compliance to Stds	17	compliance
Consistency	102	consistency
Correctness	49	correctness
Maintainability	44	completeness
Modularity	0	
Performance	6	completeness
Portability	0	
QUALIFI	0	
Reliability	8	completeness
Traceability	36	compliance
Level of Detail	36	clarity
Accuracy	12	correctness
Anomaly Management	0	
Computation	0	
Data	20	correctness
Feasibility	5	other
Functionality	54	other
Interface	16	completeness
Testability	20	testability
Other	1	other
Grand Total	745	

Design		
Data Source B	Defects	FC-MD mapping
Defect	Number of defects	
Clarity -	479	non-functional defects
Completeness -		276 dist by % profile to all categories
Compliance to Stds -	84	non-functional defects
Consistency -		169 dist by % profile to all categories
Correctness -		321 dist by % profile to all categories
Maintainability -	23	non-functional defects
Modularity -	1	other
Performance -	25	non-functional defects
Portability -	9	non-functional defects
QUALIFI	7	other
Reliability -	65	checking (i.e., error handling)
Traceability -	20	non-functional defects
Level of Detail -	32	non-functional defects
Accuracy -	0	dist to algorithm
Anomaly Management -	1	checking (i.e., error handling)
Computation -	2	algorithm/method
Data -	18	data
Feasibility -	1	algorithm/method
Functionality -	92	non-functional defects
Interface -		23 dist evenly internal and external interface
Testability -	2	checking (i.e., error handling)
Other -	77	other
	12	internal interface
	11	external interface
	766	all categories dist by designated % profile
Grand	1727	

Source Code		
Data Source B	Number of Defects	FC-MD Mapping
Clarity	310	non-functional defects
Completeness		40 dist by % profile to all categories
Compliance to Stds	26	non-functional defects
Consistency		19 dist by % profile to all categories
Correctness		75 dist by % profile to all categories
Maintainability	40	non-functional defects
Modularity	0	other
Performance	26	non-functional defects
Portability	0	non-functional defects
QUALIFI	0	other
Reliability	7	checking (i.e., error handling)
Traceability	3	non-functional defects
Level of Detail	4	non-functional defects
Accuracy	0	dist to assign, data, logic
Anomaly Management	8	checking (i.e., error handling)
Computation	0	algorithm/method
Data	6	data
Feasibility	0	algorithm/method
Functionality	31	non-functional defects
Interface		Put 2 in internal interface and 2 in external interface

Testability	2	checking (i.e., error handling)
Other	14	Other
Completeness, correctness, consistency	1	assignment/initialization
Completeness, correctness, consistency	1	data
Completeness, correctness, consistency	1	logic
Completeness, correctness, consistency		algorithm/method
Completeness, correctness, consistency	2	internal interface
Completeness, correctness, consistency	2	external interface
	134	distributed by specific % profile to all categories from above
Total	618	
Test Plan		
Data Source B	Number of Defects	FC-MD Mapping
Modularity	1	other
Maintainability	2	completeness
Other	2	other
QUALIFI	3	other
Reliability	4	completeness
Testability	5	testability
Compliance to Stds	7	compliance
Performance	7	completeness
Accuracy	7	correctness
Level of Detail	27	clarity
Consistency	44	correctness
Traceability	51	compliance
Clarity	165	clarity
Correctness	184	correctness
Completeness	211	completeness
Total	720	

Table A3. Data Source C detailed defect data

Source Code (mainly new code)		
Data Source C	Number of Defects	FC-MD Mapping
Logic	145	logic
Usability	26	external interface
Unset	6	assignment/initialization

Coding Standard	79	non-functional
Clarify	52	non-functional
Suggestions	80	non-functional
Other	21	other
Total	409	
Source Code (mainly reused code)		
Data Source C	Number of Defects	FC-MD Mapping
Logic	18	logic
Usability	26	external interface
Unset	2	assignment/initialization
Coding Standard	48	non-functional defects
Clarify	52	non-functional defects
Suggestions	43	non-functional defects
Other	64	other
Total	253	

Appendix B: Acronyms, Common Terms, and Definitions

B.1. Acronyms

The following list of acronyms is alphabetically sorted.

- CMMI Capability Maturity Model Integrated
- DB Database
- EB Experience Base
- EF Experience Factory
- FAQ Frequently Asked Question
- FC-MD Fraunhofer Center for Experimental Software Engineering, Maryland
- GSFC Goddard Space Flight Center
- GQM Goal-Question-Metric (paradigm / approach)
- IT Information Technology
- JPL Jet Propulsion Laboratory
- KM Knowledge Management
- LL Lesson(s) Learned
- LOC Lines Of Code
- MOU Memo of Understanding
- NASA National Aeronautic and Space Administration l
- NPR NASA Procedural Requirements
- ODC Orthogonal Defect Classification
- SARP Software Assurance Research Program
- SE Software Engineering
- SEL Software Engineering Laboratory
- SEPG Software Engineering Process Group
- SLOC Source Lines Of Code
- SW Software
- V&V Verification and Validation

B.2. Definition of Common Terms

The following list of terms is alphabetically sorted. An (→) inside the text of a definition indicates that a separate definition is available for the term following the symbol.

- **Algorithm/method (→) defect type:**
  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 (→) defect type**:
  A variable or data item that is assigned a value incorrectly or is not initialized properly or where the initialization scenario is mishandled.
- **Checking (→)defect type:**
  Inadequate checking for potential error conditions
- **Clarity (→)defect type:**
  A problem in the wording or organization of the document that makes it difficult to understand.
- **Completeness (→) defect type:**
  A missing requirement or test case or other piece of information.
- **Compliance (→) defect type:**
  A problem with compliance to any relevant standard.
- **Consistency (→) defect type:**
  Two or more statements in the document that are not consistent with each other, e.g., two test cases that should produce equivalent output have different expected outputs or requirements that are mutually exclusive.
- **Context:**
  The term context describes an environmental setting in which an object (e.g., a document, a process model, or a program) or real-world entity is applicable.

  Note: To describe a certain context in a formal way attribute-value pairs are often employed. A set of such attribute-value pairs relevant for a certain context is often referred to as a *context vector*.

  Examples: For describing the application context of a program one could use, for instance, the following context vector:
  <(operating system, Windows 2000), (required memory, 256 MB)>
- **Correctness (→) defect type:**
  Any statement in the document that is incorrect, including incorrect expected output for a test case.
- **Data (→) defect type:**
  Error in specifying or manipulating data items, incorrectly defined data structure, pointer or memory allocation errors, or incorrect type conversions.
- **Defect Type:**
  The categories of defects uncovered during inspections of artifacts such as requirements documents, source code, test plans, etc.

- **Experience:**
  Experience is gained by humans through utilization of information or knowledge.

  Note: The differences between knowledge and experience are:
  … that only humans can gain experience (not an intelligent system), and
  … the fact that experience is utilized.

  The second point stresses that one explicitly makes use of the given information or knowledge and experiences the results (i.e., one does not only have to believe what others learned or interpreted). As a consequence, when experience is documented, it becomes knowledge for all others (e.g., researchers or other companies) who/that have not experienced it on their own.

  Example: The best example for experience is a lesson learned (LL) which I (as a person or a company) learned (i.e., experienced) myself.

- **External Interface (→) defect type:**
  Errors in the user interface (including usability problems) or the interfaces with other systems.

- **Internal Interface (→) defect type:**
  Errors in the interfaces between system components, including mismatched calling sequences and incorrect opening, reading, writing or closing of files and databases.

- **Logic(→) defect type:**
  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 (→) defect type:**
  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.

- **Redundancy (→) defect type:**
  Test cases or other information that is not necessary because it appears more than once.

- **Testability(→) defect type:**
  A test case that is not feasible.

- **Timing/optimization (→) defect type:**
  Errors that will cause timing (e.g., potential race conditions) or performance problems (e.g., unnecessarily slow implementation of an algorithm).