How to Measure the Impact of Specific Development Practices on Fielded Defect Density

Ann Marie Neufelder
Owner, SoftRel
Softrel@ix.netcom.com

Abstract

This author has mathematically correlated specific development practices to defect density and probability of on time delivery. This paper summarizes the results of this ongoing study that has evolved into a software prediction modeling and management technique.

The author has collected data from 45 organizations developing software primarily for equipment or electronic systems. Of these 45 organizations, complete and unbiased delivered defect data and actual schedule delivery data was available for 17 organizations. The author will present in this paper the mathematical correlation between the practices employed by these organizations and defect density.

This correlation can and is used to:
a) Predict defect density
b) Improve software development practices for the best return on investment.

1. Introduction

In 1992, the USAF Rome Laboratories produced Software Reliability, Measurement, and Testing Software Reliability and Test Integration RL-TR-92-52 [1]. This document was one of the first publicly available documents to quantify the relationship between certain development practices and defect density. Some of the weaknesses of this document are:

1. Several of the factors are weighted equally
2. Some of the development factors are difficult to quantify. Examples include "experience level above average" where "average" is not quantitatively defined.
3. Some of the development factors are compiler dependent. Examples include thresholds for "Source Lines of Code (SLOC) per module". SLOC thresholds vary from one compiler to another and from object oriented compilers to procedural compilers, yet this was not taken into account.
4. Since the document was written in 1992, newer technologies such as object oriented programming and incremental life cycle models are not addressed.
5. The document is directed to defense contractors and is difficult to apply to commercial software organizations.

Even with the above weaknesses the document proved to be a reasonable starting point for selecting parameters that would ultimately correlate to defect density. The author discarded parameters that were outdated or difficult to measure consistently.

In 1994, this author selected 27 software development parameters to be included in this model, many of which were derived from the Rome Laboratories document. While the Rome Laboratory model had weighted each of these parameters equally, the goal of this study was to determine individual parameter weights. For example, this study found that test beds correlated more strongly to defect density then random testing. Later in this paper, the parameters that were extracted from the Rome Laboratory document will be illustrated in Tables 7 through 11.

Seven software organizations were evaluated quantitatively based on these 27 parameters. [2]

The work continued and by 1997, there were a total of 14 software organizations evaluated and the list of correlated parameters had grown to 102 not including 21 information only questions that do not contribute to a score. [3]

The author was able to include the seven organizations that had originally been evaluated earlier because those evaluations were so detailed that the information required for the additional 75 parameters was well documented and available.

The 75 additional parameters transpired because of:

1. Newer technologies and tools became available
2. Interviews with the organizations that exhibited the lowest and highest defect densities in an effort to determine the major differences between them.

The data collected is assumed to be unbiased because:

1) One person (the author) evaluated the practices for each of the organizations using the same criteria.
2) The author was intimately familiar with each organization and their "actual" versus "wish list" practices.
3) The author required each organization to provide physical proof of all positively answered questions.
4) To avoid documentation of overly optimistic or pessimistic responses, the author required inputs from a wide cross section of employees such as managers, lead engineers, quality engineers, test engineers, engineers with an average level of experience and new hires. It is interesting to note that the author encountered pessimism as often as optimism.

The "actual" defect data was also calculated by the author and normalized so as to represent similar defect severities from one organization to another. The author also normalized the KSLOC to be in assembler since there were different languages represented in the study.

Actual defect density was computed on projects that had been delivered using the corresponding practices measured in the questionnaire. The defect density was computed by plotting observed failure intensity (x axis) versus observed cumulative failures (y axis). The y intercept of this line is the theoretical number of inherent defects. It is theoretical because it is not possible to know with complete certainty when there are no defects left in the software. For the project data to be considered for this study, at least 95% of the estimated inherent defects must be observed or known. These criteria ensured that defect density was measured the same way on every sample.

In late 1999, the model was updated once more to include data from 3 additional organizations. [4] The author continues to refine the model as data becomes available and as technology and development practices continue to change. This paper illustrates the most recent results.

2. The results

In this authors experience, software managers and engineers often assume that engineering practices that decrease defects will also increase development time. For the companies in this study, that assumption was simply not true. As shown in Table 1, the organizations with the best practices had the highest probability of making their schedules. When they did miss their schedule the magnitude of error was significantly smaller then for the companies without good practices.

Table 1 illustrates that as score increases, the average number of corrective action releases decreases. For the organizations included in this study, more corrective action releases mean more downtime for customers.

Some of the organizations benchmarked were striving for more releases when in the results of their customer surveys showed that their customers wanted fewer releases with higher quality. For 9 of the organizations

Score on study	Classification	Average defect density in defects per Assembler KSLOC	Average probability of late delivery	Average margin of error on late delivery as a % of original schedule	Average number of corrective action releases per year	Average SEI CMM level
700+	Best Practices	.14	30%	10%	3.67	2.1
300-699	Good Practices	.48	66%	88%	6	1.2
100-299	Moderate Practices	.96	82%	215%	8.5	1
< 100	Least Practices	1.35	88%	138%	14	1

Table 1. Summary of results (Copyright SoftRel, 2000, reprinted with permission)

benchmarked in this study, the release time desired by the customers was quarterly. The objective release frequency varies from application to application, however, and should be determined via appropriate survey methods.

Table 1 also shows that the average SEI CMM ™ level [5] increased as the score increased. The highest SEI CMM level in this study was 2.5. While there is a relationship between the score on this study and the SEI CMM level, the questionnaire used in this study is **not** derived from the SEI CMM evaluation questionnaire.

3. About the organizations measured

The sizes of the software organizations (software personnel only) ranged from 5 to 60. The industry breakdown for those organizations for which defect data was available was:

Defense systems	5
Process control manufacturing equipment	11
Mathematical software	1

The software systems were between 100 KSLOC and 1000 KSLOC. C++ was the language of choice with 2 exceptions. One organization used Smalltalk and one used Pascal.

The organizations with the highest scores and lowest defect densities had this in common:

- Software engineering is a part of the **engineering** process and is not considered to be an art form by anyone in the organization
- They believe in a well-rounded sound set of development practices as opposed to a "single bullet".
- Formal and informal reviews of the software requirements prior to proceeding onto design and code.
- Testers are involved in the requirements translation process.

The organizations with the lowest scores had this in common:

- Lack of software management.
- Misconception that programmers know what the customer wants better then the customer does.
- An inability to focus on the requirements and use cases with the highest occurrence probability
- Complete void of a requirements definition process
- Insufficient testing methods

4.0 The scoring mechanism

4.1. How the score was developed

The score was developed by mathematically correlating the observed practices of these 17 companies versus the observed defect density in terms of KSLOC of assembler. The score was a composite of scores on each of these areas.

The questionnaire has 5 sections.

- Organization commitment (see Table 7)
- Life cycle practices (see Table 8)
- Product characteristics (see Table 9)
- Change control practices (see Table 10)
- Informational questions that currently do not have a correlation to defect density (see Table 11)

Tables 7 to 11 illustrate the single parameter correlation, the maximum points possible for each question and an indication of whether the parameter was part of the Rome Laboratory model discussed in section 1.

Important note: A negative correlation is expected when correlating practices to defect density. A correlation of –1 means that the practice perfectly correlates to lower defect density. A correlation of +1 means the practice perfectly correlates to higher defect density. A correlation of 0 means no correlation at all.

This questionnaire has 102 questions and was used to evaluate the practices of the organizations in this study. These questions were developed and continue to be refined by the below process:

- Review practices that had already been mathematically correlated by the USAF Rome Laboratories TR-92-52.
- Study organizations that were at the top of their industry or application type for software deliveries.
- Study organizations that were at the bottom of their industry or application type for software deliveries.
- Ask the customers of these software organizations what key factors they felt impacted software reliability and investigate.

The author developed the score mechanism described in this paper with this process:

1. Select parameters that are likely to correlate directly to higher or lower defect densities for **any** organization and not just one specific organization.
2. Make sure that each parameter is measurable. An example of a parameter that is difficult to measure is a "programmer's individual capabilities". This author has not found a consistent way to measure an individual's capabilities that can be generically applied to any organization or individual. However, this author can measure the capabilities of the organization as a whole. (i.e. the SEI CMM level).
3. Determine if each single parameter correlates to the empirical defect densities for each of the samples.
4. Drop any parameters that do not correlate (but keep that information as it is possible that that parameter may correlate at a later time when the sample size is larger.)
5. If the parameter correlates then determine it's relative weight by using its relative correlation as a seeding value. Find the best weight by maximizing scoring algorithm vs. defect density correlation and solving for the weight for each single parameter. It should be noted that while the single parameter correlation was used as a seeding value to determine the scoring weight, the score for the parameter is not necessarily directly or linearly related to the correlation.
6. Repeat steps 1 to 6 when more samples are available with complete data.

4.2. Scores versus defect density

The relationship between empirical scores and empirical defect density that maximized the correlation between the observed practices and the observed defect density was determined to be: [8]

DD = 0.000001x^2 - 0.002531x + 1.394135

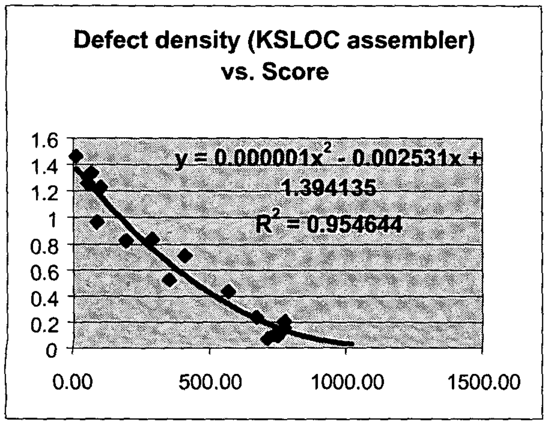

Figure 1. Actual scores vs. actual defect Density

Where x is the score and DD is the defect density on delivery day.

The units for defect density are defects per 1000 source lines of **assembler** code. To determine the defect density in terms of another language, the code expansion ratio of that language is multiplied by the above defect density prediction. [6]

This defect density is then multiplied by the size of the project to determine the inherent number of defects in the software (N_0). The fielded failure rate can be predicted by estimating the ratio (Q) between inherent defects and failures per time based on historical data.

Q is estimated by $(\lambda_2 - \lambda_1)/(n_2 - n_1)$ where n_2 is the cumulative number of defects observed up to time 2, $n_1$is the cumulative number of defects observed up to time 1, $\lambda_2$is the observed failure rate at time 2 and λ_1 is the observed failure rate at time 1. [7]

$\lambda(t) = N_0 * \exp(-Q*t/N_0) / t$

4.3. Expected versus actual results

There was one set of parameters that did not correlate as strongly as the author had expected. These parameters pertained to Configuration Management (CM) and source control.

The author has reason to believe that these parameters did not correlate strongly with this set of data because all of the organizations in this study had some type of configuration management or source control. Their methods and techniques for achieving this varied greatly therefore making it difficult to quantify a correlation between specific CM practices and defect density.

Parameter	**Single parameter correlation to defect density**
Procedures for CM and source control?	-0.14
CM system used?	-0.45
Allows for concurrent versions?	-0.15
Ability to stop shipment?	-0.24
Used for requirements and design documents?	-0.32
Fully automated?	-0.13

Table 2. Configuration management practices correlated to defect density

This author continues to monitor configuration management and source control parameters as these parameters may ultimately correlate as the sample size in this study increases.

Another set of parameters that did not correlate as strongly as this author expected was the use of automated tools. While unit-testing tools had a strong correlation, the use of requirements and coding tools did not. It is possible that the organizations that used these types of tools did not use them properly or to their fullest range of capabilities.

This author has refined the questionnaire process to filter for organizations that "correctly" use automated tools.

Parameter	Single parameter correlation
Requirements tools	-.32
Design tools	-.55
Coding tools	-.40
Unit testing debuggers	-.76
System testing tools	-.11

Table 3. Tools correlated to defect density

Another parameter that did not correlate as strongly as this author had expected was coding reviews. It is interesting to note that requirements reviews and design reviews correlated very strongly. It is likely that coding reviews did not correlate strongly in **this** study because:

1) The reviews may not have been conducted on the portion of the code that has the highest occurrence probability
2) The review criteria may have been geared towards maintainability issues that impact the longevity of the code but may not result in an immediately observable decrease in defect rates
3) The reviews may not have been followed up by action items in an appropriate time frame.

The author has added additional detail to the questionnaire to filter organizations that are not performing code reviews effectively.

Parameter	Single parameter correlation
Coding reviews	-.125
Design reviews	-.65
Reqs reviews	-.87

Table 4. Reviews correlated to defect density

Another parameter that did not correlate as strong as this author had expected is the use of Object Oriented (OO) methods. The single parameter correlation for this was -17. All of the organizations that employed OO methods admitted that their product was a hybrid of OO and procedural code. They also admitted that at least some members of the software organization were not able to name the 4 characteristics that distinguish OO programming. These are inheritance, polymorphism, abstraction and encapsulation.

The author continues to monitor OO practices and has added more detail to the questionnaire in order to filter the degree to which OO practices are employed.

The author has been monitoring the below set of parameters but to date does not have sufficient data to determine a quantitative correlation. However, the author continues to collect this data and may ultimately have these parameters as part of the model.

Parameter	Expected correlation with defect density
Module complexity	Positive
% of copy and paste code	Positive
% Comment statements	Negative
Test completion ratio	Negative
% changed requirements	Positive
% changed design	Positive

Table 5. Product measures that may ultimately correlate to defect density in this study

4.4 The parameters with the strongest correlation

With one exception, there were no parameters that correlated significantly stronger then expected. This author did not expect Year 2000 (Y2K) testing to impact defect densities since the defect densities were observed in the 1999 calendar year. However, Y2K testing had a single parameter correlation of -.54 to defect density. The author believes that the organizations that did Y2K testing were likely finding other defects during this testing process.

The ten parameters with the strongest correlation are shown on the next page. None of these were a surprise to the author.

Parameter	Single parameter correlation to defect density
1. Consistent and documented formal and informal reviews of the software and system requirements prior to design and code.	-.87
2. The language and operating system is well supported by industry.	-.86
3. Existence and use of test beds	-.85
4. Incremental testing (as opposed to "big bang" testing.)	-.84
5. Scheduled regression testing consistently performed by independent testers	-.83
6. Defect and failure tracking systems that are used by testers for test plan development	-.81
7. There is a defined life cycle model that best suits the application, market, and organization.	-.81
8. Testers involved during requirements and design. Test plans are started during the requirements phase.	-.76
9. Automated unit testing tools	-.76
10. Explicit test cases for user documentation	-.76

Table 6. The parameters that exhibited strongest negative correlation to defect density

5. Practical uses for the model

This model can be used in one of two ways. It can either be used to predict defect density or it can be used to make improvements in ones development process. This author has found that software managers are generally more interested in the relative measure as opposed to the absolute measure. While reliability engineers are generally interested in the absolute measure.

The key feature of this model is for management to select the 2 or 3 development practices that have the highest weighting with the lowest cost. The weighting factors that the author applied to each development practice provide a good starting point for this.

Another key feature is to use this model to satisfy quantitative software reliability requirements as early as the proposal stage. This model can be used before a single line of code is written.

6. Ongoing Work

This work done is ongoing. New data is added on a yearly basis. The author plans to expand the sample size to include organizations with higher SEI CMM levels as the highest SEI CMM level in this study was 2.5. The author also plans to include organizations in a variety of industry and application types.

7. Biography

Ann Marie Neufelder is the author of "Ensuring Software Reliability" published by Marcel Dekker, is the co-author of SEMATECH's Tactical Software Reliability Guidebook and "System Software Reliability Assurance Guidebook" for Rome Laboratory.

8. References

[1,7] J. McCall, W. Randell, J. Dunham, L.Lauterbach, "Software Reliability, Measurement, and Testing Software Reliability and Test Integration RL-TR-92-52", Produced for Rome Laboratory, Rome, NY, 1992.

[2]A.M. Neufelder ,"Benchmarking Software Reliability" presented at the "SEMATECH Artwork V workshop", Austin, TX, 1994.

[3], A.M. Neufelder, "SoftRel's Benchmarking Study" , Published by SoftRel, Hebron, KY, 1997. This document is available via softrel@ix.netcom.com.

[4,8] A.M. Neufelder, "The Naked Truth About Software Engineering in the Semiconductor Equipment Industry", Published by SoftRel, Sugar Land, TX. This document is available via softrel@ix.netcom.com.

[5] D.K. Dunnaway, S. Masters, "CMM Based Appraisal for Internal Process Improvement (CBA IPI) Method Description, Technical Report CMU/SEI 96-TR-007", Software Engineering Institute, Pittsburg, PA,1996.

[6] P. Lakey, A.M. Neufelder, "System Software Reliability Assurance Guidebook", Rome Laboratory, Rome, NY, 1995. Table 7-9.

Organization Commitment	**Part of the Rome Laboratory model?**	**Correlation**	**Maximum score allowed**
• Upper management views software as an engineering discipline (they measure software deliverables just as any other deliverable)	No	-.37	1.4
• Software managers view software as an engineering discipline (software is not an afterthought). The software manager does not code but does participate in requirements and design.	No	-.70	28
• There is a defined software team structure that optimizes both capabilities and time	Yes	-.52	2.8
• Software engineers are located geographically near the other engineers	No	-.32	5.6
• There is a defined structure for how software interacts with other engineering disciplines	No	-.13	0 (parameter dropped)
• The software engineers view themselves as engineers as opposed to artists (Do they consider themselves to be part of engineering? Or would they rather "create" without having to deal with engineering?)	No	-.33	7
• Are short term outside contractors used for design/code that does not require intimate understanding of your application and proprietary inventions? Are they used for code that is generic as opposed to code that someone in your organization can do better?	No	-.53	7
• There are software testers that do not write software	No	-.56	24
• There are software quality engineers that do not write software	Yes	-.56	28

Table 7. Organization factor

Life cycle practices	**Part of the Rome Laboratory model?**	**Correlation**	**Maximum score allowed**
Life cycle model			
• Is there a defined life cycle model that best suits your organization, application, market and time to delivery constraints? (Examples are waterfall, incremental, spiral, combinations of these.)	No	-.81	140
Analysis/Requirements			
• Is analysis tasks part of the schedule?	No	-.73	2
• Is the customer or customer representatives involved in this phase?	No	-.49	Dropped
• Do you translate customer requirements to software requirements?	Yes	-.43	Dropped
• Are conflicts resolved before moving forward with design and code?	No	-.37	Dropped
• Are there requirements reviews?	No	-.84	15
• Do you prototype requirements when applicable?	No	-.64	Dropped
• Are testing personnel involved in this phase	No	-.67	6
• Is the test planning started in this phase?	No	-.76	5
• Do you use tools for defining and translating requirements?	Yes	-.32	Dropped
Design			
• Is design tasks part of the schedule?	No	-.64	4
• Are there procedures for it? Are they used?	No	-.62	1
• Are requirements traced to design?	No	-.64	1
• Is there a top-level design?	Indirectly	-.36	Dropped
• Is there a detailed design?	Indirectly	-.41	Dropped
• Are conflicts between design and requirements resolved before coding?	No	-.62	3
• Are there design reviews?	Yes	-.65	2
• Is prototyping used?	No	-.64	1
• Is testing involved during this phase?	No	-.73	10
• Is the test plan evolving during this phase?	No	-.76	10
• Are design tools used?	Yes	-.55	1
Coding			
• Are there coding standards and are they used?	Yes	-.63	.5
• Are software requirements explicitly traced to the design to show that no requirements are missed in the design?	No	-.65	.5
• Are conflicts in the customer or system requirements or top level design or detailed design resolved before designing or coding?	No	-.51	Dropped
• Is module level error handling left as an after thought?	No	-.65	.5
• Are debuggers used to assist in coding?	Yes	-.37	Dropped
• Are automated tools used for coding?	Yes	-.40	Dropped
• Is at least someone from the testing organization is involved in the coding phase mainly to keep track of any changes to the requirements as a result of coding?	No	-.73	4
• Is the test planning refined during the phase in the event that the requirements are changed as a result of any conflicts found during coding?	No	-.84	10
• Is the code reviewed before moving forward to testing	No	-.125	Dropped

Table 8. Life cycle practices factor

Life cycle practices (continued)	**Part of the Rome Laboratory model?**	**Correlation**	**Maximum score allowed**
Unit Test			
• Is unit testing part of the schedule? Or is it an after thought?	No	-.66	7.5
• Are their procedures or checklists for unit testing? Are they used?	No	-.46	.25
• Are requirements and design mapped to the unit test?	No	-.54	.5
• Are conflicts found during unit testing resolved prior to going to system testing or delivery?	No	-.52	.5
• Are debuggers used?	Yes	-.76	7.5
• Are there functional unit tests to verify the mapped requirements?	No	-.65	1
• Is white box path testing done?	Yes	-.54	.5
• Is white box range testing done?	No	-.52	.5
• Are algorithms tested from a white box perspective?	No	-.64	1
• Is white box complex logic testing done?	No	-.52	.5
• Is white box error handling testing done?	No	-.41	1
• Are testers involved in this phase to perform any applicable black box testing?	No	-.65	1.5
• Is the test plan refined during this phase?	No	-.65	1.5
• Are tools used?	Yes	-.53	1.5
System Testing			
• Is system testing part of the schedule? Does someone who did not do the coding perform it?	No	-.60	10
• Are there procedures for testing?	No	-.26	2
• Does the test plan map to all written and implicit requirements?	No	-.89	40
• Are conflicts between the test plan and the requirements resolved?	No	-.65	6
• Does the test plan contain tests for the critical path and use cases?	Yes	-.27	2
• Does the test plan contain tests for error handling?	Yes	-.60	6
• Does the test plan contain tests for system behavior?	No	-.09	Dropped
• Does the test plan contain installation tests?	No	-.49	4
• Are test beds used?	No	-.85	40
• If applicable, does the test plan address performance?	No	-.72	12
• If applicable, does the test plan address security requirements?	No	-.61	6
• If applicable, does the test plan address multi-user requirements?	No	-.50	4
• If applicable, does the test plan address configuration requirements?	No	-.65	6
• Is the user documentation tested?	No	-.76	12
• Was Y2K planned for?	No	-.54	4
• Are automated tools used for testing?	No	-.11	Dropped
• Is a simulator used for testing in the event that the real hardware is not available?	Yes	-.68	10

Table 8. Life cycle practices (continued)

Life cycle practices	Part of the Rome Laboratory Model?	Correlation	Maximum score allowed
Regression Testing			
• Personnel, other then the developers who wrote the code or fixed the code, re-test the software after some amount of corrective action activity and that the test is from an end users perspective.	No	-.83	1.25
• How the regression testing is expected to proceed is documented and procedures are used.	No	-.71	.75
• Someone retests all corrective actions made after some baselined version other then the programmer on a built version of software.	No	-.62	.5
• Someone retests all new features made after some baselined version other then the programmer on a built version of software.	No	-.68	.5
• Someone retests any other changes made after some baselined version other then the programmer on a built version of software.	No	-.75	.75
• All use cases that are critical to the end user are retested by someone other then the programmer regardless of whether the changes since the last regression test impact this use case.	No	-.36	.125
• Areas of the software, which are historically high risk, are retested regardless of whether these high-risk areas were changed since the last regression test.	No	-.53	.25
Miscellaneous			
• Do you have general quality assurance procedures that apply to your software and it's process?	No	-.45	.2
• Do you have procedures for how and when software versions are delivered to your customer? A checklist is an example.	No	-.49	.2
• Do you have tools for project management for software projects?	No	-.36	.1
• Is the software design object oriented?	No	-.17	Dropped
• Understands OO – Can all programmers describe the 4 characteristics of object oriented software?	No	-.10	Dropped
• Do you have informal peer walkthrus?	Yes	-.67	1.5
• Do you measure things like complexity, size, etc for the purpose of making sure that your code is not spaghetti code?	Yes	-.50	.2
• Is there a method in place for scheduling other then to take your best guess and hope that it is right?	No	-.39	.1
• Process metrics – Do you measure these actuals at project completion? 1) how many effort months 2) how many calendar months and 3) how big.	No	-.57	.5
• Fault metrics – Do you measure MTTF, defect density or any other similar metric for the purpose of determining when the software is acceptable for delivery?	No	-.68	1.5
• Reuse libs – Do you have reusable code or design in a library that all programmers can access?	No	-.47	.1
• Root cause – Do you occasionally do a root cause analysis to see what the most common software defect categories are?	No	-.29	.05
• If so, do you make changes to your process as a result?	No	-.29	.05
• Do you have equipment, or a fragment of the equipment, that can be used by the software engineers on a regular and predictable or scheduled basis	Yes	-.26	.05

Table 8. Life cycle practices (continued)

Product metrics		**Correlation**	**Maximum points allowed**
Is the language(s) employed supported well? (Are there automated tools that support this language? Does the manufacturer of this language/compiler provide tech support? Does the manufacturer of this language/compiler provide releases on a regular basis?)	No	-.86	70
Is the operating systems(s) employed supported well? (Are there automated tools that can be used on this OS? Does the manufacturer of this OS provide tech support? Does the manufacturer of this OS provide releases on a regular basis?)	No	-.86	70

Table 9. Product metrics factor

Change Control		**Correlation**	**Maximum score allowed**
Corrective Action			
• How corrective action should be done is documented and accepted.	No	-.11	Dropped
• The programmer while making a corrective action tests all changed units.	No	-.73	5
• The programmer while making a corrective action retests all paths that intersect a change.	No	-.53	2
• Source control is employed to track corrective actions.	No	-.57	2
• The programmer tests all new units when making any corrective actions.	No	-.73	5
• The programmer documents all changes in a failure reporting system.	No	-.68	5
• The programmer retests all areas of code that use global data impacted by this corrective action.	No	-.43	1
Failure and defect reporting systems			
• Do you have a system for tracking failures reported both internally and by your customers?	No	-.46	3
• Are there written procedures for how internally and externally reported failures are tracked?	No	+.002	Dropped
• Are the procedures used?	No	-.45	3
• Is the tracking system automated?	No	-.69	9
• Do testing personnel use this system to do regression testing?	No	-.81	45
• Can your customer input information into this system either directly or via helpdesk?	No	-.49	3
• Can any applicable field persons input information into this system either directly or via helpdesk?	No	-.22	Dropped
• Do systems engineers have access to this system? Do they use it?	No	-.40	3
• Do other engineers like firmware, etc. have access? Do they use it?	No	-.75	36
• CM link – Is there a physical link between your FRACAS and your source control system so that changes made to the source code are all reflected somehow in your FRACAS system?	No	-.43	3
• Is your system on the internet or intranet?	No	-.44	3

Table 10. Change control factors

Question	Reason
Defect metrics	
What percentage of your defects found in testing are the result of corrective action to code that used to work? (Note that a review of the failure and defect databases is required to measure this accurately.)	If this is high that indicates a process that is adhoc.
What percentage of your defects found by the customer is the result of corrective action to code that used to work? (Note that a review of the failure and defect databases is required to measure this accurately.)	Same as above. Additionally, this indicates a more urgent need to get the process under control, as the lack of process is most likely visible to the customer.
What percentage of defects is found in each of these phases? 1) Requirements reviews 2) design reviews 3) coding/unit testing 4) integration 5) system testing 6) customer. (Note that a review of the failure and defect databases is required to measure this accurately.)	This profile should resemble a bell curve if each of the review/testing activities is effective.
What is the top 3 root causes for software defects? (Note that a review of the failure and defect databases is required to measure this accurately.)	The answer itself is not really the important piece of information. What is important is whether the organization is able to accurately identify the top root causes AND address them once known.
Productivity	
Amount of code or function points delivered, calendar time from start of design through delivery required and man-years expended from start of design through delivery.	The answer is not as interesting as whether or not the organization is actually tracking this. How big, how long and how much are three metrics that are basic measures yet are required prerequisites for scheduling accurately.
Releases per year	Used for informational purposes in order to compare defect density and scores to a customer visible measure.
Life cycle process measures	
Configuration Management and Source Control	As discussed earlier, there are 7 questions related to this topic which currently do not have a strong correlation to defect density but are tracked anyhow in the event that the parameters correlate at a later time when the sample size is greater.
SEI CMM level	This is tracked for information purposes.
Industry and application type	This is tracked for information purposes
Product metrics	As discussed in this paper, 6 product metrics are tracked when the information is available in hopes that there will ultimately be enough data to develop a correlation.

Table 11. Questions that don't contribute to the score