7

In Praise of Defects: Defects and Defect Metrics

If debugging is the process of removing bugs, then programming must be the process of putting them in.
—Unknown

No software system of any realistic size is ever completely debugged—that is, error free.
—Edward Yourdon and Larry Constantine **[1]**

Defects do follow a Rayleigh pattern, the same as effort, cost, and code constructed. This curve can be projected before the main build begins; it can be finely tuned. It is the key ingredient for the statistical control of software reliability.
—Lawrence H. Putnam and Ware Myers **[2]**

Huns learn less from success than they do from failure.
—Wess Roberts **[3]**

7.1 WHY STUDY AND MEASURE DEFECTS?

Let us start thinking about defects by trying to answer two defect-related questions:

1. Your system has no reported field defects. It was released six months ago. Is it time to break out the champagne?
2. You are building a commercial application. Your CEO, who is very dedicated to quality, directs you to find and fix 99% of the bugs before you ship. What is your response?

Software Measurement and Estimation, by Linda M. Laird and M. Carol Brennan

Think about these two scenarios as we study defects in this chapter. We will then return to them in Section 7.9 and provide reasonable answers based on what we have learned.

This chapter is about defects—their patterns, their rhythms, their predictability, and the stories they can tell, to any and all who will listen.

Most people view software defects as undesirable and just want to get rid of all of them as soon as possible. The quantitative software engineer sees them a little differently (although still wanting to be rid of them quickly). Defects are real, observable manifestations and indications of the software development progress, process, and quality. From schedule, quality, and process engineering viewpoints, they are invaluable. From a schedule viewpoint, they give a clear indication of progress (or lack thereof). From a quality viewpoint, they give an early indication of the expected defect density of the product. From a process engineering viewpoint, they give clear indications of the effectiveness of the software development processes and indicate targets for improvement. You can see them, count them, predict them, and trend them. Defects are actually one of the best and most useful pieces of data available in software development.[1]

Defects have their own behavioral patterns and dynamics that need to be understood to manage and control software development projects and processes. To those who pay attention to them, defects give a wealth of understanding and insight into both the software product and the software development processes.

7.2 FAULTS VERSUS FAILURES

Software engineers use the terms defects, faults, and bugs interchangeably and occasionally interject the term failures when speaking about undesirable system behavior. Faults are defects that are in the system at some point in time. Failures are faults that occur in operation. Defects metrics measure faults. Reliability metrics measure failures. Bugs are synonymous with defects and faults.

If code contains faults but the faults are never executed in operation, then the system never fails. The mean-time-between-failures (MTBF) for the system will approach infinity and software availability will be 100%.

If there is only one fault in an entire system, and it is executed in the boot sequence of the system, then the system will fail every time, and the MTBF approaches 0 and the software availability will be 0%.

Faults are defects in the system that may or may never be seen in operation.

Faults can range in severity from crucial—they must be fixed immediately— to inconsequential—they may never be worth fixing. As part of any project, you need a defect severity scheme that is relevant for that project. For defect metrics and

[1]Think of defects as the pain of software development. Pain is a huge feedback mechanism. If we did not have it, we would damage ourselves on a daily basis. Pain and defects, by themselves, are undesirable—something you want to eliminate. When understood by trained professionals, they can lead to discovery and treatment of underlying problems, which eventually lead to better health or systems in the long term.

management, focus on the defects that will actually impact your project and product performance.

The standard measure of defects today is defect density: the number of defects per KLOC or per function point.

7.3 DEFECT DYNAMICS AND BEHAVIORS

7.3.1 Defect Arrival Rates

Defects are not detected at random intervals. Defects have certain dynamics, behaviors, and patterns. Starting with the beginning of the software life cycle, they tend to follow a Rayleigh curve, as shown in the Figure 7.1 [4]. You can predict these curves, along with the upper and lower control limits, based on the project size, process maturity, and past history.

7.3.2 Defects Versus Effort

There tends to be a linear relationship between defects and effort [4], assuming that the other factors (such as process, team skill level, and technology) stay constant (Figure 7.2). In other words, people tend to make errors, which lead to defects, at a relatively constant rate. All you can do is adjust the slope of the line, through improvements in processes, training, technology, and skill levels or through changes in schedule and staffing levels.

7.3.3 Defects Versus Staffing

Defects detected over time tend [4] to be similar to staff effort, with an additional time lag for error detection (Figure 7.3).

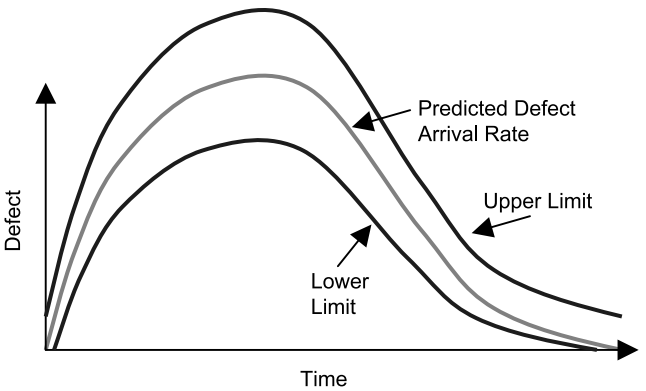

Figure 7.1. *Defect arrival rates.*

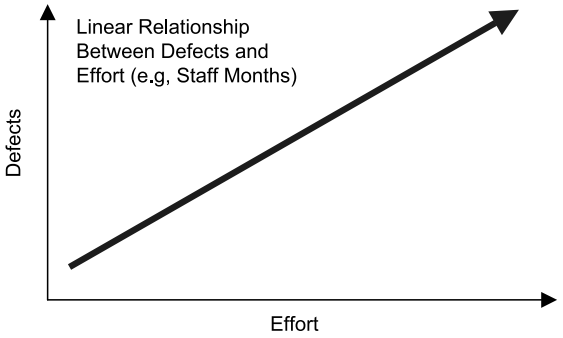

Figure 7.2. *Defects versus effort.*

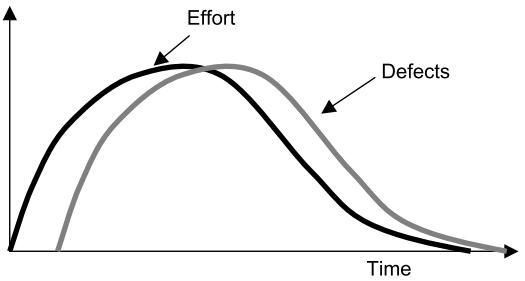

Figure 7.3. *Defects versus staffing.*

7.3.4 Defect Arrival Rates Versus Code Production Rate

Detected defects are related to the code production rate [4], which in turn is related to the staff effort curves. All of them tend to follow Rayleigh curves. Note that if you start tracking the defect data once formal testing begins, the defect discovery curve will look exponential (Figure 7.4).

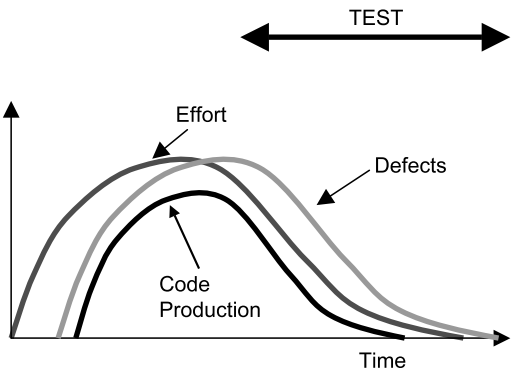

Figure 7.4. *Defects versus code production rate.*

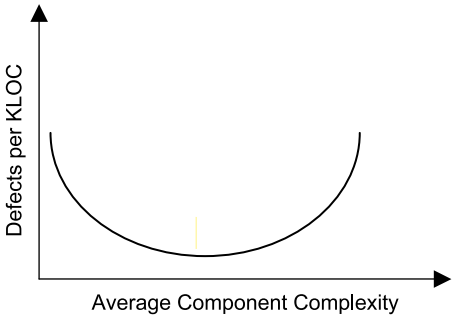

Figure 7.5. The bathtub chart.

7.3.5 Defect Density Versus Module Complexity

Figure 7.5 shows the relationship between the complexity of a software module and defect density, which we explored in detail in Chapter 5 [5]. It is occasionally called the "bathtub chart." It is important to remember that defect density for small software modules is high, falls exponentially as the size increases, bottoms out for a while, and then rises again. Intuitively, this seems reasonable. For any module, there is an initial set of "startup defects," which, if you have a very small module, will create a high defect density. On the other end, as the software becomes increasingly complex, the complexity engenders a higher rate of defects. The implications are clear—you want to design and build your systems such that your modules are at the bottom of the "bathtub," where the defect density is the lowest.

7.3.6 Defect Density Versus System Size

The "cloud" shown in Figure 7.6 is an approximation of Putnam–Myers QSM data for defects versus effective source lines of code from 1997 [4]. Although the number of defects varies widely based on the size of project, the trend is linear.

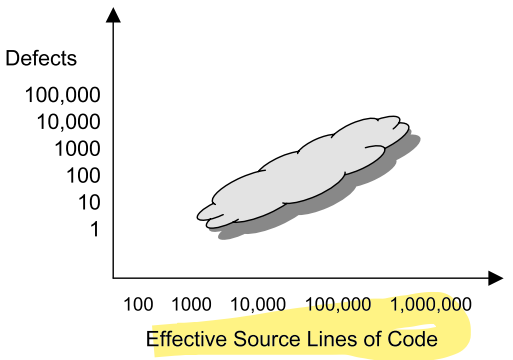

Figure 7.6. Defects found from Integration to Delivery—QSM Data Base.

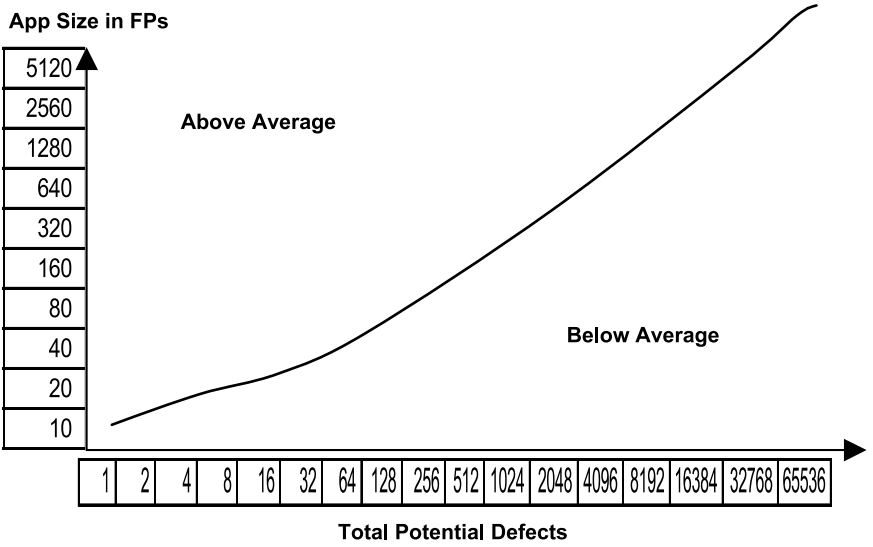

Figure 7.7. Defects versus function points.

The chart in Figure 7.7 is from Jones in 1991 [6]. The details are different, but the trend and pattern are the same as the Putnam–Myers data.

7.4 DEFECT PROJECTION TECHNIQUES AND MODELS

The objective of defect prediction and estimation techniques and models is to project the total number of defects and their distribution over time. The techniques vary considerably. There are both dynamic and static defect models. The dynamic models are based on the statistical distribution of faults found. The static models use attributes of the project and the process to predict the number of faults.

Static models can be used extremely early in the development process, even before the tracking of defects begins. Dynamic models require defect data and are used once the tracking of defects starts.

7.4.1 Dynamic Defect Models

Dynamic models are based on predicting—via calculations or tools—the distributions of faults, given some fault data. The primary concept is that the defects do follow distribution patterns as discussed above, and given some data points, you can generate the arrival distribution equations.

There are many different defect distribution equations, but the primary ones are Rayleigh, exponential, and S-curves. Rayleigh distributions model the entire development life cycle. Exponential and S-curves are applicable for the testing/deployment processes.

There are two distribution functions: the probability distribution function (PDF) for defect arrivals, which is called $f(t)$, and the cumulative distribution function (CDF) for total number of defects to arrive by time t, which is called $F(t)$. Note that $f(t)$ is the derivative of $F(t)$.

The Rayleigh and exponential curves are both in the family of the Weibull curves and have the forms

$$f(t) = m(t/c)^m * e^{(-t/c)^m}/t$$

$$F(t) = 1 - e^{(-t/c)^m}$$

When $m = 1$, these are equations for an exponential distribution. When $m = 2$, these are equations for a Rayleigh distribution.

7.4.1.1 Rayleigh Models

The equations for the basic Rayleigh curves, where t is time, c is a constant, and K is the total number of defects (e.g., area under the curve), are

$$f(t) = K * 2(t/c^2)e^{-(t/c)^2} \quad \text{and} \quad F(t) = K(1 - e^{-(t/c)^2})$$

Interestingly, $c = \sqrt{2}\, t_m$, where t_m is the time t at which $f(t)$ is maximum. Therefore,

$$F(t) = K\left[1 - e^{-(1/2t_m^2)t^2}\right]$$

$$f(t) = K\left[(1/t_m)^2 te^{-(1/2t_m^2)t^2}\right]$$

The ratio of defects that should appear by time t_m is defined as $F(t_m)/K$, which is $[1 - \exp(-0.5)]$ or $\sim .4$. Therefore, ~40% of the defects should appear by time t_m.

There are multiple ways to use these equations to predict arrival rates and total number of defects. There are many commercially available tools that will do these calculations in a statistically valid manner. A few simple methods which you can calculate yourself and that can get you started are demonstrated next.

Method 1: Predicting the Distributions—An Extremely Easy Method Given that you have defect arrival data, and the curve has achieved its maximum at time t_m (e.g., the inflection point), you can calculate $f(t)$, assuming the Rayleigh distribution. The simplest method is to use the pattern that ~40% of the total defects have appeared by t_m. This is a crude calculation but it is a place to start.

For example, assume you have the following data for arrival rates for defects:

Week	1	2	3	4	5	6	7	8	9
Defects found	20	41	48	52	62	59	52	44	33

t_m is week 5. Since ~40% of the defects appear by t_m, and the sum from week 1 through week 5 of defects is 223, then the simple calculation for the total number of defects is $223 * (100/40) = \sim 557$.

You can determine $f(t)$ once you have K and t_m.

Continuing the example and substituting in $K = 557$ and $t_m = 5$,

$$f(t) = 557(t/25)\, e^{-t^2/50} = 22.3te^{-t^2/50}$$

$$F(t) = 557(1 - e^{-t^2/50})$$

Method 2: Predicting the Distributions—A Little More Complex Method You can solve for $f(t)$ by using t_m and one or more data points to solve for K and $f(t)$. The simplest way is to take just one data point; the easiest point to use is $f(1) = 20$. Substituting in $t_m = 5$, we have

$$20 = K(1/25)\, e^{-1/50}$$

$$K = 20 * 25 * e^{1/50}$$

$$K = 510$$

So

$$f(t) = 510(t/25)\, e^{-t^2/50} = 20.4\, te^{-t^2/50}$$

$$F(t) = 510(1 - e^{-t^2/50})$$

We know that at least three points are needed to estimate these parameters using nonlinear regression analysis, and that further statistical tests (such as standard error of estimate and proportion of variance) are necessary for these parameters to be considered statistically valid. In the case where high precision is important, use a statistical package or one of the reliability tools to calculate the curve of best fit.

As an interim and easier step, we can calculate K for all values of $f(t)$, assuming $t_m = 5$ and then calculate the mean and standard deviation. That is, $K = [25 f(t) e^{-t^2/50}]/t$. These calculations are easily performed with a spreadsheet. The mean for K is 491 with a standard deviation of 22.3.

Week	1	2	3	4	5	6	7	8	9
Defects found	20	33	48	60	62	59	52	44	33
K	510.1	446.9	478.9	516.4	511.1	505.0	494.8	494.5	463.2
Mean	491.2								
SD	22.37972								

Note that these calculations are sensitive to t_m and although they are not completely valid in a statistical sense, they give the statistically challenged practitioner a

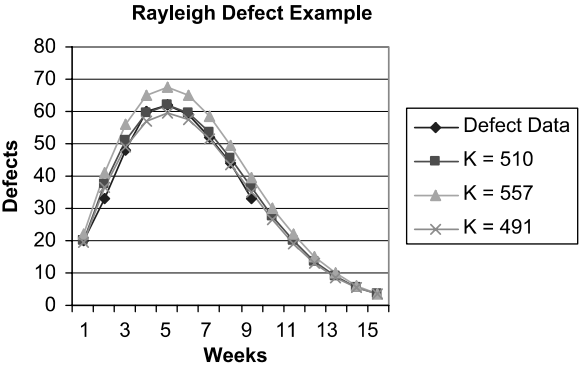

Figure 7.8. *Rayleigh defect example: defect projection for different* K *values.*

relatively easy method for calculating the defect detection distribution functions, with some initial control limits. Considering the precision of the typical data, they are reasonable. Figure 7.8 is a graph that shows $f(t)$ for $K = 510, 491, 557$. All three curves are close to the actual data and very similar to each other. From an "eyeball" perspective, $K = 491$ looks the best, as we would expect.

Method 3: Predicting the Arrival Rates Given Total Number of Defects and Schedule Once you predict the total number of defects expected over the life of a project, you can use the Rayleigh curves to predict the number of defects expected during any time period by using the equations below. You can also compare projected defects with actual defects found to determine project performance.

T_d is the total duration of the project, at which time 95% of all defects have been found.[2] K is the total number of faults expected for the lifetime of the delivery.

Then the number of defects for each time period t is

$$f(t) = (6K/(T_d^2))te^{-3(t/T_d)^2}[2]$$

For example, you know that past, similar projects have had a defect density of 10.53 defects per KLOC and you have predicted that this project will be 100 KLOC. You also have a reasonable expectation, based on process improvements, that you expect to have 5% fewer defects. Therefore, you project that the total number of defects for this project will be $10.53 * 100 * 0.95 = 1000$.

Given a total duration of the project of 26 weeks, and a total expected number of faults of 1000, then the expected distribution of faults would be as depicted in Figure 7.9.

Based on the distributions in your data from prior projects, you can add in control limits. You may want to use 2 SD, which will give you a 95% confidence range of performance.

[2]The 95% number here is used as a reasonable target for delivery. The equations are derived from $F(T_d)/K = 0.95$ Another percentage would result in slightly different constants in the equations.

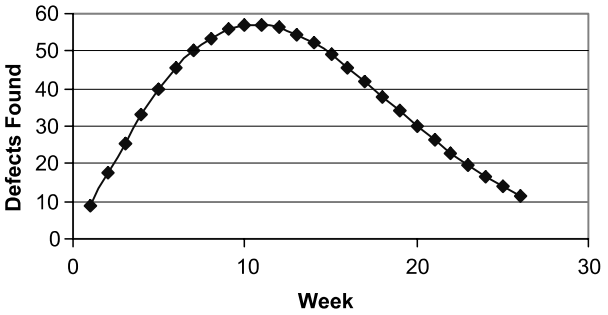

Figure 7.9. *Defects projection for 26 week project.*

If you do not have data from prior projects, you can still project overall defect densities in a number of ways, as discussed later in this chapter, and use the same technique to project arrival rates.

The Rayleigh model is used in a number of tools, which you can find on the Internet. Some of the more notable ones are:

- SPSS (SPSS Corporation)
- SAS (SAS Corporation)
- SLIM (Quantitative Software Management Corporation)
- STEER (IBM Corporation).

7.4.1.2 Exponential and S-Curves Arrival Distribution Models Once testing begins, exponential and S-curve functions are the primary modeling functions for defect arrivals. Figure 7.10 contains both $F(T)$—the cumulative distribution function—and $f(t)$— the arrival function—for both curves.

S-curves resemble an S, with a slow start, then a much quicker discovery rate, then a slow tail-off at the end. They are based on the concept that initial defects may be more difficult to find because of either the length of time for error isolation or crucial defects that need to be fixed before others can be found. There are multiple S-curve models; all are based on the nonhomogeneous Poisson process for the

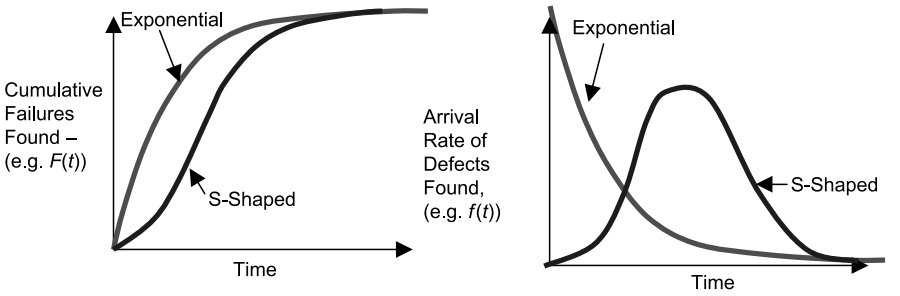

Figure 7.10. *Exponential and S-curves.*

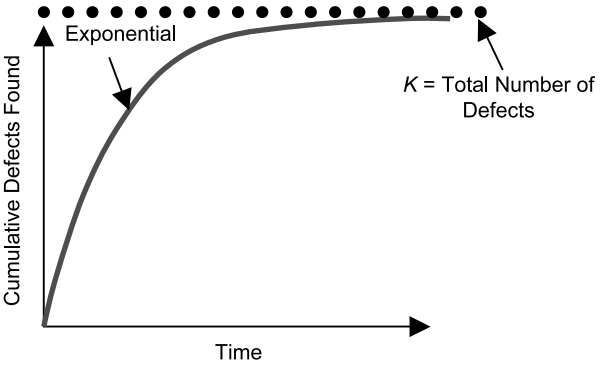

Figure 7.11. *Exponential distribution and* K, *the total number of defects.*

arrival distribution. One equation [7] for S-curves is

$$F(t) = K\left[1 - (1 + \lambda t)^{-\lambda t}\right]$$

Now let us look at the exponential distribution (Figure 7.11):

$$F(t) = K\left(1 - e^{-\lambda t}\right)$$
$$f(t) = K\left(\lambda t e^{-\lambda t}\right)$$

Given a set of data points, you want to know K. The techniques are similar to those for Rayleigh curves. You can (1) use reliability/statistical analysis tools, (2) solve for it using a few points, or (3) eyeball in your own K (. . . but don't tell anyone we said that).

7.4.1.3 Empirical Data and Recommendations for Dynamic Models You may be extremely skeptical of these models and their ability to predict the number of total defects. So let us look at some of the empirical results of using these models:

- Putnam and Myers [8] found that total defects projected using Rayleigh curves were within 5–10%.
- Using their STEER software tool, IBM Federal Systems in Gaithersburg, Maryland, estimated latent defects for eight projects and compared the estimate with actual data collected for the first year in the field. The results were "very close." [9]
- Thangarajan and Biswas [10] from Tata Elxsi Ltd reported on using a Rayleigh model successfully on over 20 projects.

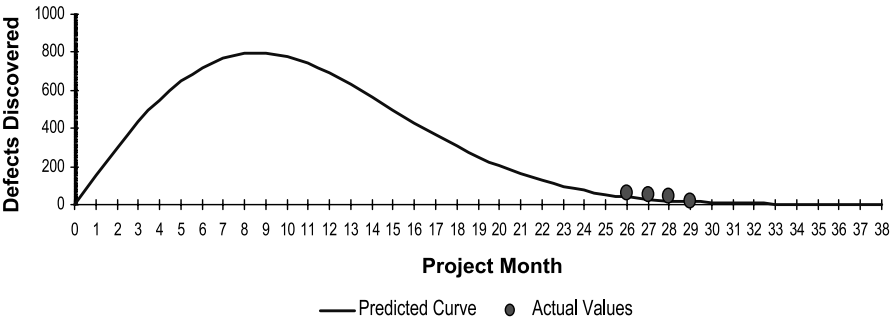

Figure 7.12. *Northrop Grumman defect discovery Rayleigh curve: discovered defects.*

- Figure 7.12 is a chart from Northrop Grumman [11] in 2001, which shows their successful use of the Rayleigh model to predict defects in the test cycle based on defects discovered earlier in the life cycle.
- Some data suggests that $m = 1.8$ for Weibull curves (see Section 7.4.1) may be best, although the calculations become more difficult.

Recommendations Over the past 15 years, and with the move to higher SEI levels, much more defect data has been recorded than ever before. Even so, organizations will vary from the standard patterns. Defect arrival rates tend to follow the Rayleigh curves, and the equations can be finely tuned to match an individual environment.

Our first and primary recommendation is to start tracking and using your defect data if you are not doing it. Our second recommendation is to try the simplest models first, especially the Rayleigh curves, and evolve if needed.

7.4.2 Static Defect Models

Static defect models are based on the software development process and past history rather than the defect data for the current project. They can be used throughout a project's life cycle, including before it begins. They are extremely useful for evaluating and predicting the impact of process changes to the overall quality of a project.

7.4.2.1 Defect Insertion and Removal Model The basic model, which has been credited to Barry Boehm and Capers Jones, is that defects are inserted into a product and then removed. You can improve a product by inserting fewer defects and/or by removing more of them. It is a good, high-level model to use to guide our thinking and reasoning about defect processes (Figure 7.13).

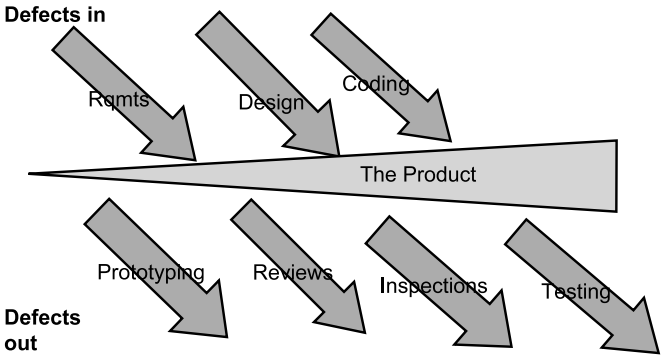

Figure 7.13. *Defect insertion/removal model.*

7.4.2.2 Defect Removal Efficiency: A Key Metric Defect removal efficiency, DRE, is a key metric that is used for measuring and benchmarking the effectiveness of the process, as well as measuring the quality of a project.

For an overall project, $\text{DRE} = E/(E + D)$, where E is the number of defects found before delivery to the end user, and D is the number of defects found after delivery. The goal is to have DRE approach 1, which would mean that no defects were found postdelivery.

For example, you found 10 defects after you delivered your product. There were 990 defects found and removed before delivery. Your $\text{DRE} = 990/1000 = 99\%$.

For each software engineering activity i, $\text{DRE}_i = E_i/(E_i + E_{i+1})$, where E_i is the number of defects found in activity i, and E_{i+1} is the number of errors found after activity i that are traceable to defects that were present during activity i. The goal is to have DRE_i approach 1 as well, so that errors are found and removed before they reach the next activity.

Projects that use the same team and the same development processes can reasonably expect that the DRE values from one project to the next are similar. For example, if on the previous project, you removed 80% of the possible requirements defects using inspections, then you can expect to remove ~80% on the next project. Or if you know that your historical data shows that you typically remove 90% before shipment, and, for this project, you have used the same process, met the same kind of release criteria, and found 400 defects so far, then there probably are ~50 defects still in the product.

The defect removal matrix is a useful matrix that identifies both the phase in which a defect was inserted and when it was removed. It is useful as an analysis tool for the defect removal process; it allows you to calculate the efficiencies of the defect removal for each step in a process.

For example, you are working on a project on which you are tracking your defects and the phases in which the defects were found and removed. You have four defect removal steps: (1) requirements review, (2) design review, (3) testing, and

(4) customer detected. You have three phases of defect insertion that you track: (1) requirements, (2) design, and (3) coding. You can represent your defect data in a DRE matrix as shown below:

	Defect Injection Phase			
Defect Removal Step	Requirements	Design	Coding	Total
Requirements Review	13			13
Design Review	2	12		14
Testing	3	5	32	40
Customer Detected	1	3	4	8
Total	19	20	36	75

This matrix indicates that 19 requirements defects have been found, 13 of which were found in the requirements review, 3 in testing, and 1 by the customer.

$$\text{DRE for System} = \text{Total Detected Before Release}/\text{Total Detected} = 67/75$$
$$= 89\%$$

which, especially for a significant system, is not bad. The DRE of a phase measures the percentage of possible faults that were removed in a phase. For example, in design, there were 26 possible faults that could have been detected (6 requirements and 20 design faults); 14 of them were discovered in the design review phase, resulting in a DRE of 14/26.

The DRE for each phase[3] is shown below:

Phase	DRE by Phase
Requirements	0.68
Design	0.54
Testing	0.83
Customer Detected	1.00

In terms of analysis, this chart tells us our design defect removal process is the least effective, and the testing is the most effective at removing defects. If this were one of our projects, we would launch two quality improvement initiatives:

1. Based on the total number of requirements defects, we would look for process improvements to avoid defects being inserted in the first place: for example,

[3]This calculation assumes that there are no "latent bugs"—bugs that have not been found by the customer. This assumption is highly unlikely, as discussed in later sections. Alternatives would be to use an estimated total number of defects, or to explain that you expect that there are additional latent defects. In either case, the analysis of the data remains valid.

using JAD sessions or prototyping to reduce the number of requirements defects.

2. Based on the DRE of the phases, we would focus on improving the design review process. Typical engineering rules for design reviews are 60–70% removal versus the 54% we have here.

7.4.2.3 Static Defect Model Tools There are tools that support the static defect model and allow you to tune it for your environment. One tool is COQUALMO [12], an extension to COCOMO II.

COQUALMO is a defect prediction model for the requirements, design, and coding phases based on the defect insertion/defect removal model. It is actually two separate models, one for defect introduction and one for defect removal. For defect introduction, COQUALMO use the COCOMO II project descriptors (size, personnel capability and experience, platform characteristics, project practices, and product characteristics such as complexity and required reliability). For defect removal, it uses ratings of a project's level of use of tools, reviews, and execution testing to determine what percentage of the introduced defects will be removed. The result is a prediction of defect density, either in KLOC or FPs. In short, COQUALMO is a mathematical model that takes as input your view of your defect injection drivers and defect removal drivers and gives you a projection of the defect density of your system at different points in the development life cycle (Figure 7.14). It can be used to estimate the impact of "driver changes" on defect density, so that you can do "what if" and "improvement investment" analysis.

COQUALMO is currently populated with data from the COCOMO clients and "expert opinion."

We strongly recommend you use a tool such as COQUALMO and tune it to your environment. It will allow you to quantitatively understand and then engineer your defect injection, prevention, and removal processes to optimize your quality and effort.

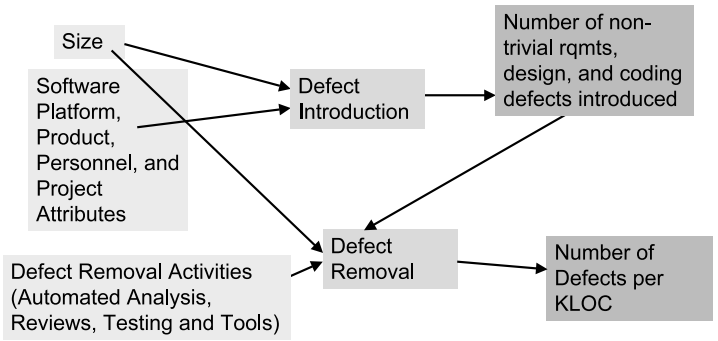

Figure 7.14. *COQUALMO.*

7.5 ADDITIONAL DEFECT BENCHMARK DATA

Defect benchmark data is surprisingly sparse. Many companies consider it proprietary. It is also easy to misuse the data, either by punishing projects that do not meet certain benchmarks or by calculating misleading results caused by differences in counting techniques, such as in size, usage, and severity. Don Reifer [13] has taken a lead in publishing software productivity, cost, and quality data, in hopes of encouraging others to do so. We fully support his position and encourage you to publish your defect data as well.

7.5.1 Defect Data by Application Domain

Reifer's delivered defect data (for 600 U.S. projects from 1997 through 2004) by domain is shown in Figure 7.15.

Reifer's additional comments include:

- Defect rates in military systems are much smaller due to the safety requirements.
- Defect rates after delivery tend to be cyclical with each version released. They initially are high and then stabilize around 1 to 2 defects per KLOC in systems with longer life cycles (>5 years). Web Business systems tend to have shorter life cycles (≤2 years) and may never hit the stabilization point.

Application Domain	Number of Projects	Error Range (Errors/ KESLOC)	Normative Error Rate (Errors/ KESLOC)	Notes
Automation	55	2 to 8	5	Factory automation
Banking	30	3 to 10	6	Loan processing, ATM
Command & Control	45	0.5 to 5	1	Command centers
Data Processing	35	2 to 14	8	DB-intensive systems
Environment/ Tools	75	5 to 12	8	CASE, compilers, etc.
Military—All	125	0.2 to 3	< 1.0	See subcategories
Airborne	40	0.2 to 1.3	0.5	Embedded sensors
Ground	52	0.5 to 4	0.8	Combat center
Missile	15	0.3 to 1.5	0.5	GNC system
Space	18	0.2 to 0.8	0.4	Attitude control system
Scientific	35	0.9 to 5	2	Seismic processing
Telecom	50	3 to 12	6	Digital switches
Test	35	3 to 15	7	Test equipment, devices
Trainers/ Simulations	25	2 to 11	6	Virtual reality simulator
Web Business	65	4 to 18	11	Client/server sites
Other	25	2 to 15	7	All others

Figure 7.15. *Defect density by application domain.*

7.5.2 Cumulative Defect Removal Efficiency (DRE) Benchmark

The percentage of defects removed before software is delivered to a customer tends to be significantly lower than one might expect. Jones' data from 1991 is shown in Figure 7.16 [14]. This table says, for example, that for software larger than 320 function points, typically only 75% of the faults were removed before shipment. Remember that this is data from 1991.

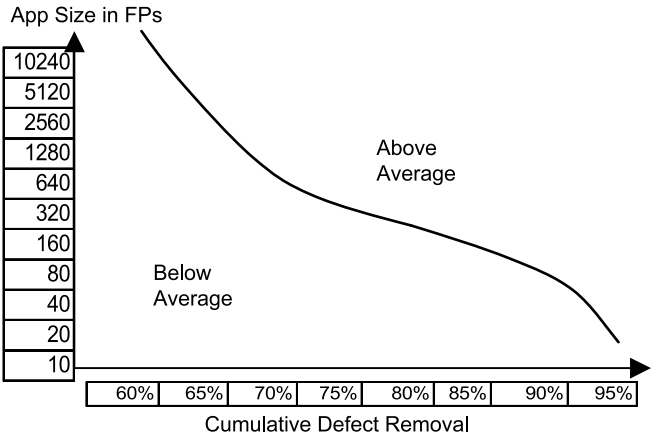

Figure 7.16. *Cumulative defect removal percentage versus function points.*

7.5.3 SEI Levels and Defect Relationships

Diaz and King [15] measured software development at General Dynamics and analyzed the relationship between SEI levels, quality, cost, and rework. They used data from ~20 projects in various stages of the life cycle, with ~360 software development engineers as shown in Table 7.1.

Jones' data from 2000 [16], which looks at DRE per FP, based on SEI levels shown in Figure 7.17. This chart tells us, for example, that the typical U.S. software project ships with only 80% of the defects removed.

TABLE 7.1 General Dynamics Decision Systems Project Performance Versus CMM Level

SEI Level	Customer Reported Unique Defects per KLOC
2	3.2
3	0.9
4	0.22
5	0.19

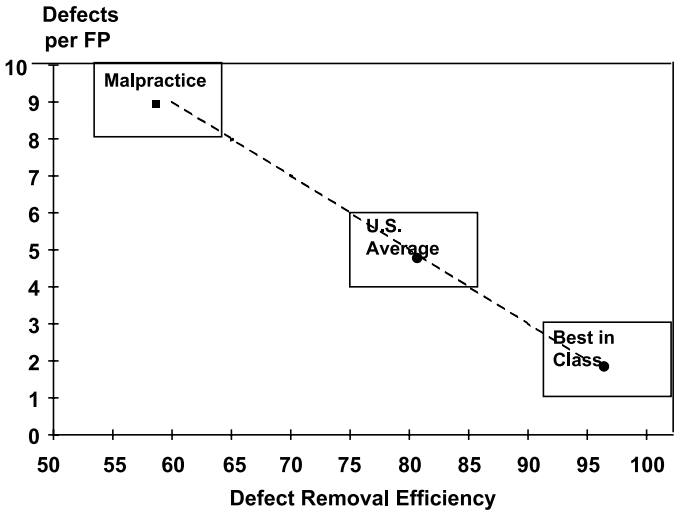

Figure 7.17. *Defect removal efficiency.*

7.5.4 Latent Defects

Two different studies by Hatton and Roberts [17] using only static code analyzers on millions of lines of debugged and released C and FORTRAN code found approximately six defects per KLOC. These were faults such as uninitialized variables. Obviously, there are more latent defects in the code than the static analyzers found.

7.5.5 A Few Recommendations

We are sure that you have noted that these benchmarks are inconsistent in the specifics. What should your goal be—less than 2 defects per KLOC or 0.25 defects per KLOC?

The differences and inconsistencies are at least partially due to differences in data sets and counting methodologies, which make it impossible to have one number. Nevertheless, there is a host of information here and some conclusions can be drawn.

- Zero defects is probably impossible. You will always have latent defects, even if you do not know they are there.
- Good software should be less than 2 defects per KLOC.
- Safe software should be at least less than 1, if not 0.5 defects per KLOC.
- Increasing SEI level conformance tremendously improves the DRE and number of delivered defects.

7.6 COST EFFECTIVENESS OF DEFECT REMOVAL BY PHASE

The later defects are removed, the more expensive they are to remove. Requirements defects that become failures in operation can be tremendously costly, easily costing orders of magnitude more to remedy.[4] Engineering rules for the cost of fixing defects range from a factor of 5 to 10 for each development phase. That is, it costs 5 to 10 times more to fix a coding defect once in formal testing and another 5 to 10 times more to fix it once it is in the field. IBM found in a 1983 study that it cost 20 times more to remove a defect in testing than in design/code inspection and approximately another 4 times more to fix it after the customer began using the system [18].

Your cost effectiveness will be based on your situation. Do you have one field site or 200? Can patches be downloaded easily, or is a site visit required along with database upgrades? Do you have extensive integration and regression testing? We recommend that you gather your own data and understand the costs for your own environment. Short of using your own data, we recommend using the following multiplicative factors as engineering rules:

Cost for fixing in requirements/design—1×
Cost of fixing in coding phase—5×
Cost of fixing in formal testing phase—25×
Cost of fixing in field—250×

7.7 DEFINING AND USING SIMPLE DEFECT METRICS: AN EXAMPLE

This section presents a sample defect construct and a few defect presentation charts. It demonstrates the wealth of information you can get from simple defect data. Notice how each one of the charts has a story to tell to those who pay attention.

The defect-related information that a program manager needs might be:

- Quality of component
- Readiness for release
- Productivity of bug fixing
- Identification of high-risk components

The defect base measures, for each module, might be:

- Defect open date

[4]Just think about all the costs involved in releasing a software fix to an end customer: requirements, development, and testing staff are impacted; development and test environments must be updated and maintained; new software must be packaged and shipped; documentation may need to be updated and reissued. Of course, there is also the additional possibility when making changes that other breakages may occur introducing more costs.

- Defect close date
- Lines of code

A sample defect construct is shown in Figure 7.18.

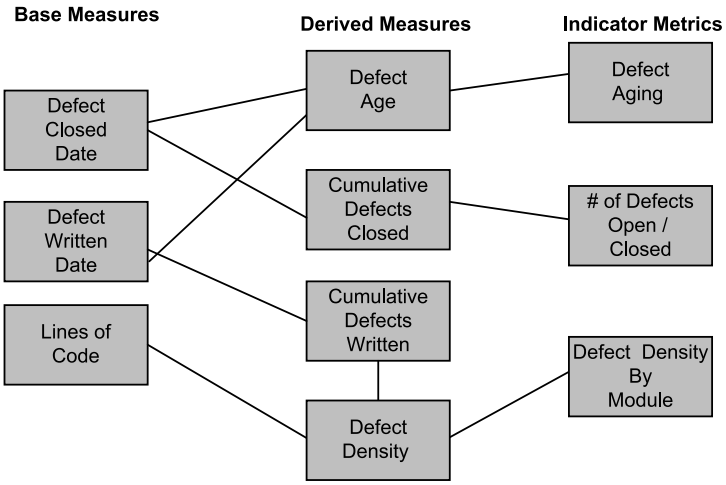

Figure 7.18. *Sample defect construct.*

From this sample construct, we can generate many different indicator metrics, a few of which could be:

- Open/closed/backlog distribution
- Open/closed/backlog cumulative distribution
- Defect density by module
- Defect aging

The objective of the open defect status (Figure 7.19) is to show, per week, the number of new defects, the number of closed defects, and the number of defects that are currently open (in backlog).

What does this chart tell you? How would you improve the chart?

- At a macro level, it looks like the quality of the release is improving, since both the arrival rate of new defects is decreasing as well as the backlog. Also, it appears that the team can fix about 10 defects per week. We need to drill down to understand new versus closed to see if we really are converging. It is not clear yet when we will be ready to release.
- This chart could be improved significantly by clearly having the goals for both ready to release and expected performance with upper and lower bounds. (In Chapter 14 will talk in more detail about how to ensure that any charts you create will be highly effective communication tools.)

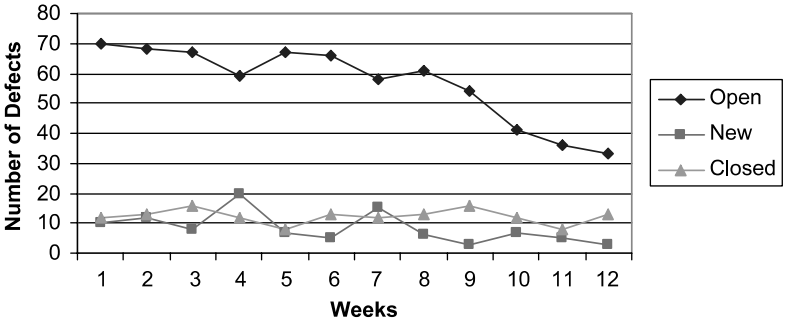

Figure 7.19. *Defect status.*

The module defect density chart (Figure 7.20) is designed to show the defect density of modules compared to a target rate. What does this chart tell you? How would you improve the chart?

- It looks like module 4's defect density is outside the control limits. We need to drill down/investigate to determine the reasons. If, for example, it happens to be a highly complex module, more inspections and testing may be warranted. For modules 3 and 5, we might check just to make sure that adequate testing has been done, and that we do not have a good result just because no one is really finding defects.
- We think this chart looks pretty good as it is. It might be improved if we had trend data over time to add in or some additional detail on what "defects" here really mean—Defects found after release? Defects total? Defects in formal testing?

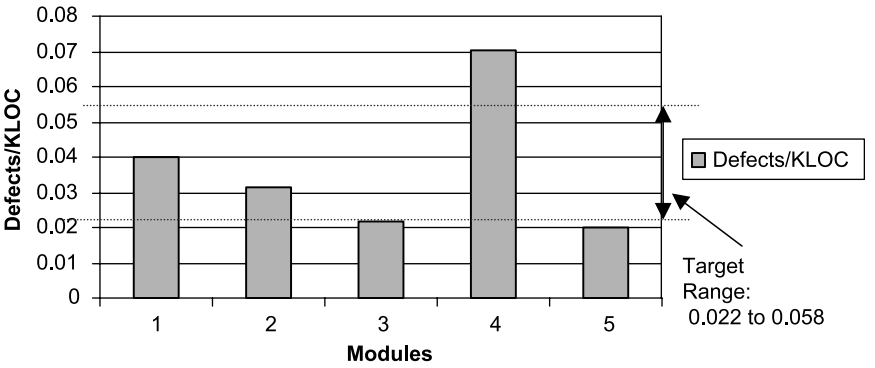

Figure 7.20. *Module defect density (defects/KLOC).*

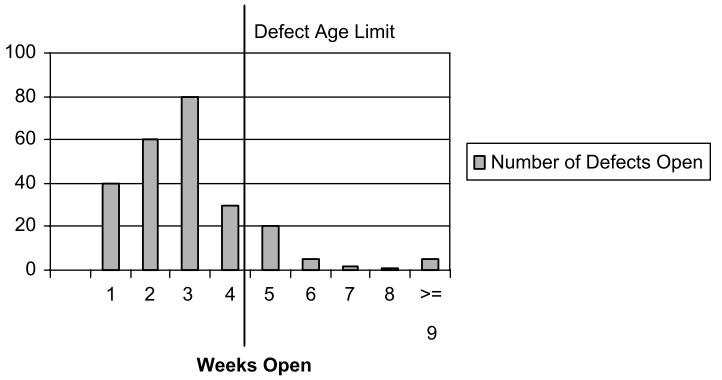

Figure 7.21. *Defect aging chart.*

The defect aging chart (Figure 7.21) is another simple chart that tells a story. What does this chart tell you? How would you improve the chart?

- First, we would be very concerned about the number of defects open past the target date, especially those at or after week 9. What is the cause? Do they really need to be fixed? Or does no one know how to fix them? It looks like, on average, it takes about three weeks to close a defect. This is just a good number to know.
- We have been assuming all the defects are critical. This chart could be improved (along with all of the others) by clearly indicating the severity of the defects.

These three charts are simple examples of the type of information you can easily get from sample defect data. They are invaluable to the project manager and to anyone wanting to understand the true progress of the project. They frequently do not tell the whole story; instead, they give clues and hints that tell you to investigate further to understand what really is going on.

7.8 SOME PARADOXICAL PATTERNS FOR CUSTOMER REPORTED DEFECTS

Jones pointed out in 1991 [6] that there are two general rules for customer reported defects:

1. The number of customer reported defects found correlates directly with the number of users.
2. The number of customer reported defects found correlates inversely with the number of defects found prior to shipment.

These general rules seem, at first, to be at odds with each other, but actually they are interrelated. They are both based on the concept that the more the software is used, the more defects that will be discovered. If the software has many users, it will have more execution time, and hence, more defects will be uncovered. Conversely, if the software is buggy, people will use it less, and fewer defects will be uncovered.

7.9 ANSWERS TO THE INITIAL QUESTIONS

1. Your system has no reported field defects. It was released six months ago. Is it time to break out the champagne?

HP had a system with no reported field defects. Initially, it was thought to be an example of "zero defects." They later discovered that was because no one was using it [17]. Be very careful to understand the usage of your system if the number of reported field defects is significantly lower than expected. It may be due to low usage.

2. You are building a commercial application. Your CEO, who is very dedicated to quality, directs you to find and fix 99% of the bugs before you ship. What is your response?

As studies repeatedly show, finding and fixing 99% of bugs is near impossible and very costly. Our suggestion is to discuss the defect benchmark data with your CEO and to suggest that a reliability or availability target may be more appropriate.

7.10 SUMMARY

Defects have their own behavioral patterns and dynamics, which need to be understood in order to properly predict, plan, and then monitor individual projects.

Defects are inserted into a project throughout its life cycle and are removed through activities such as inspection and testing. The quality of the processes, the skill of the people, and the size of the project highly impact the number of defects inserted. The number and type of defect removal activities highly impact the number of defects removed.

In this chapter we presented multiple methods of predicting and understanding both the number of defects in a project and their arrival rate during testing, development, and deployment. Some of the methods are static, based on characteristics of the project and the processes, such as SEI levels. Other methods are dynamic, based on the early test data for a project.

The two primary defect metrics are:

- *Defect Removal Efficiency* (*DRE*): The percentage of defects removed, either before shipment or for each phase of development
- *Defect Density*: The number of defects remaining in the product per KLOC.

There are benchmarks and engineering rules for both of these metrics, a few of which are:

- A DRE of 100% for a project is probably impossible. You will always have latent defects, even if you do not know they are there.
- Good software should be less than 2 defects per KLOC.
- Safe software should be 0.5–1.5 defects per KLOC.

We love defect data. It gives you real, observable events that allow you to understand how well the project is progressing, in terms of quality and schedule.

PROBLEMS

7.1 What does it mean if you have a higher arrival rate of defects for your project in the first month of system test than for other similar projects? List at least three options.

7.2 Given the defect removal matrix below, what is the defect removal efficiency for testing?

	Defect Injection Phase			
Defect Removal Step	Requirements	Design	Coding	Total
Requirements Review	10			10
Design Review	2	12		14
Testing	3	5	22	30
Customer Detected	1	3	4	8
Total	16	20	26	62

7.3 What is a reasonable number of defects in *good* quality released code?

7.4 Describe two ways in which you can predict the number of defects remaining after system test.

7.5 You have a system that is very similar to one that you built last year, except it is twice as large. In that system, you had 50 faults found in the field in year 1. You intend to roll out out this system twice as fast. How many faults do you predict to be found in year 1? Tests so far show that you have improved the defect density, as found in system test, by 50%. Discuss your reasoning.

7.6 True or False: Architecting to have small modules ($<$100 LOC) is a good way to reduce the number of defects in a system.

7.7 True or False: Defect density increases as system size increases.

7.8 True or False: A cost-effective way to find defects is to have a short testing cycle and then turn it over to real customers to use.

7.9 You have a one year development schedule, at which point your system is expected to be tested and 95% of the defects removed. You predict that you will have 200 defects in your system. How many defects, roughly, do you expect to find in month 8 if you are on schedule?

7.10 There are two different sets of data in Section 7.5.3. One is based on KLOC, the other on FPs. Do you think they are consistent? Why or why not?

PROJECTS

7.1 Project 1: You are now in system test. For theater tickets, you have the defect arrival data points below. Assume a Raleigh curve.

Month	1	2	3	4	5	6
Found Defects	13	22	25	22	17	5

(a) What do you predict as the total number of bugs in the system? Use two methods.

(b) How many bugs do you predict as being left in the system?

(c) What is the equation that predicts the defects?

(d) If you shipped at the end of week 6 (and assuming you removed all the defects found at that time), what would you predict as the defect removal efficiency?

(e) If this is a 10,000 LOC program, what would you predict as the remaining defect density after 6 months?

(f) Should you ship after 6 months? Why or why not?

7.2 Project 2: Download COQUALMO or another static defect prediction tool from the Internet. Use this tool to predict the number of defects inserted and removed by phase assuming a 10 KLOC program. Use your own judgment as to the adjustment factors to be used.

REFERENCES

[1] E. Yourdon and L. Constantine, *Structured Design*: *Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice-Hall, Englewood Cliffs, NJ, 1979.

[2] L. H. Putnam and W. Myers, "Familiar metric management—reliability," www.qsm.com, 1995. Accessed Sept. 1, 2005.

[3] W. Roberts, *Leadership Secrets of Atilla the Hun*, Warner Books, New York, 1995.

[4] L. H. Putnam and W. Myers, *Industrial Strength Software—Effective Management Using Measurement*, IEEE Computer Society Press, New York, 1997.

[5] L. Hatton, "Software failures: follies and fallacies," *IEE Review*, Mar. 1997.

[6] T. C. Jones, *Applied Software Measurements*, McGraw-Hill, New York, 1991.

[7] A. Wood, "Predicting software reliability," *IEEE Computer* **X**(II): (1996).

[8] C. Putnam and W. Myers, *Measures for Excellence*: *Reliability Software on Time, Within Budget*, Yourdan Press, Englewood Cliffs, New Jersey, 1992.

[9] S. Kah, *Metrics and Models in Software Quality Engineering, Second Edition*, Addison-Wesley, New York, 2003.

[10] M. Thangarajan and B. Biswas, "Mathematical model for defect prediction across software development lifecycle," www.qaiindia.com/Conferences/SEPG2000/sepg2000/sepg2_selectedPP.htm. Accessed Mar. 4, 2005.

[11] C. Hollenbach, "Quantitatively measured process improvements at Northrop Grumman IT," 2001.

[12] S. Chulani, "Modeling software defect introduction and removal: COQUALMO," University of Southern California Center for Software Engineering Technical Report USC-CSE-99-510, 1999.

[13] D. Reifer, "Software cost, quality, & productivity benchmarks," *The DoD Software Tech News*, July 2004.

[14] T. C. Jones, *Applied Software Measurements*, McGraw-Hill, New York, 1991.

[15] M. Diaz and J. King, "How CMM impacts quality, productivity, rework, and the bottom line," *Crosstalk*, 2002.

[16] T. C. Jones, "Software benchmarking: what works and what doesn't," *Software Productivity Research*, November 27, 2000. Available from www.cs.uml.edu/Boston-SPIN. Accessed Jan. 5, 2005.

[17] L. Hatton and A. Roberts, "How accurate is scientific software?," *IEEE Transactions on Software Engineering*, 1994.

[18] H. Remus, "Integrated software validation in view of inspections/review," *Proceedings of the Symposium on Software Validation*, North Holland, 1983.