Developing A Test Budget

Robert V. Binder

How Much Testing?

This month's column outlines an answer to a question I'm asked in every one of my testing seminars: how much time should we budget for testing in object-oriented development? There is no one-size-fits-all answer to this question. Ideally, we'd like to test until we have very good evidence that our system is sufficiently reliable for its intended use. But an open-ended testing budget is rare. Development budgets are nearly always constrained, so this question can be restated as *how much of our budget should be spent on testing*? In particular, with object-oriented development, we want to know how much testing effort should be allocated to cycles within an iterative incremental process. The following analysis yields a testing budget for object-oriented development.

1. Establish an overall project budget.

2 Identify and size each development increment.

3. Locate your project in the Scope/Criticality matrix to determine the appropriate overall testing effort.

4. Allocate overall testing effort to each increment and to system test.

5. Fine tune your plan.

Bracketed development and professional testing are assumed. Bracketed development refers to an incremental development strategy with $n + 2$ phases. In the first phase, general requirements and system architecture are established. This information allows the system to be *sliced* into development increments. Then each of the n slices is produced and integrated with existing slices. Slice development includes the production of testable OOA/D. Finally, system test is conducted upon all slices. Professional testing means that documented test plans with explicit coverage goals are prepared and executed. In contrast, testing-by-poking-around obtains where developers exercise code to their own satisfaction without using a prepared test plan.

This analysis result in a plan which spells out testing effort for each component to be developed. We'll consider each step in detail.

Establish an Overall Project Budget

Project budgets may be developed and denominated in many ways. For example, the scope of a system and its slices may be estimated by using function points. Cost, time, and capability form the "iron triangle" of project management -- an increase (decrease) in one must result in an increase (decrease) in another, other things being equal. For example, time or cost cannot be reduced unless capability is reduced. The trade-off between time and cost is not linear, especially in large projects. The relationship between the iron triangle and reliability is complex. A full discussion of sizing, cost estimation, and reliability analysis is beyond the scope of this article (see [3] for more on a bracketed development model.) I'll illustrate our test budget analysis using the hypothetical Acme Widget Order System and assuming a total project budget of 20,000 hours (think of a project staffed with roughly a dozen people for one year.)

Identify and Size Each Slice

Object-oriented systems are typically are designed, programed, and tested in relatively small increments, or *slices*. A slice is a subset of system capabilities developed in a single increment. Slices reflect an intersection of related capabilities, technical dependencies, and user/customer priorities. If some components are sufficient to support a reduced but coherent subset of features, we can defer development of remaining capabilities for a later slice. This is the "design a little, code a little, test a little" process applied to an entire application system.

Preprint. *Object Magazine,* June 1997.

The identification of slices is often a difficult and interesting puzzle. For example, suppose the first slice of the Acme Widget System requires the ability to set up orders, products, and customers. Shipment scheduling and billing can be deferred. The initial capabilities require an interface to a legacy system. For Acme Widget, we assume six slices whose relative sizes are 15%, 20%, 20%, 20%, 15%, and 10%.

Determine Total Testing Budget

Next, we determine how much of the total project budget should be spent on testing. The total budget reflects some larger economic calculation about the value of the system under development. Project budgets are finite -- we must always allocate scare resources to activity within our projects. So how much should we allocate to testing? Pre-release removal of bugs provides value by avoiding very expensive post-release fixes and the costs of system failures inflicted upon users/customers. Testing contributes value to the extent that it removes bugs which are more expensive or infeasible to find by other means (e.g., prototyping or inspections.) As the scope of a system increases, the effectiveness of non-test quality techniques decreases. So, the relative scope of the system under development provide one parameter for setting our test budget. Table 1 lists some size categories based on total effort.

The cost of a post-release bug also depends on the *criticality* of the application. Criticality is a qualitative spectrum of failure impacts ranging from nuisance to wide-scale loss of life and extensive physical damage. Table 2 identifies several generic levels of criticality. Be aware that the term "mission critical" has become part of the hypester's vocabulary and is often used as sales puffery. A third factor (not included in this simple model) could be termed a correction multiplier. The correction multiplier accounts for the cost of making and distributing the fix. If there are many copies of the application system in use, the distribution cost can be very high.

An appropriate testing budget reflects both scope and criticality. The scope/criticality matrix in figure 1 is based on ranges I've observed in actual projects. There is wide variation. As little as 15% to as much as 60 % of the total project budget can be spent on testing. Development organizations producing complex and highly critical systems typically spend three to five times as much on testing compared to developers of simpler, less critical systems.

Total Testing effort includes time to (1) design, document, and execute tests, (2) correct bugs which the tests reveal, and (3) retest and re-integrate bug fixes.

The total effort for the Acme Widget Order system (20,000 hours) corresponds to a high-medium scope. A worst-case failure in this system would delay order processing but would not result in more severe problems, so we assess the this system as *operation critical*. We'll peg this as a medium level of criticality. The mid-range values for scope and criticality suggest that roughly 30% of our total development (6,000 hours) should be spent on all forms of testing. This analysis is purposely fuzzy. In most situations, more precise quantification is either not justified or infeasible. These ranges should be interpreted according to particular situations.

Table 1. Scope Spectrum

Scope	**Total Project Effort**	**Example**
Small	Up to 3,000 hours total effort	Four people for 6 months or less.
Medium	3,000 - 30,000	Six people for 18 months.
Large	over 30,000 hours	Twenty-five people for one year.

Table 2. Criticality Spectrum

Criticality Level	**Immediate Worst Case Consequence of a failure**	**Example Application**
Trivial	Minor Annoyance	Self-study Program
Personal	Hours/days lost work	Word Processor
Operation Critical	Significant Operational Disruption	Payroll system, Order Processing
Business Critical	Significant Financial Loss	Electronic Funds Transfer
Mission Critical	Inability to Perform Vital Mission	Terrain-following Guidance System.
Safety Critical	Loss of Life	Automotive Anti-Lock Brake Control.

Figure 1. Scope/Criticality Matrix

	Low Criticality	**High Criticality**
Large Scope	15 - 30%	40 - 60 %
Small Scope	15 - 20%	30 -50 %

Table 4. Acme Widget Example

Phase	**Relative Size (% of total function points)**	**Test Budget (Hours)**	**Note**
System Definition			
Slice 1	15	720	15% of non-system test testing
Slice 2	20	960	20% of non-system test testing
Slice 3	20	960	20% of non-system test testing
Slice 4	20	960	20% of non-system test testing
Slice 5	15	720	15% of non-system test testing
Slice 6	10	480	10% of non-system test testing
System Test		2400	40% of total test allocation
		6000	30% of total project budget

Allocate Overall Testing Effort

The next step is to allocate overall testing effort to slice and system test. Experience in object-oriented development has shown that relatively more time should be spent on component testing than in system testing [2], [3]. Skimping on component testing leads to buggy systems as well as painful, ineffective, and expensive system testing. There is no magic number for system testing effort, but a 40 percent allocation is a workable balance. In turn, 60 percent of our effort should be spent on class and class-cluster testing. This percentage is a guideline that reflects known good practices and should be applied with engineering judgement.

We've allocated 6,000 hours of testing for the Acme Widget order system and plan six slices. This means there eight distinct phases (System Definition, Slices one through eight, and System Test.) Following our 40 percent guideline, we allocate 2,400 hours for system testing. We have 4,800 hours of testing time left to allocate (the remaining 60 percent of the total testing budget.) We begin by allocating test time to each slice in proportion to its relative size: 15 percent for slice one (720 hours), 20 for slice two (960 hours), and so on. Table 4 summarizes this analysis.

Allocate Component Testing Effort

The slice test budget is next allocated to constituent classes, clusters, or components. If

slice components have been individually sized, a good first order approximation is to allocate effort in proportion to component size. Alternatively, you can take a cut-and-fit approach. Consider each testable unit and decide what level of testing is warranted. Roughly the same analysis may be repeated, but on a smaller scale.

Plan System Testing Effort

How should we use our system testing budget of 2,400 hours? If you've developed use-cases, the FREE approach provides a very specific answer. In short, testing time is allocated to use-cases in proportion to their relative usage frequency. Details can be found in [1].

Fine Tuning

A budget developed by the foregoing analysis should be considered a first cut approximation, not a final plan. Its allocations are made in a top-down, systematic manner without considering special cases. There may be some components for which you have high confidence, others may be suspect. If you know something about particular components that indicates a different allocation, adjust accordingly. For example, suppose we have high confidence in the Acme Widget order, product, and customer objects since they are re-used from an existing business object library. We still need to test and integrate newly developed extensions of these objects, but we probably don't need extensive re-testing of member functions that have been previously verified. However, the legacy system interface is new and complex. In this situation, we'd want to spend a higher proportion of our testing budget on the legacy interface objects.

Here are few additional testing scope and productivity factors to consider.

- Developing automated test suites is software development and should be treated accordingly. Significant benefits can be obtained from test automation if it is used to support regression testing. With incremental development, this is both possible and advantageous. However, development of automated scripts and drivers requires a significant investment in time and can have a high maintenance cost if the system under test has many changes.

- Integration and regression testing weren't considered in this simplified budget model. They were assumed to be accomplished as a side-effect of incremental development, as successive slices use components tested in preceding slices. This is a pragmatic compromise, since slice-specific testing doesn't necessarily guarantee good exercising of system-wide dependencies. If slice test suites have been automated, we'll probably be able to re-run the tests for preceding slices at a relatively low cost. If not, some additional time should be budgeted for developing and executing a regression tests. In sufficiently large systems, a separate and independent integration test process should be defined and funded.

- If you need to reduce the time or cost of testing, increase the scope of the units (components) being tested. Instead of testing each class in a cluster, test the cluster or a component that uses the class. Focus on most-probable errors. Reduce or skip tests on capabilities that have been gotten careful developer attention. Use your suspicions or any other kind of analysis to identify special cases and unusual ways to use the system.

Finally, if it seems that your test budget is spread too thin, then reconsider your basic assumptions. Is your basic project budget is overly optimistic? Often, budgets are established without consideration for testing. Attempting to allocat test effort from such a budget is a zero-sum game -- someone must loose. Remember the "iron triangle" -- there is no free lunch in software development. If your budget analysis indicates that you won't be able to perform testing appropriate to scope and criticality, then something must give or someone must take responsibility for the inevitable reduction in operational reliability.

References

[1] Robert V. Binder, "Use-cases, Threads, and Relations: The FREE Approach to System Testing," *Object Magazine*, v 6, n 2, February 1996.

[2] Gail C. Murphy, Paul Townsend, Pok Wong, "Experiences with Cluster and Class Testing," *Communications of the ACM*, v 37, n 9, September 1994, 39-47.

[3] Matthew Pittman, "Lessons Learned in Managing Object-Oriented Development," *IEEE Software*, v 10, n 1, January 1993, 43-53.