When Should We Test?
An Economic Model

Kent Beck, Three Rivers Institute

When and how should you test? I'll build an economic model balancing four factors:

- The cost of maintaining the tests
- The cost of project chaos
- The value of feedback from tests
- The time value of money

Balancing these investments and returns, we conclude that automated tests written just in advance of implementation maximizes the profitability of testing.

Problem

When should we test?

- All together at the beginning of a project as a part of specification?
- Immediately before of implementation?
- Immediately after implementation?
- All together at the end of a project?

In the classical waterfall model, testing occurs at the end of a project lifecycle, immediately preceding deployment. In practice, testing time is often squeezed by delays in the previous phases. As a result, project managers are often faced with a nasty tradeoff between timely shipment and software quality. This leads to what would otherwise be ironic meetings to decide just how many critical defects a project can pass on to the customer, and exactly which defects are critical. Everyone loses in this scenario:

- Testers don't have time to do a thorough job. It is deeply unsatisfying to pass a product on to customers knowing that you are responsible for quality, and the product is defective.
- Project managers have a potent source of stress right at the end of the project, where there are already plenty of sources of stress.
- Customers receive software that simply doesn't work.

A specification-based lifecycle, where the tests are written all together in advance of any implementation, is a response to the unsatisfying results obtained from the waterfall's placement of testing. What if we wrote all the tests before beginning implementation? The tests then become part of project tracking. No tests work at the beginning of implementation. All the tests should work at the end of implementation. If the team is making good progress making tests work, the project as a whole is making good progress.

In the face of changes in scope, though, tests written far in advance of implementation are obsolete by the time they are addressed in the project. Older tests must be re-worked before they can be used, leading to unpredictable effort during implementation.

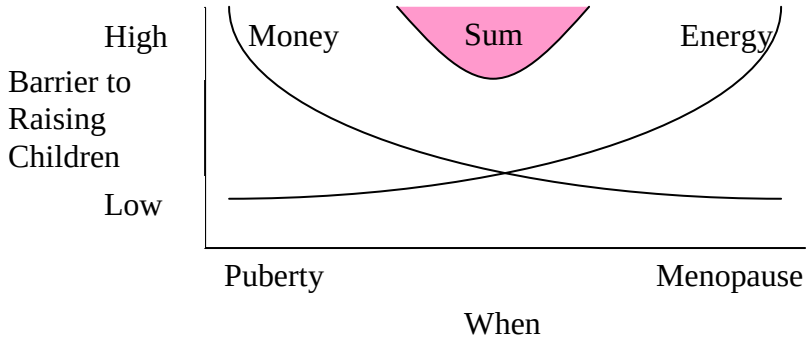
Given that each approach to testing has drawbacks, how can we wisely decide when to write tests? Are there other possible timings for testing better than either the specification or waterfall lifecycles?

We cannot find the one true answer to these questions, but by examining the economic tradeoffs involved perhaps we can help teams choose more wisely for themselves. In the remainder of the paper we will look at the cost of testing, the value of testing, and the profitability of testing (value – cost).

Trade-off Chart

We are trying to find a balance between conflicting constraints. One useful tool for analyzing such a situation is the tradeoff chart. A tradeoff chart plots two intersecting curves and their sum. By finding the minimum value of the sum on one axis, we can establish the optimum value on another axis.

For example, suppose we want to make a rational decision about when to have children (an obviously contrived example, but illustrative nonetheless.) On the one hand, the longer we wait, the less energy we have for raising them. If this was the only consideration, we would have children as early as possible. However, there's a contradictory force at work. The sonner we have children, the less financially able we are to pay for them. To make a rational decision, we have to take both factors into consideration.



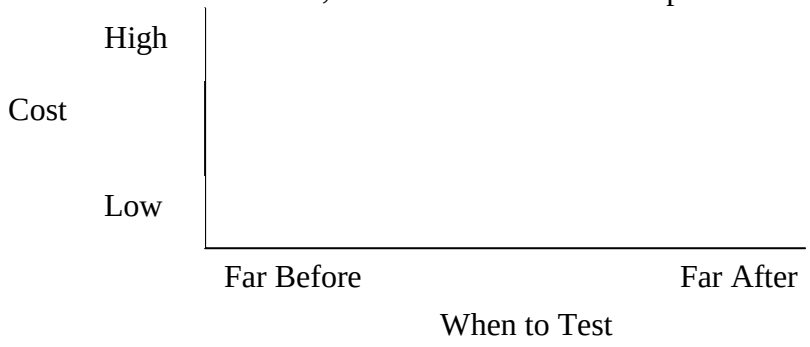
If we follow the "Sum" curve to its lowest point, we find the optimum moment to have children.

When to have children is influenced by a host of factors beyond the two listed here, energy and money, and subject to random biological and social factors. My goal when using trade-off charts is not to find the one best solution, but to inform my intuition about trends in the space. Choosing the units along each axis helps sharpen my thinking. Choosing the two contradictory trends helps sharpen my thinking. Choosing the other N-2 factors to ignore for the moment helps sharpen my thinking. Drawing the precise shape of each curve helps sharpen my thinking. The result is not a single answer, a "rational" decision as hinted above, but a mind better prepared to deal with the unique situations occurring in practice.

In analyzing when to test, we'll have an even more complicated situation. We'll have one chart for maximizing value and another for minimizing cost. In the end, we'll have to put the two together to identify trends towards maximum profitability. We'll start with cost.

Cost

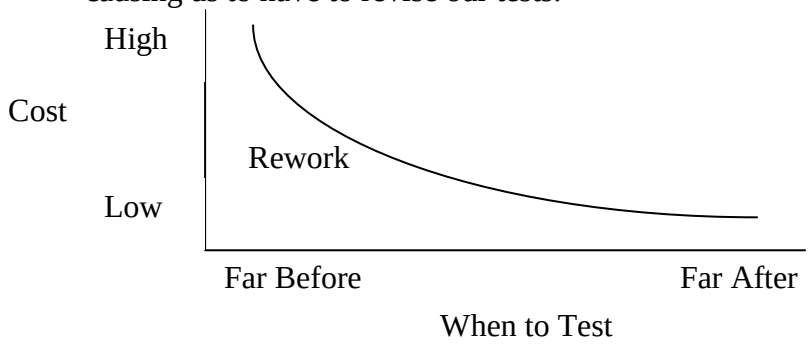
The vertical axis of our cost tradeoff chart has rising costs up the y-axis. The horizontal axis tracks the time to test, from far in advance of implementation to long after



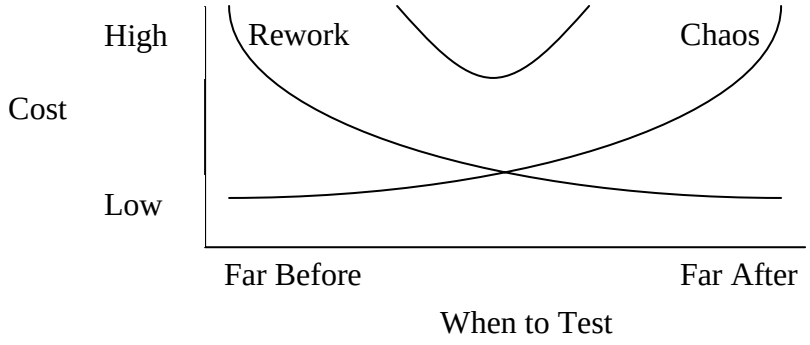
Our goal is to maximize profitability by choosing a testing time that best balances minimizing cost and maximizing value.

What are our two intersecting curves? The falling cost is the cost of reworking the tests. The further in advance of implementation we test, the more time we have to spend refurbishing the tests when we begin to implement the corresponding part of the system. The costs come in two forms:

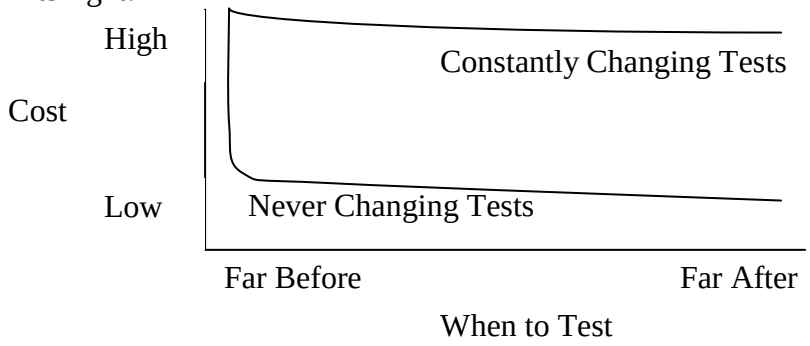
- Content—After implementation and real use, we are much more likely to know for sure what the right answers are.
- Form—After implementation, we know precisely what interface to use to invoke the function under test. Before implementation, the interface is likely to change, causing us to have to revise our tests.



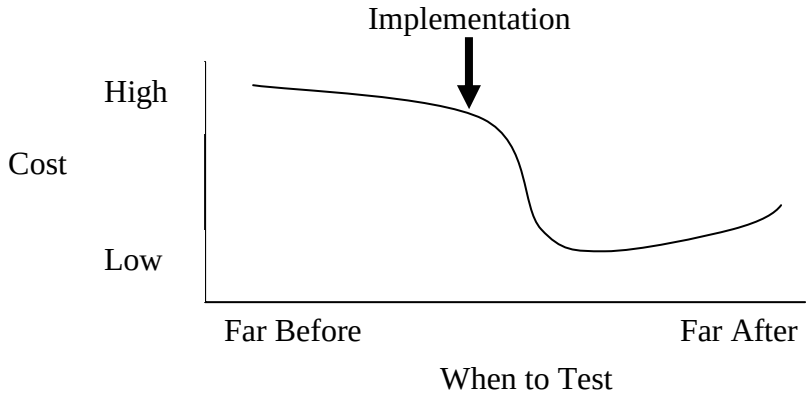
The rising cost in our trade-off chart is the cost of project chaos. The late stages of most projects are by far the least productive time. People are tired and grumpy, so they make more mistakes and are less likely to collaborate effectively. If we could remain effective throughout the project, we could get more done, lowering the effective cost per feature in the system.



As with all tradeoff charts, the x coordinate of the minimum depends on the actual shape of the curves. If we lower the cost of rework by testing more effectively, the optimal test time shifts left. If the cost of rework rises sharply because of rapid learning, the test time shifts right:



The real shape of the curves is certainly not smooth. Immediately after implementation, the cost of rework drops dramatically. Then, the cost of rework actually rises when testing far after implementation, because the team has to recreate their understanding of the circumstances under which the software is supposed to work:

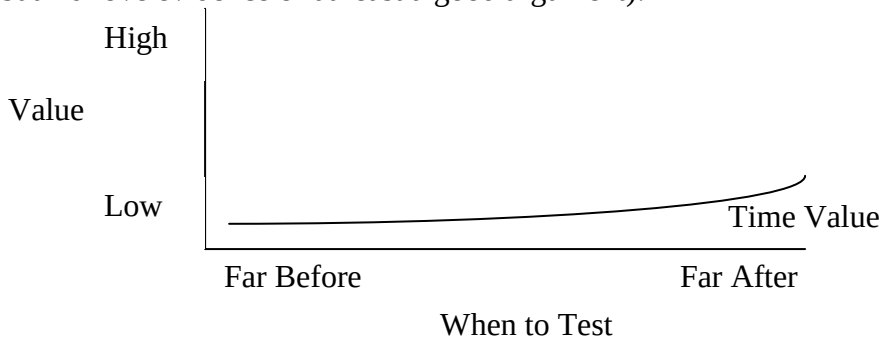


We can’t find the single lowest cost testing time because project, team, organization, and technology factors affect the shapes of both curves. The general conclusion is that, like Goldilocks, we would like our testing to be not too late and not too early, but just right.

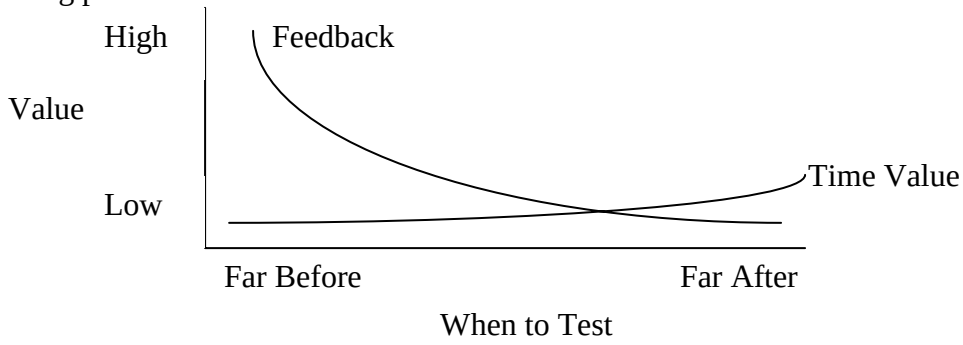
Cost is only one component of profitability. We need to also consider value. Fortunately, the value picture will be a bit less ambiguous.

Value

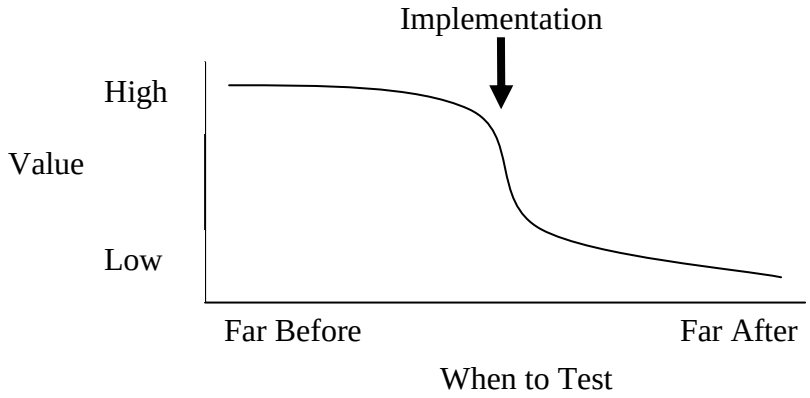
The rising curve is the time value of money—the more we delay testing investment, the better the net present value of the project. Intuitively I believe this is not a strong factor (but I'd love evidence or at least a good argument).



The falling curve is the tracking value of the tests. The later we test, the less feedback testing provides.



This time, the rising curve really is smooth. The feedback value of testing, however, falls sharply after implementation. Tests written in advance of implementation serve to improve the design of the software, reduce the scope of implementation by specifying scope precisely, and track the progress of implementation:



If my assumption about the time value of money having a much smaller effect than the feedback value is correct, we can conclude that to maximize value, the earlier we test, the better.

Conclusion

What happens when we put the two curves together? Again, we can't make a single once-and-for-all conclusion, but the trend is clear. Value drops and cost rises as soon as implementation begins.Testing before implementation is always superior to testing afterwards. We don't want to get too far ahead of ourselves, though, or the cost of reworking the tests starts to eat into our profits.

The conclusion I reach, not surprisingly if you know anything about Extreme Programming, is that testing just in advance of implementation maximizes profitability. In Test-Driven Development, tests are written piecemeal, just in advance of implementation. TDD takes place at two time scales. Every week the testers, analysts, and business sponsors on the team collaborate to write system-level tests for that week's features before any implementation begins. Then, during a programming episode, two programmers write a much smaller-scale test every few minutes, again in advance of implementation or physical design. The cost is minimized because the tests are written so close to implementation, and because the team can incorporate everything they've learned from previous test/implementation cycles in writing the next test. The value is maximized by defering investment in testing as long as possible while still realizing the pre-implementation feedback.

Okay, but how to test in advance of implementation? Most testing is done manually. It makes no sense to manually test a system that doesn't yet exist. If we choose to test in advance of implementation, we have to write automated tests. While test automation is thought to be prohibitively expensive, recent advances in technique and tools have substantially reduced the costs[1].

This model of testing profitability is based entirely on gut feel. More careful modeling and measurement are needed to validate the model. In particular:

- Hypothesis—The cost of reworking the tests rises dramatically over time. We could validate this hypothesis by tracking the cost of writing tests initially and the cost of modifying them later, and then correlating the cost of maintenance with the age of tests at implementation.
- Hypothesis—The cost of automated testing is actually less than the cost of equivalent manual testing. We could validate this hypothesis by tracking two equivalent projects, carefully accounting for the time spent on testing to reach an equivalent level of quality.
- Hypothesis—The project estimation and tracking value of early tests is small. This is linked to the rework hypothesis. If rework is common, then the project estimation and tracking value of tests is small. If there is an alternative project estimation and tracking practice providing similar benefits to using the tests for

[1]Tools like xUnit for programmers and FIT (http://fit.c2.com) for analysts and testers enable TDD at the level of unit and system tests, respectively.

tracking, you can defer test writing without reducing value. We could validate this hypothesis by measuring the number of new tests that need to be written after the testing phase of the specification lifecycle.

- Hypothesis—Maintaining automated tests is cheap. If the cost of maintaining automated tests is too high, we will have to settle for manual testing regardless of the constraint it places on us to defer testing until after implementation. We could validate this hypothesis by tracking time spent maintaining tests. I predict that the use of automated refactoring tools will dramatically reduce the cost of maintenance.
- Another area for further research is the place of testing profitability in the entire value chain of software development. Without looking at the entire system by which software value is created, we risk maximizing testing profitability but lowering overall profitability.

The profit to be gained from testing is greatest when automated tests are written piecemeal just in advance of implementation. That is the conclusion suggested by this model. Your mileage may vary. If the shape of the cost and value curves on your project are different from those presented here, or if other consideration are more important, you will find a different optimal time for writing tests.

Acknowledgements

Thanks to Ron Jeffries and Ward Cunningham for timely, insightful, and gentle corrections. Thanks to Jeff Grigg for spotting the flaw in the opening example.