AP-M Appropriate Process Movement

Cost of Change - Modernised

Paul Oldfield

Mentors of Cally

Member of the Appropriate Process Movement

http://www.aptprocess.com

Abstract

The Boehm model of Cost of Change has been generally accepted for years now, and has been used as the basis for arguments in favour of heavyweight processes that attempt to prevent errors early in the development cycle and at all stages thereafter. Various proponents of Agile methodologies have challenged the validity of the model. This paper tries to form a basis on which the arguments may be evaluated, should figures ever be measured.

Copyright Note

Cost of Change - Modernised

The Boehm model of Cost of Change has been generally accepted for years now, and has been used as the basis for arguments in favour of heavyweight processes that attempt to prevent errors early in the development cycle and at all stages thereafter. Various proponents of Agile methodologies have challenged the validity of the model. This paper tries to form a basis on which the arguments may be evaluated, should figures ever be measured.

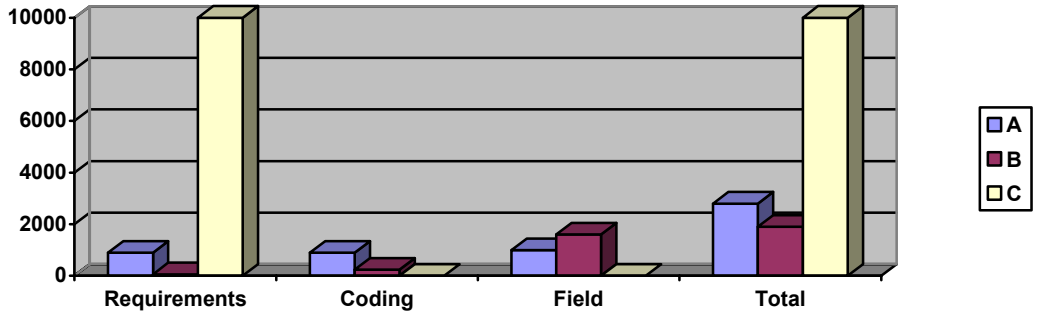
There are many variants on the traditional view; let us consider one common variant. This asserts that where the cost of fixing an error discovered during the requirements phase is 1 unit, the cost of fixing that error if it is discovered during coding is 10 units, and the cost of fixing that error if not discovered until the software is delivered is 100 units. This assertion is supported by measurement; the costs to fix an error are more or less as quoted.

Unfortunately, the costs of change in requirements also follow this pattern. The problem with this observation is that a change in requirements is not so readily amenable to prevention. One could always disallow changes to requirements, but this would mean delivering software that does not meet the needs. If the change in requirements is genuine, then the cost and benefit of allowing the change needs to be taken into consideration. Extra analysis cannot prevent all changes to requirements; some occur as a result of changes to external conditions that could not readily be predicted. Requirements change; we need to accept this as a reality, and deal with the consequences.

Kent Beck, one of the originators of the Agile Alliance, posed a question – "What if the cost of change curve were flatter?" This would have some interesting consequences. Suppose the ratio instead of 1:10:100 were 1:3:9 or even 1:2:4? It would still be a geometric progression, but the impact of missing an error would be greatly reduced. This means we could employ lightweight processes where the cost of process would be greatly reduced, but the risk of missing an error might be increased owing to the reduction of crosschecking.

There are no figures to support any suggestion that Beck's "what if" scenario is a true representation of what happens with agile processes. Indeed, Alistair Cockburn makes a convincing argument that the ratio will still be 1:10:100. What agile approaches do instead is to provide a dramatic reduction to the unit cost. How dramatic a reduction will depend on how agile the approach that is put into place.

Below I give cost figures for 3 scenarios. The projects are identical; each starts having 100 errors in the requirements. In Scenario A we consider costs for a hypothetical heavyweight process that allows 10% of defects to slip through at each stage to be caught later. It has a unit cost of $10. In Scenario B we consider costs for a hypothetical agile process that allows 40% of defects to slip through at each stage to be caught later. It has a unit cost of $1. In Scenario C we consider a hypothetical rigorous safety-critical process that allows no defects to slip through, but has a unit cost of $100.

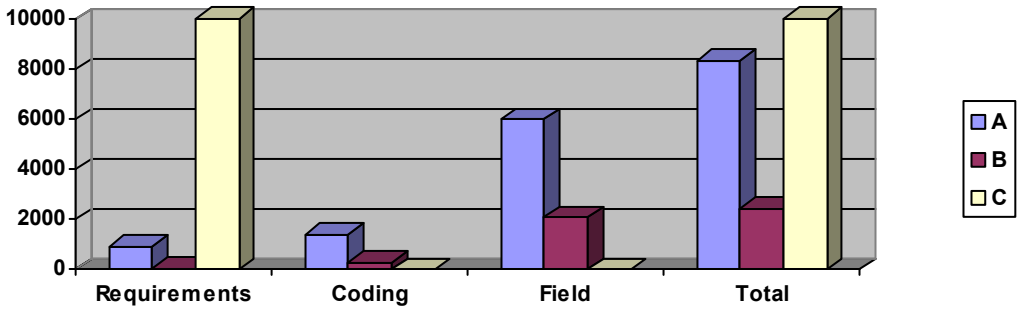


Let me first say that these scenarios are in no way meant to be realistic, they are to give us a feel about how we may reason about costs.

Scenario A gives us a fairly steady cost, 90 errors detected during requirements fixed at $10 apiece; 9 errors detected during coding fixed at $100 apiece and 1 error detected after deployment, fixed at a cost of $1000. Total cost for 100 errors of $2800. Scenario B gives rising costs. The initial cost for finding and fixing 60 errors during requirement at $1 apiece is excellent value for money. A further 24 errors were found and detected during coding at $10 apiece, still good value for money. Unfortunately that leaves 16 errors undetected until after deployment, to fix at $100 apiece. Of the total $1900 spent on fixing errors, the bulk of this, $1600, is spent fixing errors after

deployment. Scenario C really hammers the errors during requirements; 100 errors at $100 apiece, giving a total cost of $10000. However, no errors are released to field, and in a safety critical system this is extremely important. Scenario B gives the lowest overall cost, but it requires the customer to be part of Quality Assurance, and customer expectations need to be managed in this respect. Scenario A cost more, and still released one error to be found after deployment.

Now let us consider changing requirements. Suppose in addition to the 100 errors in requirements we have 15 changes in requirements that could not reasonably be anticipated. Let us allow that Scenario C is a special case and either doesn't get hit by changing requirements or does so much extra work up front that they are caught and dealt with for no extra cost. However, Scenario A and B both get hit, with 5 changes occurring in each phase. How do the figures look now?



For Scenario A, the cost of change adds $50 during requirements, $500 during coding and $5000 after deployment. For Scenario B the cost of change adds $5 during requirements, $50 during coding and $500 after deployment.

Clearly, if these scenarios were realistic, then Scenario B would be the one to aim for where requirements are likely to change. However, if we want to inject realism, then it is likely that the changes would be loaded toward the back end – arising from feedback once the users get to use the software that has been deployed. That makes Scenario B even more attractive.

It may not be easy to find realistic figures to use when costing out alternative scenarios. However, one lesson should be clear – in cases where the requirements are going to change anyway, it is sensible to bring the unit costs for change down to a fairly low figure, and accept the risk that this approach may let more errors through the system.

Robust approaches document, review and analyse requirements thoroughly, then perform rigorous, detailed, documented and reviewed design, before coding and reviewing. All this is to eliminate as many errors as possible. On top of this, each major step in the process will be traced from, and back to, the requirements. This addition is to ensure that when change does occur, the relevant bits of requirement, design, code and documentation can be updated. The test suite also needs the tests to be altered to match the new requirements. In addition to these updates, the traceability needs to be restored. It is little wonder that the need for change is expensive.

An agile approach may only require changes to the tests and the code. There is little or no traceability so there need to be alternative ways of locating the parts that need to change. Documentation will be much lighter, but may still require alterations to be made. In general, as long as the cost of locating the parts that need to change can be kept low, the cost of change will be considerably cheaper than for robust approaches.