Change Costing in a Maintenance Environment

ABSTRACT

Over the years, as a software product is maintained, the factors involved in costing a change become much more complex than most customers and vendors realize. Failure to understand and properly deal with these factors frequently leads to cost overruns, schedule impacts, and dramatic declines in quality and reliability. This presentation will address the root cause of this problem. As requirements are recursively modified over time, basic software architecture assumptions become eroded. Users continue to subject the software to new scenarios neither ‘envisioned” nor accommodated by the original design and test strategies. Repair and change histories must be added to the long lists of traditional considerations to be addressed by the software maintainers. The cost impact of these additional activities are not readily recognized because software is an invisible abstract entity. Common resource control consideration algorithms based on lines of code, function points, and numbers of test cases break down because they do not include all of the terms required, and inherently apply improper weighting factors to the terms they do include. A systematic approach is provided for developing a more accurate change costing algorithm in the maintenance environment.

Even small changes can require large efforts depending on the sensitivity of the pre-existing software and architecture to the change being implemented. For new systems, the amount of code (for example, “lines of code”) being added or modified is a significant factor in determining the cost of the change. However, for older software systems, there are much more dominant factors involved in correctly making a change and assuring that the change has not inadvertently decreased the quality and functionality of legacy code.

The number of “modules” and “functions” affected by each change have far more impact on the total cost of changes as the software ages. Other “factors” including added or modified “code paths” and “execution scenarios” become extremely significant. Evolving “requirements complexity” also dramatically adds to the cost of a change in the maintenance environment, as programmers endeavor to unravel the total “net intent” of recursively modified requirements. Every new change needs to be carefully analyzed in-depth to assess the compatibility of that change with the assumptions on which the architecture was originally based. (War stories abound with examples of “trivial code changes” which inadvertently violated software design or architecture assumptions leading to serious software failures.) Each of these factors represents a very important independent component of the total implementation cost. Software projects which do not thoroughly perform the activities represented by each of these factors suffer because their software products have poor performance and quickly become un-maintainable. Those projects which do attempt to perform all the appropriate change implementation activities must accurately cost them (dollar expense and schedule impact) or they will suffer unacceptable overruns and schedule delays.

An effective approach to deal with these factors involves empirically weighting each factor individually based on experience and data retained from similar projects. The weights can be in any dimensions chosen to result in cost units compatible with project conventions. One advantage of such a technique is that it provides a quantitative justification for each separate component of the total cost, which can be challenged and validated individually. This is an extremely important credibility attribute when predicting change costs for skeptical project management and customers. Continuous calibrating of the factor weighting is possible as new data is recorded for successive changes. It is even possible to have sets of weights which uniquely apply to processes of different maturity levels. For example, one set of weights would be applicable to a software project using a CMM Level 2 process while another set of weights would apply for a project on which a CMM level 4 process is required. Accuracy of these weights increases with the expertise and repeatability of the maintenance personnel involved.

Thus, the total cost of a change in the maintenance environment can be more accurately expressed by an algorithm consisting of multiple terms with individual weights, representing the complete set of cost factors for the entire change activity. Maintainers can more systematically identify the cost components using this algorithm. The algorithm lends itself to automated costing processes and cost modeling. It provides a means of explaining to customers and sponsors what is involved.
in making a change to an existing system and in determining and justifying the true cost of such changes.

Ted Keller has over 27 years experience in the computer industry and is recognized internationally as an expert in Software Processes, Software Reliability Engineering and Total Quality Management. He has presented over 200 tutorials and lectures on these subjects across North America, Europe and Japan. For the past 20 years Mr. Keller has played a lead role in the synthesis and continuing evolution of one of the world's most mature software processes by which virtually error-free, life-critical software has been produced consistently for America's Space Shuttle. During the past ten years this project has received the NASA Excellence Award for Quality and Productivity (George M. Low Trophy) twice, the IBM "Best Software Lab" award twice, was evaluated to be at Process Maturity "Level 5" (highest) on the Software Engineering Institute scale by a NASA evaluation team trained by the Carnegie Mellon University, and was "unconditionally" ISO 9001 Certified. In 1988 Mr. Keller was awarded the NASA Public Service Medal for his "Contributions to the Space Shuttle Return to Flight Activities." In 1991 he was selected for the Institute of Electrical and Electronics Engineers Reliability Society "Distinguished Lecturers Program." and he was the recipient of that society's 1993 Reliability Award for his "Contribution to the Development of Highly Reliable Software."