Measuring Software Process *Efficiency*

By Gary Gack, Process-Fusion.net

This article is the second in a series of three. The first article, "Measuring Software Process Effectiveness" describes use of the defect containment metric to quantify effectiveness of alternative appraisal strategies. In the third article I describe an approach to modeling and managing efficiency and effectiveness that incorporates both prospective (leading) and retrospective (lagging) indicators. The model combines the information presented in the first two articles to facilitate "simulation" of alternative strategies and provides quantitative indications of the impact of alternatives under consideration.

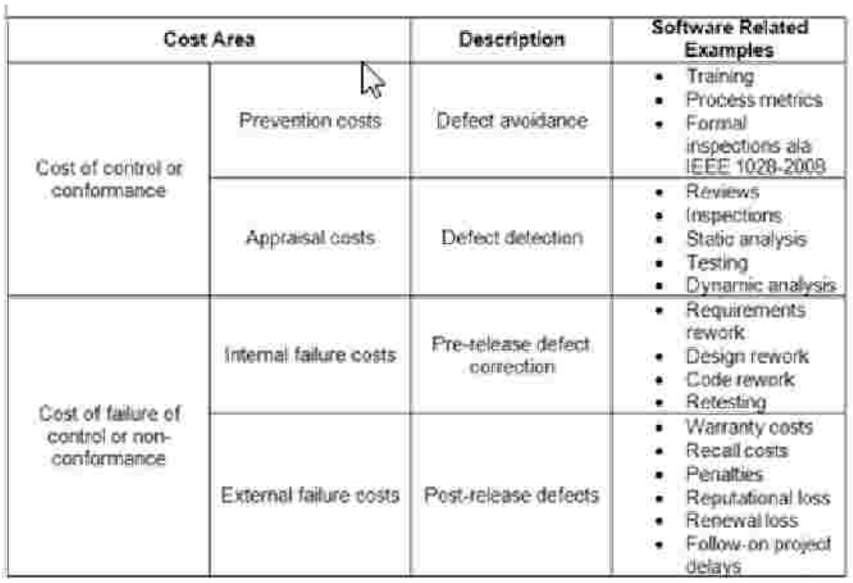
"Efficient"? What does that mean?

A software process is "efficient" if, relative to an alternative, it produces an equivalent or better result at lower cost. In our present context, for example, if appraisal strategy "A" finds the same number of defects as does appraisal strategy "B" (i.e., the two strategies are equally *effective*), but does so at lower cost, strategy "A" is more *efficient* that strategy "B". A is more "productive" than B.

ORIGINS AND EVOLUTION OF COST OF QUALITY IDEAS

Cost of (Poor) Quality Frameworks

First described in a 1956 *Harvard Business Review* article titled "Total Quality Control," Armand Feigenbaum introduced an approach to categorizing quality-related costs that transcended traditional cost accounting practices. This framework (as it relates to software) is summarized here.



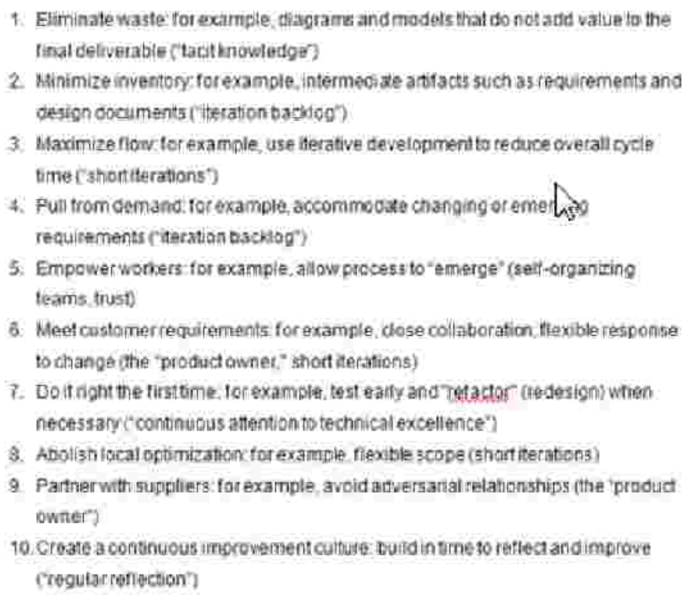
Cost Area		Description	Software Related Examples
Cost of control or conformance	Prevention costs	Defect avoidance	• Training • Process metrics • Formal inspections ala IEEE 1028-2008
	Appraisal costs	Defect detection	• Reviews • Inspections • Static analysis • Testing • Dynamic analysis
Cost of failure of control or non-conformance	Internal failure costs	Pre-release defect correction	• Requirements rework • Design rework • Code rework • Retesting
	External failure costs	Post-release defects	• Warranty costs • Recall costs • Penalties • Reputational loss • Renewal loss • Follow-on project delays

James Harrington's 1987 book *Poor Quality Costs* introduced a refinement of Feigenbaum's approach intended to emphasize that investment in detection and prevention leads to lower total cost.

Lean and Agile Methods

Lean methods and tools focus on identifying and reducing non-value-added elements of CoQ. Lean ideas originated in the Toyota Production System in a manufacturing context and have since been adapted to many other areas of business activity, including services, information technology (IT), and software development. As applied to software development, key practices of Lean may be mapped to software development, as shown here.

10 basic practices of Lean, with software translations (linked to Agile ideas)

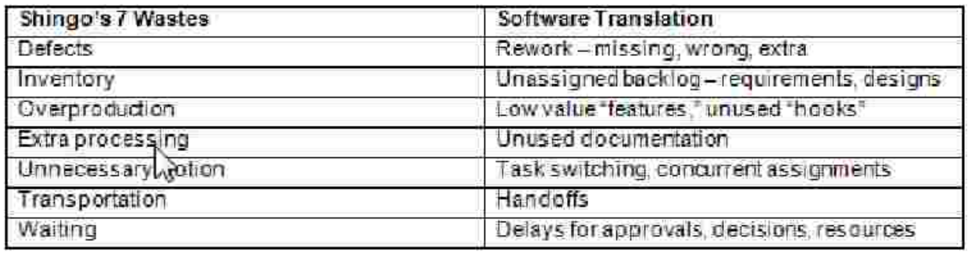


1. Eliminate waste: for example, diagrams and models that do not add value to the final deliverable ("tacit knowledge")
2. Minimize inventory: for example, intermediate artifacts such as requirements and design documents ("iteration backlog")
3. Maximize flow: for example, use iterative development to reduce overall cycle time ("short iterations")
4. Pull from demand: for example, accommodate changing or emerging requirements ("iteration backlog")
5. Empower workers: for example, allow process to "emerge" (self-organizing teams, trust)
6. Meet customer requirements: for example, close collaboration, flexible response to change (the "product owner," short iterations)
7. Do it right the first time: for example, test early and "refactor" (redesign) when necessary ("continuous attention to technical excellence")
8. Abolish local optimization: for example, flexible scope (short iterations)
9. Partner with suppliers: for example, avoid adversarial relationships (the "product owner")
10. Create a continuous improvement culture: build in time to reflect and improve ("regular reflection")

Value Stream Mapping and Elimination of Waste

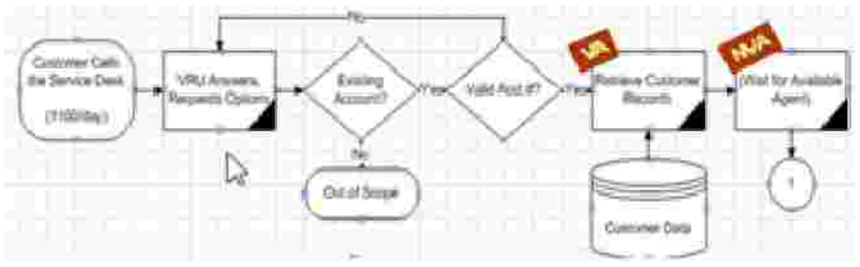
In his work with the Toyota Production System, Shingo (1981) identified seven wastes commonly found in manufacturing processes. These wastes have parallels in software development, as described below (based loosely on Poppendieck (2007)).

Shingo's 7 wastes translated to software development



Shingo's 7 Wastes	Software Translation
Defects	Rework – missing, wrong, extra
Inventory	Unassigned backlog – requirements, designs
Overproduction	Low value "features," unused "hooks"
Extra processing	Unused documentation
Unnecessary motion	Task switching, concurrent assignments
Transportation	Handoffs
Waiting	Delays for approvals, decisions, resources

These wastes may be discovered in any process by applying one of the most popular Lean tools--value stream mapping (VSM). As illustrated by this highly simplified example, the VSM method (of which there are many variants) annotates various facts and categorizes the steps in a process map to differentiate those that are "value-added" (VA) from those that are "non-value-added" (NVA), that is, steps that are among the seven wastes.

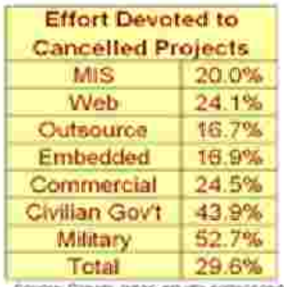

No
Customer Calls the Service Desk (7100/day)
VRU Answers, Requests Options
Existing Account?
No
Out of Scope
Yes
Valid Acct #?
Yes
VA
Retrieve Customer Records
Customer Data
NVA
(Wait for Available Agent)
1

These terms are used from a customer's perspective, that is, VA activities are those that a customer would choose to pay for, given a choice. Certain activities, such as testing, are certainly necessary but are not value added from a customer's perspective. Given a choice, the customer will always prefer companies produce a perfect product the first time. Certain other activities (for example, regulatory compliance), while not truly value added, may nonetheless be required. As it applies to software development, all appraisal (defect-finding) and rework (defect-fixing) activities are, by definition, NVA, which is not to say they are unnecessary, merely that they should be minimized.

Software Cost of Quality Benchmarks

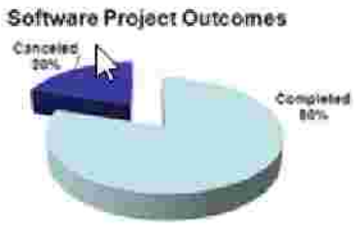
Harry and Schroder (2000) estimate that CoQ for an average company is approximately 15 to 25 percent of sales. Based on available industry benchmarks, the author believes average NVA costs for software organizations are in the area of 60 percent of total effort--while not a direct comparison, it is clear CoQ in software is higher than nearly any other area of business activity.

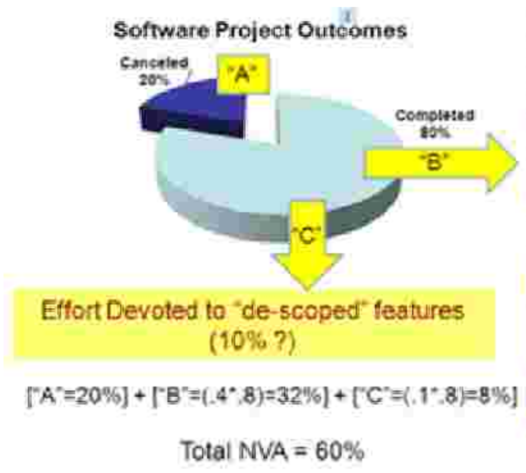
In software, NVA has three principal components. As illustrated here, cancelled projects in the MIS domain account for around 20 percent of total software effort.

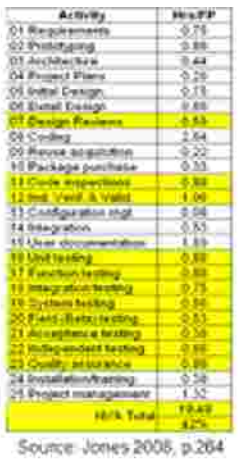


Effort Devoted to Cancelled Projects	
MIS	20.0%
Web	24.1%
Outsource	16.7%
Embedded	16.9%
Commercial	24.5%
Civilian Gov't	43.9%
Military	52.7%
Total	29.6%

Source: Capers Jones, private correspondence







Activity	Hrs/FP
01 Requirements	0.75
02 Prototyping	0.88
03 Architecture	0.44
04 Project Plans	0.29
05 Initial Design	0.75
06 Detail Design	0.88
07 Design Reviews	0.53
08 Coding	2.64
09 Reuse acquisition	0.22
10 Package purchase	0.33
11 Code inspections	0.88
12 Ind. Verif. & Valid.	1.06
13 Configuration mgt.	0.06
14 Integration	0.53
15 User documentation	1.89
16 Unit testing	0.88
17 Function testing	0.88
18 Integration testing	0.75
19 System testing	0.66
20 Field (Beta) testing	0.53
21 Acceptance testing	0.38
22 Independent testing	0.66
23 Quality assurance	0.88
24 Installation/training	0.38
25 Project management	1.32
NVA Total	19.49 42%

Source: Jones 2008, p.264

As illustrated here, the second component includes appraisal and rework activities (shaded items labeled "B") and also includes effort devoted to "de-scoped" features ("C") that were partially completed and then dropped prior to delivery, typically due to schedule or cost overruns. The author is unable to find solid benchmark data for this element, but from experience something around 10 percent seems reasonable. As indicated, A+B+C equals approximately 60 percent of total effort. (Note: This includes pre-release rework only--when post-release rework is added, the picture is even worse.) Certainly some organizations do better than these averages suggest, but a significant number are even worse. As illustrated in the third article in this series, best-in-class groups using a sound combination of best practices will reduce NVA by two-thirds or more.

Many groups are skeptical that NVA is so high, but when actually measured, it is invariably much higher than believed beforehand.

Effort Accounting in a Software Cost of Quality Framework

Adapting these ideas to software development provides a robust approach to objectively answering several age-old questions retrospectively: "How are we doing?" "Are we getting better?" Clearly things are getting better if the VA percent of total effort is improving relative to a baseline. A software CoQ framework provides a comparatively simple approach to effort accounting that avoids many of the traps and complications that often defeat efforts to measure productivity. To apply this framework, we require *less* effort accounting detail than is typically collected, yet we gain *more* understanding and insight from what is collected. Using the CoQ framework moves us to the most useful level of abstraction—to see the forest, not just the trees.

Applying this framework does not require time accounting to the task level; rather, we collect data in a simplified set of categories for each project as follows:

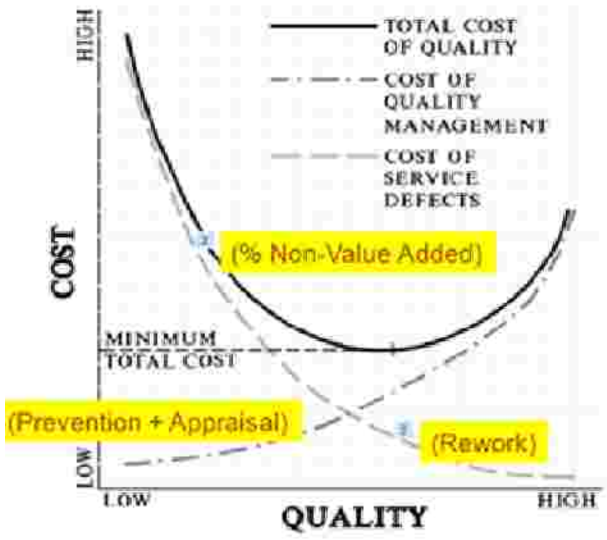
- Project ID
- Phase or iteration (note these include "post-release")

- VA effort within each phase or iteration, that is, all effort devoted to the creation of information and/or artifacts deemed necessary and appropriate (for example, requirements elicitation, architecture definition, solution design, construction)
- NVA effort within each phase or iteration:
    - All appraisal effort (anything done to find defects) per appraisal method (in many instances only one form of appraisal, such as inspections or testing, will be used in a single phase--if more than one method is used during the same phase, the effort associated with each should be separately recorded)
    - All rework effort (anything done to fix defects)
    - All effort devoted to cancelled projects (may require reclassification of effort previously recorded)
    - All effort devoted to effort previously categorized as VA that is related to features or functions promised but not delivered (that is, "de-scoped" features) in the current project or iteration.
- Prevention effort, that is, effort devoted to activities primarily intended to prevent defects and waste in any form. Many of these activities will be projects in their own right, separate from development projects per se, for example, training, SEPG activities, and process improvement initiatives generally. Some prevention activities, such as project-specific training, may occur within a development project. Certain activities, including formal inspections such as IEEE Std. 1028-2008, may serve a dual purpose of both appraisal and prevention. When in doubt a conservative approach suggests categorizing these as appraisal.

Many groups are accustomed to collecting time accounting at the task level. Many believe these data will prove useful for future estimates. In practice, however, few actually use the data collected. Jones (2008) reports time is sometimes under-reported by 25 percent of total effort. A few organizations actually use task-level data for earned value analysis, and many simply collect data at that level of detail because they always did it that way and/or because time reporting is tightly integrated with project status tracking and no lower-overhead option is available. In instances where task-level reporting cannot be discontinued in favor of a simpler CoQ structure, it is nearly always possible to map tasks to CoQ categories.

The Cost of Quality Message

As illustrated by here, judiciously chosen increases in prevention and appraisal will reduce rework and lead to an overall increase in the portion of total effort that is value added. Judiciously chosen means the application of best practices at the right time in the development lifecycle.



The diagram at the right reflects experience in manufacturing contexts – it needs to be qualified somewhat in a software context. It turns out, as I will explore in the third article in this series, that the prevention and appraisal curve for software is different than it is in manufacturing.

The "Productivity" Trap

Efficiency and *productivity* are essentially synonyms. Hence, the VA to NVA ratio measures both efficiency and productivity. Many attempts have been and are made to measure productivity using a more traditional approach in which productivity is defined as the ratio of "input" (generally person hours/days/months of effort) to "output" (generally function points or lines of code). This approach works well in a manufacturing context, but is seriously flawed when applied to software. Why is that?

1. Time accounting systems that attempt to measure "input" are rarely accurate. According to Capers Jones time accounting systems commonly under-report by 25%. Distortions are sometimes the result of political factors that lead to time being charged where budget is available rather than where the work is actually being done. A simple reconciliation of headcount times nominal work week is rarely performed as a sanity check on reported hours. Often we see more "form" than "substance". Lots of overhead and annoyance, little useful information. Accuracy of reporting is generally inversely related to the number of charge categories in use. A simpler CoQ reporting framework will invariably be more accurate than will task level reporting commonly used.
2. Size measures, even when actually done, are expensive and often incomplete. In many cases size is measured at the start of a project, but not at the end. In such

cases “productivity” may be inflated due to features not delivered (“de-scoped” due to schedule or cost overruns).

3. This approach fails to account for canceled projects. It does not consider the post-delivery costs due to poor delivered quality. If post-release rework costs are 50% of the original project budget (not a rare occurrence) true productivity is actually greatly lower than a measure determined at the time of delivery.

In short, for all of the above reasons, the VA / NVA approach will give far more meaningful results at significantly less cost compared to the traditional approach.

REFERENCES

Feigenbaum, A. 1956. Total quality control. *Harvard Business Review*.
Harrington, J. 1987. *Poor quality costs*. Milwaukee: ASQ.
Harry, M. J., and R. Schroder. 2000. *Six Sigma: The breakthrough strategy***.** Doubleday.
Jones, C. 2007. *Estimating software costs*, 2nd edition. New York: McGraw Hill.
Jones, C. 2008. *Applied software measurement,* 3rd edition. New York: McGraw Hill.
Jones, C. 2010. *Software engineering best practices.* New York: McGraw Hill.
Poppendieck, M. 2007. *Implementing lean software development.* Boston: Addison-Wesley.
Shingo. 1981 *A study of the Toyota Production System*. New York: Productivity Press.

BIOGRAPHY

Gary Gack is the founder and president of Process-Fusion.net, a provider of assessments, strategy advice, training, and coaching relating to integration and deployment of software and IT best practices. Gack holds an MBA from the Wharton School, is a Lean Six Sigma Black Belt, and is an ASQ Certified Software Quality Engineer. He has more than 40 years of diverse experience, including more than 20 years focused on process improvement. He is the author of many articles and a book entitled *Managing the Black Hole: The Executive’s Guide to Software Project Risk*. LinkedIn profile: http://www.linkedin.com/in/garygack.