

Cost of Quality (CoQ) Metrics for Telescope Operations and Project Management

Nicole M. Radziwill

National Radio Astronomy Observatory, P.O. Box 2, Green Bank, WV 24944 USA

ABSTRACT

This study describes the goals, foundational work, and early returns associated with establishing a pilot quality cost program at the Robert C. Byrd Green Bank Telescope (GBT). Quality costs provide a means to communicate the results of process improvement efforts in the universal language of project management: money. This scheme stratifies prevention, appraisal, internal failure and external failure costs, and seeks to quantify and compare the up-front investment in planning and risk management versus the cost of rework. An activity-based Cost of Quality (CoQ) model was blended with the Cost of Software Quality (CoSQ) model that has been successfully deployed at Raytheon Electronic Systems (RES) for this pilot program, analyzing the efforts of the GBT Software Development Division. Using this model, questions that can now be answered include: What is an appropriate length for our development cycle? Are some observing modes more reliable than others? Are we testing too much, or not enough? How good is our software quality, not in terms of defects reported and fixed, but in terms of its impact on the user? The ultimate goal is to provide a higher quality of service to customers of the telescope.

Keywords: Quality, quality costs, work breakdown structure, WBS, process improvement, metrics

1. INTRODUCTION

Software is the window through which many observers perceive whether a telescope's operations are high quality or not. If the proposal cannot be submitted, telescope time cannot be granted and the observation is not possible. If the hardware is faulty, the software will not function properly and the observation will be incomplete. However, if the software is intuitive and well-documented, allows the observer to carry out his or her observations effectively, and helps the astronomer analyze that data to reach scientific conclusions, the perception of high quality is likely.

The development of software, although similar to product manufacturing in many ways, is economically unique. As expressed by Campanella 1999:

- ▶ “Software is an intellectual, rather than a physical product, so its development is subject to human and logical constraints, rather than physical laws.
- ▶ One cannot assume that a software specification is stable. Changing requirements is expected behavior in software development.
- ▶ Product defects are a result of human misunderstanding and mistakes, not deficient materials.
- ▶ The economics of software quality hinges on the process of **understanding requirements**; this process... is commonly responsible for the value (quality) of a software product.”

Quality can be defined in many ways, including fitness for use, zero defects, and conformance to requirements. Despite the range of definitions, the goals underlying the pursuit of quality are the same: reducing variation, eliminating waste and rework, preventing human error, preventing defects, improving productivity, and increasing efficiency and effectiveness (Okes & Westcott 2003). The ISO 8402 definition of quality is particularly applicable to telescope operations, however, establishing that quality is “the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs.”

In 2005, the NRAO Program Management Office (PMO) initiated a project to convert its manual timekeeping process to a paperless system which could also be used as a project management resource. This system, currently in development, will enable employees to record their time worked against a charge number which describes the nature of the task. Starting January 1, 2006, members of the Software Development Division (SDD) at NRAO in Green Bank have recorded their billed work time to the newly proposed charge numbers. This data is being used to examine the differences between maintenance and project work, and to examine work patterns by development cycle, product, product category, quality cost category, and other factors. The goal is to institute a continuous improvement practice which yields a distinct and measurable benefit upon the elusive *quality perceived* by outside observers using the telescope.

2. BACKGROUND

The “goal of any quality cost system... is to facilitate quality improvement efforts that will lead to operating cost reduction opportunities.” (Campanella 1999). To do this, it is critical to understand the relationship between cost of quality and total development costs, and to measure the costs of *conformance* (developing and delivering a product that meets the stated and implied needs of its users) and *nonconformance* (failure to provide these stated and implied needs). These components of total development costs are graphically illustrated in Figure 1; components of the cost of quality are further described in Figure 2 below.

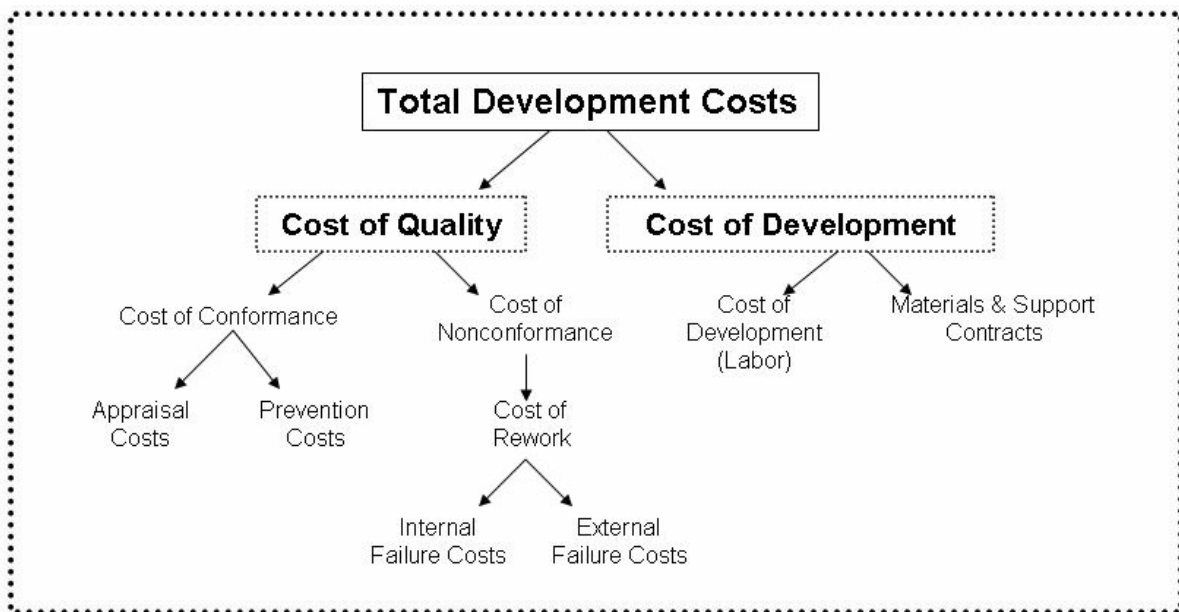


Figure 1. Relationships between terms in a software-specific CoQ model.

An accounting system must be structured so that the costs at lowest levels of this tree are captured: appraisal costs, prevention costs, internal and external failure costs, the cost of development in labor, and costs of materials and support contracts. All of these categories, except materials and support, are dominated by labor costs because of the unique economic aspects of software development as previously described. Two key performance indicators (KPIs) are total quality costs and the cost of rework, which are measured as follows:

- ▶ Total **Quality Costs** = Cost of **Prevention** + Cost of **Appraisal** + Cost of Internal and External **Failures**
- ▶ Because failures incur rework, the total Cost of **Rework** = Cost of Internal and External **Failures**

<p>Prevention Any up-front activities conducted to ensure that requirements are defined and understood, the development process is undertaken with functional and nonfunctional requirements in mind, and all stakeholders are trained appropriately. <i>Includes problems discovered by software engineers before they are exposed as internal or external failures; this includes preventive maintenance.</i></p>	<ul style="list-style-type: none"> • Requirements analysis • Requirements elucidation and clarification • Accurate internal (developer) documentation • Accurate documentation for support staff • Early prototyping • Acquisition of quality data & metrics • Participating in project/task meetings
<p>Appraisal Activities performed to ensure that the product is suitable for intended use</p>	<ul style="list-style-type: none"> • Design reviews • Code inspection • Developing test cases • Integration testing • Regression testing • Pre-release testing by support staff
<p>Internal Failure Discovered <i>before</i> the problem impacts visiting observers (e.g. in commissioning, testing, program checkouts)</p>	<ul style="list-style-type: none"> • Investigating complaints by support staff and other engineers • Notifications to support staff of updated product • Administration associated with retesting
<p>External Failure Discovered <i>by an observer</i> executing a proposed and approved science program (regardless of whether the observer is a staff employee or not); can result in lost customer goodwill</p>	<ul style="list-style-type: none"> • Investigating user complaints • User support calls and emails • Coding/testing/deployment and training associated with interim bug fix releases (patches) • Notifications to customers of updated product • Concessions (e.g. retroactive award of observing time to compensate for systems problems)

Figure 2. Quality cost categories for software process improvement efforts in a telescope operations context.

Typical quality cost programs consider internal failures to be those failures that occur before a software product is *shipped* to the consumer. In the case of telescope operations, the software is continually being used and tested even after it is deployed on the telescope. For example, if new instrumentation is being developed, it is likely that tests will be required using the existing telescope and existing software. Failures uncovered in the telescope software during these times are still failures, but they do not impact the end users when they are readily addressed.

Measurement basis is also significant. For software development in maintenance phases, an appropriate basis is total development costs (the sum of quality costs, cost of development in labor, and cost of materials and support contracts) or total time worked (because software development is dominated by labor costs). For a particular project which includes software, hardware, and other support, a more appropriate basis might be total project costs. Because project cost data is not currently available, total development costs were consistently used as a measurement basis throughout this pilot study.

3. CASE STUDIES

CoQ concepts were introduced by Crosby (1979), and refined by Juran and Gryna (1988) to help managers gain support for quality improvement initiatives from senior management. There were initially two criticisms of this approach: first, monitoring quality costs did not provide insight into the roles of direct or indirect overhead costs. Additionally, it was difficult to determine the root causes of resource consumption. As a result, Leung and Tummala (1999) proposed a 12-step process to implement an *activity-based* cost of quality program, based on “measuring the total cost of each significant activity performed and identifying the cost driver of each activity.” Because the cost driver for all software activities is labor, an activity-based model as illustrated by Tummala et al. (2002) is suitable.

From 1987 through 1996, the Raytheon Electronic Systems (RES) organization used Cost of Software Quality (CoSQ) to document the results of a multi-year process improvement program. This program, based on the Capability Maturity Model (CMM) from the Carnegie Mellon Software Engineering Institute (SEI), aimed to raise the maturity level of the RES software development organization from a 1 (initial, where software processes are ad-hoc) to a 3 (defined, where software processes are standardized and integrated into an organization's operations).

CoSQ is conceptually identical to CoQ, except the term specifically refers to the model as applied to software development organizations; in empirical studies, CoSQ consistently appears to be twice the magnitude of CoQ measured in a traditional manufacturing environment. Knox (1993) investigated the relationship between software quality costs and SEI CMM maturity level, and determined a CoSQ baseline of approximately 50% of development costs for an organization operating at CMM Level 3. An undated Price Waterhouse study confirmed these findings based on a sample of 19 United Kingdom (UK) software suppliers, and also showed that the ratio of conformance to nonconformance costs was about 1.5 to 2.

The findings of the Knox study and resulting model can be summarized as follows:

- ▶ For SEI CMM Level 1, the ad-hoc software development organization, total quality costs are approximately 60% of the total development costs. This falls to 50% by achievement of Level 3 and 20% at Level 5.
- ▶ External failures decrease rapidly, from 35% of total development costs to 10% between Levels 1 and 3, and to near 0% by Level 5.
- ▶ Prevention costs can be expected to increase as a software development organization matures.
- ▶ The costs due to internal failures increase from Level 1 to Level 3, but then decrease rapidly.

Note that while SEI CMM Level 1 represents an organization where software development is ad hoc, Level 3 typifies the organization where a process is in place (requirements gathered, validated, developed, tested, released). As a result, software development in a Level 3 organization is routine, whereas it is chaotic and crisis-driven in a Level 1.

The RES data provided empirical support for the Knox model. In 1988, when the continuous improvement initiative was launched, total cost of software quality was approximately 60% of development costs at an estimated SEI CMM Level 1. This fell to 40% by 1991, when RES reached CMM Level 3, and dropped below 20% by 1996 (at which point CMM Level was not estimated). Cost of rework fell from 40% to 15% to 5% at these same markers. The cost of conformance (prevention plus appraisal) started out around 20%, grew to nearly 30%, and then fell back to approximately 15% by 1996, the end of the study.

4. ACCOUNT STRUCTURE AND QUALITY COST CATEGORIES

In the NRAO timekeeping system, each task an employee performs is billed to a charge number, made up of a 5-digit business unit and a 4-digit subsidiary (or task identifier):

- the first five digits describe *what function the employee was performing*, the code is comprised of a project class identifier and a quality cost category identifier, and
- the last four describe *the task or functional area* to which the work corresponds.

A pay code is also used in the electronic timekeeping system which describes the nature of the productive time (e.g. regular, administrative, worked holidays) or unproductive time (e.g. holidays, vacation, sick leave). For the purposes of this quality cost study, unproductive time is not tracked, and metrics are only generated to understand how regular and administrative time is spent.

Function is determined by **product class** and **quality cost category**. For accounting purposes, GB projects are classified in one of three ways:

- Facilities & Infrastructure (includes all operational support & maintenance) – this is the **43600** series
- Development Projects (includes all development) – this is the **44600** series
- University-Built Instrumentation Projects (CCB, Penn Array, Zspectrometer) – this is the **45600** series

As a result, the first five digits of the code to bill to are determined as shown in Figure 3:

Project Class	Quality Cost Category	Code
Facilities & Infrastructure (43600)	Analysis/Cost Prevention	43621
	Development (<1 Cycle)	43622
	Development (>1 Cycle)	43623
	Appraisal	43624
	Internal Failure	43625
	External Failure	43626
Development Projects (44600)	Analysis/Cost Prevention	44621
	Development (<1 Cycle)	44622
	Development (>1 Cycle)	44623
	Appraisal	44624
	Internal Failure	44625
	External Failure	44626
University-Built Instrumentation (45600)	Analysis/Cost Prevention	45621
	Development (<1 Cycle)	45622
	Development (>1 Cycle)	45623
	Appraisal	45624
	Internal Failure	45625
	External Failure	45626

Figure 3: The first five digits describe the function being performed and the quality cost category.

The final four digits (the “subsidiary code”) describe the actual task worked by the employee. These have been developed so that work can be tracked by product category or subproduct (see Figure 4), software process and infrastructure, or management and administrative tasks. A product-based organization was chosen over a project-based one because products will remain virtually invariant while projects can be expected to change. The full list of task codes is shown in the Appendix.

Product	Subproduct
Control System (5500 series)	M&C/Ygor, Configuration, Grail
Observing Systems (5600 series)	Astrid, Turtle, GFM, Gbtstatus
Data Processing (5700 series)	Data Capture/SDFITS, GBTIDL, CASA
End to End (5900 series)	Proposal Submission, Archive, Pipeline, VO

Figure 4: Listing of products and subproducts.

There are two challenges associated with selecting a task, which were described to the staff as follows.

- **Specific or General?** Was the task specific to a particular part of the product (such as a subproduct or device in the control system), or did the task deal with the product in general? For example, any work spent on configuration improvements should be billed to task codes in the 5520’s, because configuration is a specific aspect of the control systems.
- **Software List or Hardware List?** If you pick from the hardware list, we assume you were working on control system infrastructure for that piece of hardware. So if you can localize your work to a specific device, choose

from the hardware list, unless you are doing software specific work like writing unit tests. Work on the Penn Array would be task 1120, but working on control system libraries for the Penn Array Manager would be 5510.

5. METRICS

In order for data collected by a timekeeping system to be useful, there must be a process and tools for extracting management metrics from the reporting system. In our case, we wish to view hours:

- By quality cost category (prevention, appraisal, internal failure, external failure)
- By cost of rework (internal failure + external failure) as a proportion of total development costs
- By product category (control systems, observing systems, data processing)
- By time spent on process/infrastructure and management tasks
- By time spent on testing and troubleshooting versus product development

Most significantly, we aim to determine for an *arbitrary period of time* (week, month, development cycle, quarter, year) the *total quality costs* are and how they relate to the total *labor costs of development*, and the total *costs of development*. The value of these breakdowns is not so much in the values themselves, but how the values change over time and how they change in response to changes in characteristics of our workloads. The key questions to be answered include:

- **What is the cost of quality as a proportion of total development costs?** According to the Knox model, this suggests the maturity of the software development process in addition to providing a global metric for software and systems quality improvement efforts. A reduction in total quality costs should correlate with long term (e.g. annual) reductions in post-delivery defect reports.
- **How much of our time do we spend on maintenance work versus new development?** This is critical to determine how to budget operations, especially since maintenance work depends on the volume of internal and external failures which can vary substantially.
- **How good is our software quality?** The ultimate measure of quality is not in defect density or response to bug reports, but in how many external failures occur that require rework; internal failures, which do not incur loss of goodwill with external customers, are preferred if failures are to occur. However, internal failures are significant as well; by comparing defect arrival times and outstanding defects with the cost of rework, we will be able to determine the magnitude of the correlation between timely response to bugs and quality costs.
- **Is our development cycle length appropriate?** This can be determined by looking at the pattern of time spent on rework. If there is a significant front-loading of time spent on rework at the beginning of a development cycle, this is clear indication that we have released functionality which was not yet ready for the user and we might wish to consider lengthening our development cycle. This indicator cannot be considered alone, however; we must also account for the typical length of time required to complete a new requirement or function point, and the significance and frequency of repairs needed to production software mid-cycle.
- **Are we spending too much or too little time on testing?** If too little time is spent on testing, the cost of rework will be high, post-cycle. Too much testing time is indicated by an excessively low cost of rework coupled with a high cost of conformance. Knox's model may be used to estimate what percentages are within range, too high, or too low.
- **Are certain observing modes more reliable than others?** Unfortunately, our pilot exercised determined that given our current accounting structure, it is not possible to obtain the answer to this question just by reviewing quality cost data. Continuing investigations are underway.

The management metrics portion of the NRAO-wide system is under development in 2006, but in the meantime Excel is being used as a metrics exploration environment. For more sophisticated analyses, we also need a generalized way to

sum times over a range of task codes (e.g. 5500 through 5599) or an enumeration of task codes (e.g. 0300, 0308, 0302 and 6020). This will, for example, allow us to evaluate the quality cost structure for a particular project over time, and assess whether the management approach for the project is significantly stronger or weaker than others. If the project is stronger, we can evaluate its practices to determine how to improve other projects. If the project is weaker, we should identify bottlenecks and remove them. Ideally, we would also have a way to persist key values for further analysis.

6. RESULTS

Note that from the RES study, the effectiveness of continuous improvement initiatives could only be detected over a period of years. However, marking the results achieved by these initiatives requires an understanding of the current situation and of quality goals. Our long term quality goals are to reduce the cost of external failures (even if this means temporarily increasing the cost of internal failures), and ultimately reducing the total cost of rework as a proportion of total development costs. Time data was collected for the 10-member GBT Software Development Division (SDD) during the first two development cycles of 2006. Cycle 1 (C1) spanned from January 1 to February 15th, while Cycle 2 (C2) started February 16th and continued through the end of the first quarter on March 31st. All of the results in this section are presented based on labor costs because they dominate the total development costs. A quality cost breakdown for C1 appears in Figure 5.

Note that there is a substantial drop in total quality costs at Week 6. The development group was able to determine that one specific bug fix led to a substantial improvement in both the software systems and the efficiency of the group by correlating a drastic reduction in systems downtime with the quality cost data shown below.

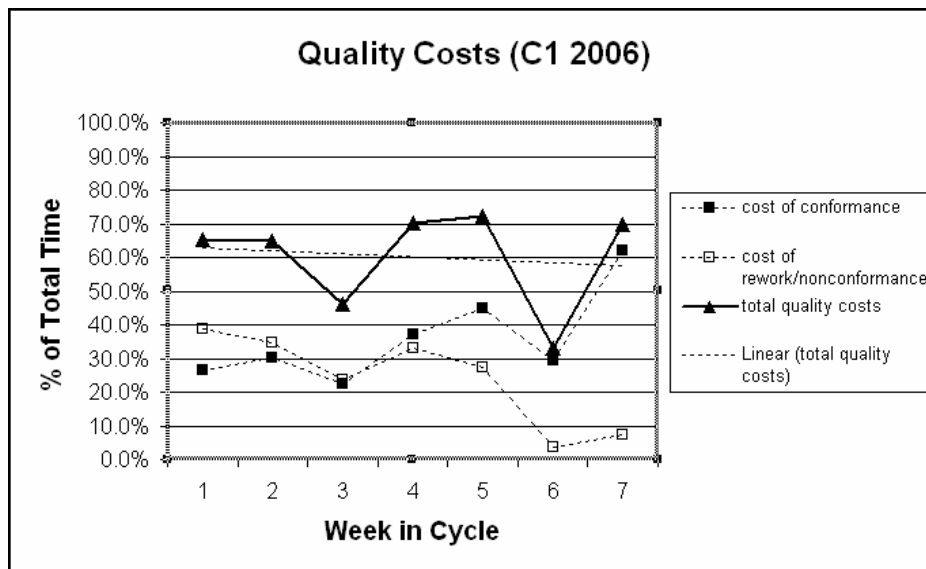


Figure 5: Quality cost data for C1 2006.

Both figures 6 and 7 display alternative breakdowns of metrics from the timekeeping data collection process. Note that substantial effort was spent of infrastructure during this development cycle. When such data is collected over a complete period, such as a year, an effective baseline may be generated to understand the level of resources that must be maintained for systemic, major improvements to software systems. Information such as that in Figure 7 can be used to determine whether a disproportionate amount of time is spent on testing, which may indicate that additional requirements validation must take place prior to development.

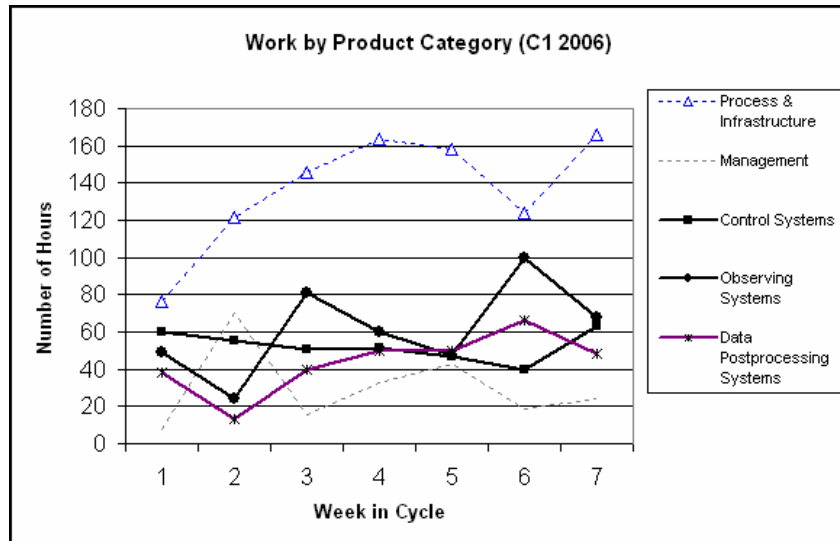


Figure 6: An alternative view of expended effort based on systems in focus.

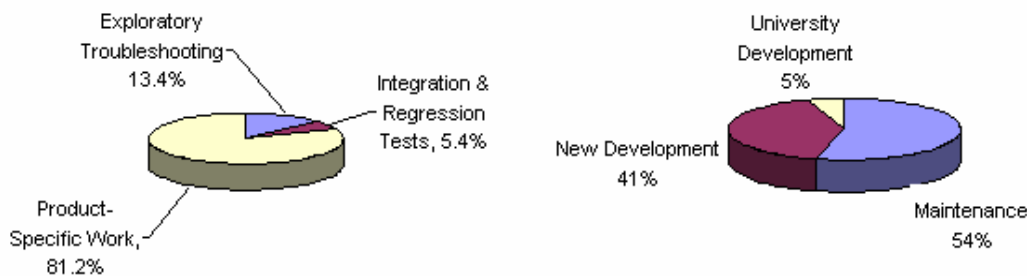


Figure 7: An alternative view of expended effort based on a regrouping of timekeeping data.

7. CONCLUSIONS AND FUTURE WORK

The exploration of a quality cost system and generation of requirements for NRAO's Program Management Office (PMO) for the employee timekeeping system helped to generate many conclusions:

Philosophical:

- Software CoQ reflects the combined efforts of the people who specify requirements, explain requirements, develop conforming software, and test and train support staff and users.

Specific to the software engineering process:

- Defects may be discovered in software by engineers and developers before they actually have an impact on the user. Working to minimize defects is fine, but the real measure of success is playing the timing game: making sure the defects are resolved before they are encountered by external users. These are charged to the "analysis/prevention" category because they represent problems detected before failures have been incurred.
- When "test-first development" characterizes a software engineering process, the time spent on unit test development and functionality development should both be classified as development costs and not as appraisal (a quality cost).

Conclusions for business, project management and accounting:

- Because software development costs are dominated by labor costs, total quality costs can be estimated using time reports
- An accounting system must be *designed* to identify when quality costs are incurred
- No value was gained by artificially separating out “small” development tasks that lasted less than 6 weeks from “large” development tasks that lasted longer than 6 weeks
- Quality costs as a percentage of total development costs is a useful metric for evaluating whether continuous improvement efforts are yielding bottom-line benefits

We also aim to be able to classify quality costs by project to identify aberrations in project management practices. For example, if there is one project for which internal and external failures are high, but prevention and appraisal are low compared to the organizational baseline, this indicates that the project may be more chaotic than other work going on in the organization, and the project manager should be instructed to invest more effort in up-front analysis before development begins.

8. ACKNOWLEDGEMENTS

The National Radio Astronomy Observatory is a facility of the National Science Foundation, operated under cooperative agreement by Associated Universities, Inc.

REFERENCES

- Campanella, J., 2003: Principles of Quality Costs. ASQ Press, Milwaukee, WI.
- Juran, J. and F. Gryna, 1988: Juran's Quality Control Handbook, 4th Ed. McGraw-Hill, Inc.: New York.
- Kaner, C., 1996: Quality Cost Analysis: Benefits and Risks. *Software QA, Volume 3 Issue 1*, 1996, p. 23.
- Krasner, H. 1997: Accumulating the Body of Evidence for the Payoff of Software Process Improvement". Retrieved on February 16, 2006 from <http://www.utexas.edu/coe/sqi/archive>.
- Krasner, H. 1998: Using the Cost of Quality Approach for Software. *CrossTalk: The Journal of Defense Software Engineering*. Vol. 11 No. 11, November 1998.
- Krasner, H. 1999: Exploring the Principles that Underlie Software Quality Costs. *Annual Quality Congress*, 53(0), Anaheim CA (May 1999), pp. 500-503.
- Okes, D. and Westcott, R., 2003: Certified Quality Management Handbook, 3rd Ed. ASQ Press: Milwaukee, WI.
- Tummala, R. V. and Leung, Y. H., 1999: Applying a Risk Management Process to Manage Cost Risk for an EHV Transmission Line Project. *International Journal of Project Management*
- Tummala, R. V., 2002: An Activity-Based Costing Model to Reduce COPQ. *Quality Management Journal* 9(3), 2002, pp. 32-47.

APPENDIX: SUBSIDIARY CODES FOR TASKS

GBT Software	5000 through 5999
Cross-Disciplinary Work (involves combination of astronomy, sw, hw)	5000
Cross-Product Work (involves many software products and includes troubleshooting when cause of problem is not yet known)	5001
Integration Tests	5002
Regression Tests	5003
Quality & Continuous Improvement	5010
Techniques development	5011
Quality systems software	5012
Improvement initiatives	5013
Infrastructure	5020
Physical (machines, platforms, compilers)	5021
Tools (3 rd party software, languages)	5022
Information (supporting databases)	5023
General Management & Administration	5100 through 5199
Reports, Presentations, Committees, Executive Briefings	5101
Performance Evaluation & Goal Setting	5102
Attending Conferences	5103
Attending Classes (ex. IDL class)	5104
Observatory-Wide Software Projects	5200 through 5299
General work with ALMA	5201
General work with EVLA	5202
Control Systems	5500 through 5599
Product Management	5501
Release Management	5502
M&C General (libraries, utilities)	5510
M&C Specific	<i>See 1000 - 4999</i>
Configuration	5520
Transaction Management (Grail)	5530
Observing Systems	5600 through 5699
Product Management	5601
Release Management	5602
Astrid	5610
Observation Management (Turtle)	5620
Quick Look (GFM)	5630
Gbtstatus	5640
Data Post-Processing	5700 through 5799
Product Management	5701
Release Management	5702
GBTIDL	5710
CASA	5720
Data Capture/SDFITS	5730
End to End	5900 through 5999
Product Management	5901
Release Management	5902
Proposal Submission Tool	5910
NRAO Archive	5920
Virtual Observatory	5930
Pipeline	5940

GBT Systems Tasks

GBT Hardware/Electronics	1000 through 4999
Frontends	1000
PF1	1010
PF2	1020
L-Band	1030
S-Band	1040
C-Band	1050
X-Band	1060
Ku-Band	1070
K-Band	1080
Ka-Band	1090
Q-Band	1100
W-Band	1110
Penn Array	1120
IF System	2000
LO1	2010
LO2/LO3	2020
LO Distribution	2030
IF Rack	2040
Converter Rack	2050
Analog Filter Racks	2060
Backends	3000
Spectrometer	3010
Spectral Processor	3020
DCR	3030
Test Spectrometer	3040
CCB	3050
Zspectrometer	3060
BCPM	3070
Antenna-Related	4000
Servo Systems	4010
Active Surface	4020
Laser Rangefinders	4030
Site Time Equipment	4040
Weather Stations	4050

Miscellaneous

Safety Meetings/General Facility Related Work	0300
Science Center (not related to exhibits)	0302
Rec Board	0308
Science Center (exhibits)	6020