CALIFORNIA STATE UNIVERSITY
FULLERTON™

COLLEGE OF ENGINEERING
AND COMPUTER SCIENCE

Advanced Software Process

Part IV: The Managed Process

16. Managing Software Quality

Dave Garcia-Gomez
Faculty / Lecturer
Department of Computer Science

Course Roadmap

Part I: Software Process Maturity
- 1. A Software Maturity Framework
- 2. The Principles of Software Process Change
- 3. Software Process Assessment
- 4. The Initial Process

Part II: The Repeatable Process
- 5. Managing Software Organizations
- 6. The Project Plan
- 7. Software Configuration Management (Part I)
- 8. Software Quality Assurance

Part III: Defined Process
- 9. Software Standards
- 10. Software Inspections
- 11. Software Testing
- 12. Software Configuration Management (continued)
- 13. Defining the Software Process
- 14. The Software Engineering Process Group

Part IV: The Managed Process
- 15. Data Gathering and Analysis
- 16. Managing Software Quality

Part V: The Optimizing Process
- 17. Defect Prevention
- 18. Automating the Software Process
- *19. Contracting for Software*
- *20. Conclusion*

Managing Software Quality

- The Quality Management Paradigm
- Quality Examples
- Quality Motivation
- Measurement Criteria
- Establishing a SW Quality Program
- Estimating SW Quality
- Removal Efficiency
- Quality Goals
- Quality Plans
- Tracking and Controlling SW Quality

Managing Software Quality

- One of the best ways to evaluate a SW organization is to examine the quality of its products.
- Products quality is thus a key measure of the software process.
- Products quality provides a clear record of development progress, a basis for setting objectives, and a framework for current action.

Managing Software Quality

- As we strive to improve the quality of our software products, it is instructive to examine the experiences in other fields.
- Agresti (1981) has pointed out that the principles of industrial engineering can be effectively applied to software.
  - Quality measurement
  - The impact of such measurements on the performance of the people who do the work

Managing Software Quality

- In any human-intensive process, motivation is the key to good work.

- To consistently achieve superior performance, management must establish challenging quality goals and strive to meet them.

Managing Software Quality

- Four basic quality principles:
  - Unless you establish aggressive quality goals, nothing will change.
  - If these goals are not numerical, the quality program will remain just talk.
  - Without quality plans, only you are committed to quality.
  - Quality plans are just paper unless you track and review them.

The Quality Management Paradigm

- The basic principles of SW quality management are much like those for cost management.
  - You set goals, make plans, track performance, and then adjust the plan.
  - This cycle is the foundation for industrial engineering, and it applies equally well to SW.

Quality Examples

- Quality programs really work.
- Poston and Bruen (1987) describes the Leeds and Northrop experience with a real-time monitoring program of 27,719 lines of source code.
  - They reported that, by rigorously following a quality plan, failure density rates were reduced a lot, and they found an enormous productivity improvement as well.
  - They found that the quality and productivity improvements resulted from finding and fixing essentially all the errors early in the program.

Quality Motivation

- The purpose of a quality program is to motivate action.
- To ensure that this action results in better products, the measurements must reasonably represent goodness in customer terms.
- When considered as a motivational program, quality measurements take on special meaning.

Measuring Quality

- A different approach is needed to motivate an aggressive quality effort.
- Since people can only respond to a few motivational drives at a time, the measurement and tracking for the quality program must be tightly focused.
- The need is to establish a small number of specific, numerical quality measures.
  - Typical examples are defects found at various stages of the process, inspection efficiency, or test coverage.

Classes of Quality Measures

- Development
  - Defects found, change activity
- Product
  - Defects found, SW structure, information (doc) structure, controlled tests
- Acceptance
  - Problems, effort to install, effort to use
- Usage
  - Problems, availability, effort to install, effort to use, user opinions
- Repair
  - Defects, resources expended

Classes of Quality Measures

- Table 16.1 Classes of Quality Measures
  - The quality measures are characterized by:
    - Objective
    - Timely
    - Available
    - Representative
    - Controllable

Measurement Criteria

- In deciding what measures to use it is essential to consider the objectives of the measurement program.
- If the measures will be used to manage software development, they should be objective, timely, available, and controllable.

Measurement Criteria

- On the other hand, if the quality measures are to support decisions on product acceptance, they must reasonably represent user needs and be available before the acceptance decision.
- While defect measures may be of some value, better results can generally be obtained with controlled tests.

Defect Measures

- Defect counts only moderately represent the customer's view of product quality.
- While it is generally true that programs with large numbers of bugs have poor customer satisfaction, once a base level of quality is reached, defect measures no longer predict customer satisfaction.

Defect Measures

- Table 16.2 User Quality Rankings
- Beyond a certain point, problems and defect activity is less important than other factors such as usability, performance, and functionality.
- Clearly, determining what truly represents quality in the customer's eyes is not a simple task.

Change Activities

- Change activity can be a useful measure of development quality.
- When change activity remain high late in a development program, it is a good indication of overall quality problems.
- Unfortunately, this measure is generally not directly controllable by the development group but is a consequence of such factors as defect rates or requirements stability.

Error Seeding

- Error seeding is a potentially interesting way to evaluate program quality.
- The idea is to inject a known number of dummy defects into the program and then to track how many of them are found by the various tests or inspections.
- While there is increasing evidence that the seeding idea is promising, it must still be viewed as an experimental technique.

Controlled Tests

- With controlled tests the product is subjected to a simulated work environment, and an evaluation is made of its operational suitability.
- For example, human factors tests are typically used to determine the adequacy of user interfaces and documentation; a special test environment is established with the documentation and system facilities required to demonstrate specific aspects of operational behavior.

Problems Measures

- Problem activity data provides a potentially useful measure of program quality.
- After program installation and cut-over, records can be kept of user-reported problems.

Problems Measures

- The remainder generally concern user errors, HW problems, duplicate defects, suggested improvements, and unknown causes.
- For large system with many users the number of problems due to software defects is typically less than 3 to 5 percent of this total.

Installation and Operational Effort

- While the effort required to install and operate the system is a good indicator of customer satisfaction, such measures are again not available until after completion and customer delivery.
  - They also require the customer to do a lot of work.
  - In this case, however, the factors that make installation and operation expensive can be usefully examined early in program development.
    - Define the step-by-step procedures for installing and operating the system and then actually measure the time and effort

Customer Satisfaction Surveys

- To understand the customer's view of their products, some organizations use customer satisfaction surveys.
- IBM, for example, periodically conducts mail or telephone surveys to determine their customer's views of certain key products and support services.
- They include in the questions a request for suggestions on where the products or services could be improved.

Customer Satisfaction Surveys

- Such surveys, if properly conducted, can provide an informed view of product quality.
- Unfortunately, this measure again is not available until substantially after development completion.

Reliability and Availability

- Availability
  - For some programs the customer is most concerned with the system's ability to perform the intended function whenever needed.
  - Such measures are particularly important for system and communication programs that are fundamental to overall system operation.
  - Control program, DB manager, input-output system

Reliability and Availability

- $\text{Availability} = \left(1 - \frac{\text{MTTR}}{\text{MTTR} + \text{MTBF}}\right) \times 100$

- MTTR
  - The mean time required to repair and restore the system to full operation
- MTBF
  - The mean time between failures

Selecting Quality Measures

- While the most appropriate quality measures depend on their intended use, they also must depend on the kinds of data available.
- Since defect data is all that most software development organizations can obtain before system shipment, it should be used.
- While this properly emphasizes the need to reduce the number of delivered defects, it does not consider many other important quality attributes.

Selecting Quality Measures

- When no quantitative measures are available for such things as functionality, performance, and human factors, other steps must be taken to ensure that these important topics are given adequate attention.
  - Some practical steps are to include selected customer representatives in specification reviews, conduct laboratory usability tests, do benchmark testing of customer programs, and conduct customer surveys.
  - It is worthwhile to do some early testing in either a simulated or real operational environment prior to final delivery.

Establishing a Software Quality Program

- The software quality principles stated at the beginning of this chapter can be expanded into the following steps:
  - Senior management establishes aggressive and explicit numerical quality goals.
  - The quality measures used are objective, requiring a minimum of human judgment.
  - These measures are precisely defined and documented so computer programs can be written to gather and process them.

Establishing a Software Quality Program

- A quality plan is produced at the beginning of each project.
- These plans are reviewed for compliance with management quality goals.
- Quality performance is tracked and publicized.
- Since no single measures can adequately represent a complex product, the quality measures are treated as indicators of overall performance.

Development Plan Quality Measures

- When the objectives is to motivate superior development or maintenance performance, defects are the most practical quality measure.
- This is because there is little other data that can reasonably be gathered and used during development.
- Further, if defects are not measured, it is hard for the SW professionals to take any other measures very seriously.

Portraying Software Quality During Acceptance Test and Use

- For tracking customer-found defects, Inglis (1986) suggests a set of cumulative defect curves, starting with system test.
- These curves are normalized per 1000 lines of code and plotted (as shown in Figure 16.1).
- Figure 16.1 Plan vs. Actual – Three Product Release
  - Here, the cumulative number of defects are plotted each month as received.

Portraying Software Quality During Acceptance Test and Use

- Figure 16.2 Defect Rates – Release 2
  - Here, the defects found each month are shown for the release 2 plan and for actual experience.
  - When normalized by program size, such historical data can provide a helpful reference in producing new quality estimates.

Estimating Software Quality

- In making a software quality estimate it is important to remember that every project is different.
- While all estimates should be based on historical experience, good estimating also requires an intuitive understanding of the special characteristics of the product involved.

Making a Software Quality Estimate

- An estimate of program quality is made as follows:
  - Recent programs completed by the organization are reviewed to identify the ones most similar to the proposal product.
  - Available quality data on these programs is examined to establish a basis for the quality estimate.
  - The significant product and process differences are then determined and their potential effects estimated.

Making a Software Quality Estimate

- Based on these historical data and the planned process changes, a projection is made of anticipated quality for the new product development process.
- This projection is then compared with goals and needed process improvements are devised to meet the goals.
- The project quality profile is then examined to determine the areas for potential improvement, and a desired quality profile is produced.
- A development plan is produced that specifies the process to be used to achieve this quality profile.

Software Quality Models

- To make an accurate quality estimate it is essential to have a quality model.
- While this need not be an explicit mathematical model, it should identify the basic assumptions behind the estimates.
- The reason most mathematical software models are not helpful for estimation purpose is that they make some highly restrictive assumptions about the SW process.

Software Quality Models

- Assumptions:
  - The failures are independent.
  - The number of failures is constant.
  - Each failure is repaired before testing continues.
  - All failure are observed.
  - Testing is of uniform intensity and representative of the operational environment.
  - The failure rate at any time is proportional to the current number of remaining defects in the program.

Software Quality Models

- The times between failures are independent.
- The different types of errors are of equal importance.
- Each error exhibits the same failure rate.
- No errors will be introduced during the testing and repair process.
- The failure rate will decline during debugging and operation.

Intuitive Quality Models

- Unique, non-uniform characteristics of quality models that make the SE process tractable:
  - Program module quality will vary, with a relatively few modules containing the bulk of the errors.
  - The remaining modules will likely contain a few randomly distributed defects that must be individually found and removed.
  - The distribution of defect types will also be highly skewed, with a relatively few types covering a large proportion of the defects.
  - Since programming changes are highly error-prone, all changes should be viewed as potential sources of defect injection.

Intuitive Quality Models

- Figure 16.3 Pareto Analysis of SW Defects
  - The Pareto Principle: 80/20 Rule
    - A majority of the problems (about 80%) are from a few key causes (about 20%).
  - Over one third of the defects were in the three highest categories.
- Table 16.3 Categories of SW Defects
  - By focusing their process improvement efforts on the most prevalent defect categories, Hewlett- Packard was able to achieve significant quality improvement.

The Quality Estimate

- In comparing the components and modules of the new product with similar elements of prior products, each program's unique design and implementation characteristics must be considered.
- As a consequence the quality profile for each program will also probably be different.
- Table 16.4 Development Defect Profile
  - Inspection efficiency

Removal Efficiency

- One indication of the overall efficiency of the development process is the cumulative removal efficiency, which indicates the cumulative percent of the previously injected errors that have been removed by the end of each project phase.
- Since defect removal costs can be expected to roughly double with each project phase, attention should be focused on early removal.

Removal Efficiency

- Table 16.5 Planned Injection Rates and Removal Efficiencies
- Table 16.6 Planned Defect Profile
- Comparison:
  - Increasing inspection efficiency – by improving the effectiveness of code inspections and unit test.
  - A 20% improvement in inspection efficiency (from 63.8% - Table 16.4 to 76.3% - Table 16.6) has resulted in a three times improvement in the resulting shipped product quality (91% to 97% - Table 16.5).
  - Inspection efficiency is a very sensitive indicator of shipped program quality.

Quality Goals

- The quality goals of every development, manufacturing, or service organization must be established by senior management.
- The senior managers are the only ones who can set such standards and they are the ones who best appreciate the full benefits of an aggressive quality program.
- This means that senior management must establish quality goals and clearly state them to their people.

Quality Goals

- Establishing quality goals:
  - Every new product or product release must have better quality than its predecessor.
  - The corporate quality organization, with the assistance of the SW groups, will establish the quality measures to be used.
  - Each product manager is responsible for producing a documented quality plan to meet these goals.
  - Product quality performance will be judged by:
    - The degree to which quality plans show improvement
    - The effectiveness of the action plans to address areas of deficient performance
    - Product management's willingness to establish more aggressive goals when performance exceeds plan
  - The quality organization will periodically report to senior management on product and organizational performance against these goals.

Quality Plans

- The quality plan documents the quality actions management intends to implement.
- It includes the derivation of the quality measures, the identification of the planned process changes, and the anticipated quality improvements.

Quality Plan Outline

- Table 16.7 Quality Plan Outline
  - Product introduction
  - Product plans
  - Process description
  - Quality plan
  - Risks and exposures

Tracking and Controlling Software Quality

- The critical elements of a software quality management system are:
  - A responsible authority is named to own the quality data and the tracking and reporting system.
  - Quality performance is tracked and reported to this authority, during both development and maintenance.
  - Resources are established for validating the reported data and retaining it in the process DB.

Tracking and Controlling Software Quality

- Actual product and organizational performance data is periodically reviewed against plan.
  - Results are initially reviewed with responsible line management and any discrepancies resolved.
  - Performance against targets is determined, and, if worse than plan, line management prepares an action plan for review with higher management.
  - If actual performance is substantially better than plan, product management establishes more aggressive targets.
- Quality performance is published, and a highlight report is provided to senior management.
- Overall performance is periodically reviewed with senior management.

References

Humphrey, Watts S., *Managing the Software Process*, The SEI Series in Software Engineering, Addison-Wesley, 1989. (29th Printing, May 2003) (ISBN 0-201-18095-2)