PROCESS IMPROVEMENT FOR SOFTWARE DEVELOPMENT AND MAINTENANCE IN SMALL ORGANIZATIONS

By Lee Bolleter, Texas Children's Hospital, Houston, Texas

Abstract

Process improvement for software development and maintenance has roots in manufacturing, engineering, and science process improvement. Process improvement is a continuous cycle of:

- understanding
- change
- control
- automation.[12]

Increased productivity, predictability, and adaptability are desired benefits of continuous improvement [10]. Another reason for implementing continuous change in the form of process improvement is the generally poor record software development and maintenance has for product quality, product timeliness, or both.[13]

To implement process improvement one takes measurements of the software development and maintenance processes. Measurements of the software process, or metrics, and the subsequent analyses indicate areas to change for improvement. There are many methods of metrics analysis available. These methods depend on the degree of emphasis of each of the multiple goals specified.

The Software Engineering Institute (SEI) along with Watts Humphrey is a forerunner in the field of software process improvement. The SEI method of process improvement uses stepped levels to progress towards process maturity.

This paper focuses on the smaller development and maintenance shop in contrast to the vast majority of material on software process improvement. The last section provides general suggestions for Programming and Development Groups for the implementation of software process improvement. The material discussed is also applicable to almost any information systems related group willing to change and improve.

Description of process improvement

A. The basics

Process improvement in software development is probably one of the more difficult areas for the Quality Improvement discipline because of the newness of the computing science field. For comparison, software development has been around for a few decades and civil engineering has been practiced for over two thousand years. One can model quality improvement benefits from the engineering and manufacturing areas to the software development and maintenance discipline. This will reduce the time and effort required to develop process improvement techniques for software.

Process improvement for software development and maintenance specifies that one must define the process, measure the process, and continuously improve the process in a cyclical manner. A process is the steps one takes to perform a task. Process improvement has been part of the manufacturing and engineering disciplines for ages. Henry Ford and Japanese manufactures achieved great successes with the ideals of defined, monitored, and continuously improved processes. Companies including HP and AT&T migrated process improvement strategies from manufacturing to software production.

Definition of the process is the first step. This is not a trivial step but something that nearly everyone recognizes as a necessity, and a task that should have already been completed. While it may not be feasible to make procedures for everything that is being done in a shop today, it is worth

the effort to attempt a cultural change. A starting point of the present time can be used to require proper documentation on all further activities. Reviews are needed on all new procedures until a general consensus is obtained, then reviews can be decreased. Too much documentation can be an overwhelming burden.[13] The people involved are the experts and will know how much documentation is necessary.[14] Find related examples of high quality concise process definitions and model new documentation from the examples.

B. Knowledge capture

The definition of a process makes the process available for others to learn. Sharing of knowledge is particularly important in the field of software development because of the fast pace and newness of the field. The software development field can be very closed and unsharing. Jargon exists to describe some practitioners and acts, for example, gurus, wizards, and magic. Many reasons keep software professionals from not doing a professional job and not recording process methods. These reasons include unrealistic time constraints, job insecurity, personal insecurity, intolerance of another's experience level or expertise, and so on. An effort must be made to collect and maintain unique project and product knowledge. Knowledge capture will ease the eventuality of project reassignment, personnel turnover, and further work on the project.[6]

C. The type of problem process improvement addresses

Process improvement is not a "silver bullet" solution to all the problems of a programming department. In the paper "No Silver Bullet: Essence and Accidents of Software Engineering," Frederick Brooks Jr. describes two basic types of hurdles facing software engineers. Brooks divides them into:

1. essence "... the difficulties inherent in the nature of software."
2. accidents "... those difficulties that today attend its production but are not inherent.[4]"

Technological solutions such as improved languages and better programming environments address the accident type of difficulty but do not address the difficulties with the essence of software production. An essential quality of software is that with increasing system size there is a greater than linear increase in complexity and required team communication.[4] Process improvement addresses the essence of software production difficulties.

D. Metrics and metrics analysis

Measuring a process is a difficult but essential task. The infamous "lines of code" measurement has not gone away despite notable shortcomings.[11] "Lines of code" measurements are as controversial as complexity metrics. Measurement of engineering time is subjective. A manager will probably receive close to what they want to get whether it is the truth or not.[6] Measurement includes counting:

- defects
- accuracy of estimates
- complexity of code
- comments and documentation
- overtime used
- amount of reused modules
- new modules created that are now available for reuse

The key to process improvement is analysis of the measurements. A typical item stemming from the initial measurement and analysis of many manufacturing processes is that defects need attention as early as feasibly and economically possible. This will save a lot of money because of reduced waste. The same is true in software development. This is "just common sense" but why are there so many errors in such a high percentage of software produced today? Much of the answer lies in poor analysis and design. Many defects in software stem from the continuation of the process of producing software without a proper understanding of the function that is being developed.[2,11,12] "For example, a well designed program to control a missile was designed by someone who understood missiles.[13]" The point here is that metrics analyses enable problem resolution at an earlier and less expensive stage than without metrics analysis.

Reasons for process improvement

A. General reasons for software process improvement

The goal of software process improvement is to boost software engineer's efficiency. There has not been any scientific proof that software process improvement is a method to increase productivity, but employees throughout the software departments in HP, AT&T, and other companies enjoy process improvement. They perceive more control of the software process, better estimation of the software process, and better productivity in the software process.[11,12,16]

Without a doubt most programmers enjoy producing software systems to the point of celebration when a system or piece of a system performs correctly.[3] Equally popular is the notoriety of software production as an art that unfortunately is very difficult to predict.[2] Software measurements require consistent and thoughtful application.[12] Metrics analysis should be done continually through the software cycle. Without metrics and metrics analysis future estimations of software development and maintenance are doomed to the same fate as all the past bad estimates.[13] "When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science.[13]" It is true that with more experience better estimates are possible but:

- without metrics collection (mostly automated) and analysis of the data there is usually not a trail or document for others to learn from
- without measurement and analysis of past projects, future surprises will be more frequent
- new types of projects will be unnecessarily difficult to estimate.[13]

There is the perception in many software shops that it is not worth the effort to define the processes of development and maintenance. Many shops also have the attitude that it is trivial to define the process and that procedures are too much trouble to write. Procedures are a lot of trouble to write and the endless tomes frequently produced are equally worthless. Lean structure for procedures requires flexibility.[13] The review process should hold in check any abuses of flexibility. Efficiency requires flexibility.[13]

Increases in productivity from process improvement come from the continuous analysis and improvement of the process. By tracking defects the root-causes of defects become visible. The cause of a category of defects could be nonintuitive.[8,12] When there are facts and figures to back up the identification of some activity or lack of activity as a problem source the solution has a much greater chance of gaining acceptance.

B. The government's lead in implementing software process improvement

The United States government is a large consumer of software. It has made an early start in specifying the quality of software delivered and the processes used to produce the software they purchase.[17] The Department of Defense (DoD) has three quality improvement standards. DoD standard 2167A outlines how to develop software within the military. The standard is very thorough and includes at least four types of suggested testing methods.[16] The second standard, DoD 2168, is very important in that it requires a vendor to track metrics. The standard on "... defense system software quality program outlines the elements needed in a contractor's software quality program (SQP), including objectives, responsibility, documentation, planning, implementation of SQP, software quality records, software quality evaluation records, software corrective action, certification, management review of SQP, and access to data for review by the contracting agency.[16]" DoD 5000 is the third standard for software. DoD 5000 is Total Quality Management (TQM) - A Guide for Implementation. This guide shows vendors how to implement process improvement and encourages them to do so. The guide also allows for a vendor disqualification if the vendor lacks a quality program or has a record of having a bad product or service. Given that the government is using software process improvement for software developed in-house and outside purchased software it is probable that the private sector will do the same.[1]

How PI can be achieved in software development

A. The focus of this paper

Many books contain information on improving software processes in programming and development departments. The differences are great between a large software department that writes and maintains larger software packages and a department that supports larger software packages with somewhat limited development activity. There is enough of a similarity in the material written about larger development and maintenance sites to use for smaller departments.

B. The ISO 9000 standard

The International Standards Organization (ISO) has developed a set of standards that allow a business to become certified as providing a quality manufactured product or a quality service. The ISO 9000 series of standards help a business define what processes are necessary to offer a quality product. The ISO 9000 standard is very broad and can apply to almost any business.[14]

Certification to the ISO 9000 group of quality standards is a good first step in software process improvement. The ISO 9000 standards basically state that a facility must have procedures, job descriptions, and a chain of command. "ISO 9000 requires that you simply operate in the manner that you say you do while meeting certain basic requirements.[14]" Procedures do not require specific sections or a set format. The ISO 9000 specification has the point of view that the employees of the business are the experts about that business and should know what steps need documentation. It is common sense to have procedures and documentation but having an outside party review this information would probably eliminate many programming sites from passing certification.

Because the first step in process improvement is definition of the process, achieving ISO certification will have positive effects:

- will help the customer by showing that the quality of what is given them is important
- will help employees to be confident and proud of all of their organizations work
- will begin the cycle and focus on quality and quality improvement.[14]

ISO 9000 is becoming a standard to do business in the European community. ISO 9000 has gained wide acceptance with European companies and United States companies that sell to the European market.

A Metrics Analysis focus for Process Improvement

A. HP, AT&T and software metrics

The Hewlett-Packard (HP) solution to achieve process improvement capability does not require the regimented Software Engineering Institute's steps. The two books Software Metrics and Practical Software Metrics for Project Management and process Improvement stress the collection and analysis of metrics. The HP way of attacking process improvement is through the careful gathering and analysis of metrics. The purpose of the two books just listed are to describe the process improvement effort at HP so that the reader can implement a similar but custom tailored approach at their own organization. The HP books describe the successes and mistakes of HP's process improvement efforts over approximately a decade.

Many software process improvement books and articles have very detailed instructions on how to continue with developing process improvement at a large software development facility. The HP references frequently mention multiple divisions of sixty people each. Since the target audience is large development groups there are specific types of meetings preplanned and described in the books.[12] A process improvement effort for a smaller facility does not need such a large effort because the communications problems are not as significant. Similarly the process and quality groups described by Watts Humphrey are not possible in departments with less than twenty people.

The ten steps to a successful startup of process improvement from the HP perspective are:

1. "Define company/project objectives for program
2. Assign responsibility
3. Do research
4. Define initial metrics to collect
5. Sell the initial collection of these metrics
6. Get tools for automatic data collection and analysis
7. Establish a training class in software metrics
8. Publicize success stories and encourage exchange of ideas
9. Create a metrics database
10. Establish a mechanism for changing the standard in an orderly way.[12]"

B. Metrics

With the metrics focus of the HP process improvement effort one must decide what data items need collecting. The metrics must be carefully defined and verified. Failure to do so will invalidate the analysis results because one is in a situation of comparing dissimilar items. If the metrics definitions change then comparison with the previously collected data is not possible. The goal/question/metric paradigm is discussed in the analysis section of this paper. A very good list of goals, questions, and associated metrics are in appendix 2 of this paper. "Without such measures for managing software, it is difficult for any organization to understand whether it is successful, and it is difficult to resist frequent changes of strategy.[11]"

Quantification of Ambiguity is possible. The four types of ambiguity are:

1. problem-statement or analysis - confusion in what the problem is
2. requirements or design - confusion in how the problem will be solved
3. design-process or method of solution - confusion about the particular process to solve the problem
4. final-product - confusion in the solution to the problem.[7]

To measure ambiguity one must poll qualified individuals on different bases of the project. For example, the number of modules a section of a requirements analysis would require or the amount of time and number of people required to code and test a specific function. Compare the results from the poll of the same items for a certainty factor or ambiguity metric.[7]

The accuracy of programmer metrics is important. Brooks, DeMarco, Grady, Humphrey, Yourdon, and Yeh stress that data must be kept confidential and must not be used for personnel review purposes.[5,6,11,12,13,30,31] To counter this problem but still enable collection and dissemination of information:

- data is collected on an individual basis but is confidential
- data is reported in group format
- management is not given access to all metrics
- reports are approved by the group prior to distribution.[11]

Complexity is a very difficult metric. People seem to disagree on how to measure complexity as if it were a matter of pride - which it arguably is. One must categorize the types of complexity to agree on metrics:

- computational complexity addresses the amount of time required to compute an answer
- syntactic complexity is related to the number and type of commands in a program and the number and sophistication of the data structures
- semantic complexity relates to the types of commands issued and the number of possible execution paths.[16]

In this age of yearly doubling or more of computational power for the monetary unit, computational complexity can often times be deferred or given less priority because of a long-term perspective. As long as one is cognizant of how their program works and how it could be more efficient (if needed in the future), then the tradeoff (if present) between maintainability and execution speed should favor maintainability.

Syntactic complexity is fairly easy to measure automatically. Discussions on weights for different commands and the benefits of certain data structures for the target application can be fun. The game type situation of

setting up complexity weights and measures can easily involve all the software engineers at an organization. Semantic measurements show the inherent complexity in the software measured. Examples are McCabe's cyclomatic complexity measure and Halstead's effort measurement.

A prudent time to measure complexity is at the design phase. For a module with a high design complexity either:

1. break up the module or
2. if the module is irreducible the higher complexity will require an exponentially larger amount of resources.[5]

As number two above states the exponential increase in required resources is a stiff penalty from which quantification would be very valuable. Metrics will give a base to compare the benefits of module simplification.

C. Analysis of Metrics

The possible danger with the collection of metrics is that the metric may be selected as the way to solve the problem. Instead careful analyses of the tradeoffs and interrelations of the specific metric with other metrics and organizational goals have to be considered. If a project is late and a supervisor uses the metrics database to discover, in their opinion, too much time being spent on meetings then the supervisor could curtail that activity. What might be happening is that the current project stage requires a high degree of communication and requires the extra meeting time. Communication problems can arise from meeting time being shortened by mandate.

C1. FURPS and customer satisfaction

FURPS and FURPS+ are HP acronyms used to focus on customer satisfaction and product quality. The FURPS+ model allows developers a method of task prioritization. Because of the categorization of FURPS+, actions taken are more easily measured.[11] The components of FURPS+ are:

- Functionality
- Usability
- Reliability
- Performance
- Supportability

A more detailed breakout of FURPS is in Appendix 1 of this paper.[11] Obtaining metrics on FURPS+ progress is important. Quality surveys are one method of getting data and interviews are another method. Suggested guidelines for conducting a survey are in Appendix 2 of this paper.

C2. The 3 HP market focuses

HP uses a measurement and evaluation paradigm invented by Victor Basili of the University of Maryland. The paradigm defines major business strategies for a software organization. Even if an organization is not in the business of selling software the perspective is still useful. The paradigm uses three focuses:

- Minimize engineering effort and schedule
- Maximize customer satisfaction
- Minimize defects.

The chart in appendix 3 presents timing, usefulness, metrics, and drivers for the 3 market focuses. Appendix 4 lists questions and associated metrics for each process focus.

C3. Minimizing engineering effort and schedule

In these times of increased emphasis on efficiency and cost reduction, the focus on minimizing engineering effort and schedule is frequently used. The fine line to balance in minimizing engineering effort and schedule is to keep the goals of cost and schedule high on the software engineers' lists of priorities but not create a situation where compromises in functionality are deemed necessary.[11] Robert Grady suggests that this perspective is particularly useful at the middle part of a project. After system analysis, the coding phase of a project can be streamlined to get the best first draft. Defects are removed from a functionally complete system.[11] This is identical to the method many instructors suggest for writing a paper or letter.

Minimizing engineering effort and schedule does not preclude the investment of time for future payoff in decreased maintenance requirements. A complexity study on a system requiring a high degree of maintenance will highlight the modules most likely causing problems.[11] The higher

complexity modules will quite possibly have higher error rates. Further modifications should stress decreasing the complexity of the code. Modifications to portions of the system that are "more complex" are repaired or replaced with code that is less complex and more easily maintained. This will save time and effort in some situations.

The preceding paragraph illustrates a situation of correcting a design flaw. At HP the analysis of certain metrics have shown "... potential downstream costs of as much as two engineering months for each design defect .... [11]" Pressure for high productivity is an effect of the focus to minimize engineering effort and schedule. "True productivity also includes other aspects such as quality, time to market, and long-term investments such as design for reuse.[11]"

C4. Quality Function Deployment

Quality Function Deployment (QFD) represents a proposed software system from the customer's perspective. A QFD analysis pushes the software engineer into the perspective of the customer. Weighted values show priorities in customer needs. For further examples of QFDs refer to references 11 and 16.

Minimizing defects entails fairly straight forward metrics and analysis. One needs to define and measure defects though different project cycles. Comparison of project metrics will point out weakness' in the process that cause defects. Graphs are very important in defect trend analysis.

C5. AT&T analysis methods

Hsiang-Tao Yeh lists some very illuminating questions and problems that stem from possible "wrong" answers about a shop's condition (appendix 4). The book also suggests certain best current practices (BCPs):

- customer satisfaction surveys
- project estimation tools (metrics)
- written procedures
- rapid prototyping
- build on a reusable platform
- analysis, design, code, and test inspections
- root-cause defect analysis
- post-project reviews.[16]

C6. Goal/question/metric paradigm

Another technique used at HP is the goal/question/metric paradigm. An example of a question line is "How accurate are our estimates?" and "What is the trend?" The question is hopefully answerable through the metrics data. In this case a graph comparing estimates and actual results over time is appropriate. There are many examples of very insightful questions, metrics, and graphs throughout both HP and AT&T books.[11,12,16] See appendix 4 for a list of goals, questions, and metrics. An important point to begin a metrics program is to ask questions about the groups goals and to look for differences and similarities between the examples and lessons pointed out in the HP and AT&T books and the organizations current situation.

The Software Engineering Institute (SEI) Process Improvement Model

A. Introduction to SEI

The SEI model for process improvement requires a lot of work just to implement their brand of process improvement. One must weigh the suggestions and sequences provided by each step of the SEI model with the perceived need of their own organization. Watts Humphrey invented the five ascending SEI process levels. The SEI process levels are:

- level 1 - Initial
- level 2 - Repeatable
- level 3 - Defined
- level 4 - Managed
- level 5 - Optimizing.[13]

B. Level one, and change management

The initial level organization has problems in cost overruns, schedule overruns, and poor quality. This level is also called the ad-hoc stage because each project is handled in

an arbitrary manner with little or no effort to learn from the past. This level lacks measurement and analysis of software production and maintenance activities. "Until the process is under statistical control, orderly progress in process improvement is not possible.[13]"

The following detailed steps allow an organization to get past the initial process level:

1. "Plan the work.
2. Track and maintain the plan.
3. Divide the work into independent parts.
4. Precisely define the requirements for each part.
5. Rigorously control the relationships among the parts.
6. Treat software development as a learning process.
7. Recognize what you don't know.
8. When the gap between your knowledge and the task is severe, fix it before proceeding.
9. Manage, audit, and review the work to ensure it is done as planned.
10. Commit to your work and work to meet your commitments.
11. Refine the plan as your knowledge of the job improves.[13]"

The above steps implemented on successive projects will keep the same types of problems from reoccurring. To go from the first to the second process level a group must implement proactive project management, effective management oversight, begin quality assurance measures, and implement change management.[13]

A process level assessment is an important first step for a strict SEI approach to process improvement. An outside group should perform the evaluation because of objectivity although a procedure for self evaluation is in the book Managing the Software Process.

Careful change management is a key improvement to an organization's process to bring it above the initial level. Without review of changes (meaning before commitment) work completed may be lost and schedules will probably be adversely affected - possibly without the requester's knowledge. A part of the desired outcome is to get the user's attention and make them think about the product being developed. Changes to software are reviewed from the analysis and design perspective to completely understand the impact of the change and to protect from the introduction of errors.[13] Software is frequently produced in a highly dynamic environment making change management difficult yet mandatory. The methods for controlling change include:

- careful analysis and design
- careful review of the analysis and design
- identification of volatile design areas before coding
- use of prototyping
- ability to delay the change until the next release
- use of rigorous software engineering.[2]

The benefits of change control are:

- project schedules become more achievable
- error rates decrease
- programmers are happier because schedules and defects are more controllable.[13]

C. Level two and level three

The second SEI process maturity level, the repeatable level, has stability of process allowing schedules and budgets to be accomplished on routine projects. An organization at the repeatable level is capable of fairly consistent project successes because of basic management control. The repeatable level does not have the ability to handle new and different situations with predictable results because it is only repeating the processes of previous accomplishments.[13]

To ascend from the repeatable level an organization needs to define the software process to the point that there is the capability to handle new types of projects effectively. While this does not entail volumous tomes of procedures, it does require:

- forming a process group - this is a special group that in a large organization can concentrate on, and is solely responsible for, process improvement
- reviews and inspections to enforce standards and provide consensus
- formal testing procedures
- initial use of software engineering technologies.[13]

"By way of analogy, Humphrey describes the differences between the levels by comparing different forms of advice that one might get when navigating to an unknown destination, for example, navigating from a downtown hotel in a strange city to an airport at the edge of town. In the level 1 organization, the driver doesn't ask for directions at all, but simply trusts his luck and intuition. The level 2 organization is comparable to the situation where the driver receives verbal instructions, for example, ""drive 2 miles until you see a gas station on the right, then take a right and go four traffic lights, then . . . "" problems with this form of navigation are (1) you don't know that you have made a mistake until it's too late and (2) once you do realize you've made a mistake, it's quite difficult to get back on the right path again. The level 3 organization, which we'll discuss next, is comparable to the driver with a road map: the written document not only helps her determine where she is, but also makes possible midcourse corrections.[17]"

At the third level an organization has defined their process. Development of a foundation to enable the next two levels abilities for continuous process improvement occurs at the third process level. Everyone in the organization knows and uses the written standards and procedures.[13] There is a cultural attitude requiring quality and pride in work performed. During the third level phase the following occurs:

- reporting of quality assessments to management
- implementation of a process metrics database
- quality assessment studies of products.

D. Level four and level five

The fourth level is the managed level. An entity at this level has a metrics program and is ready to benefit from the analysis of the data collected.[13] The metrics collected are beyond person hours, lines of code, and function points. See section V, part B of this paper for more information on metrics. The level four organization has quantified goals for software projects and can measure degrees of success.

At the fifth process level metrics analyses provide direction on ways to optimize the software process. Metrics analysis allows continuing optimizing of the process. This level is also called the optimizing level.

One cannot hope to jump from a level one or two to a level five in a period of months - a couple of years is a very fast track because of the required change in culture to that of quality orientation. It is also generally impossible to start an organization at the fourth or fifth level because:

- standards are required
- procedures are required
- metrics must be captured
- a history of metrics must be available for analysis (to guide process improvement).[13]

The use of object-oriented programming or the purchase of CASE workstations will not effect a change from one level to the next.[13]

General Suggestions

1. Pursue ISO 9000 certification. This is not a call for a large increase in the amount of documentation required. Find examples of topnotch, lean documentation and use them as models. "You're the expert" when it comes to what is needed.[14]
2. Use appendix 3 (goal/question/metric paradigm) to extend the suite of metrics already collected. Remember to emphasize automatic collection of metrics.
3. Expand metrics activities with an emphasis on automatic collection.[11,12,13,16]
4. Expand the review process of projects - especially the design and analysis review, and the post-project review. Emphasize this because reviews build consensus and standardization better than multitudes of procedures.[11,12,13,16]
5. Perform an analysis of a process repository including knowledge capture. Include a method such as automatic electronic mail to distribute new information.

6. After six months of extended metrics collection form a process group assigned to use the data for improvement.[11,12,13]
7. Seriously increase effort for the creation and sharing of reusable code.[3,13,16]

Summary

Through the extra effort of software process improvement an organization stands to benefit from increased productivity. The documentation required to define the process should already be available. Analysis of metrics and improvement of software processes will increase efficiency. The initial upfront effort required for process improvement will be paid back with decreases in errors and the other positive effects just mentioned. While the application of process improvement to the software development and maintenance field is relatively new, it will become standard practice in the near future.

Appendix 1

Expanded FURPS+ definition:
"Functionality
- Feature Set
- Capabilities
- Generality
- Security

Usability
- Human Factors
- Aesthetics
- Consistency
- Documentation

Reliability
- Frequency/Severity of Failure
- Recoverability
- Predictability
- Accuracy
- Mean Time to Failure

Performance
- Speed
- Efficiency
- Resource Consumption
- Thruput
- Response Time

Supportability
- Testability
- Extensibility
- Adaptability
- Maintainability
- Compatibility
- Configurability
- Serviceability
- Installability
- Localizability[11]"

Appendix 2

"Guidelines for creating surveys and interviews:
- Define what goals are for the survey, what questions must be answered, how the data will be analyzed, and how results will be presented. State or graph sample conclusions.
- Test the survey and your method of data analysis before sending it out.
- Ask questions that require simple answers, preferably quantitative or yes/no.
- Keep surveys short (preferably one page).
- Don't send surveys with other material so they won't get lost in the shuffle.
- Make them very easy to return (for example, a fold and seal, prestamped from).
- Formulate at least one question from each of the FURPS+ categories.
- Customer interviews are generally more accurate than surveys - with enough data[11]"

Appendix 3

"Major Strategies of a Software Business"[11]

MAJOR CHARACTERISTICS	MAXIMIZE CUSTOMER SATISFACTION	MINIMIZE ENGINEERING EFFORT & SCHEDULE	MINIMIZE DEFECTS
MAJOR BUSINESS FACTOR	Attempt to capture market share	Competitive pressures forcing new product development or cost control	Hold/increase market share
WHEN MOST EFFECTIVE	When initially entering market	When there are several competitive products or you sell more profitable products	When features are competitive and adequate market share is held
CHARACTERISTIC FEATURES	Customer communication quick responses	Focus on delivery dates and effort	Analysis and removal of defective causes
MOST VISIBLE METRICS	Survey and interview data, product metrics, defects	Calendar time, engineering effort, defects	Failure analysis by module, cause & severity; size ; code coverage
GROUP MOST LIKELY TO DRIVE STRATEGY	Development team initially, customer support later	Division company management	Development team and/or quality organization
GROUP MOST LIKELY TO BE IN DIRECT CONTACT WITH CUSTOMER	Development team	Marketing/factory customer support	Field support organization
POTENTIAL DRAWBACKS IF FOCUS TOO RESTRICTED	Process of developing products may not improve	Defect backlog can get unmanageable; customer and	Defects may be fixed that are not cost effective

Appendix 4

"GOAL: MAXIMIZE CUSTOMER SATISFACTION

Q1. What are the attributes of customer satisfaction?

M. Functionality, usability, reliability, performance, supportability (FURPS)

Q2. What are the key indicators of customer satisfaction?

M. Survey data, Quality Function Deployment

Q3. What aspects result in customer satisfaction?

M. Survey data, QFD

Q4. How satisfied are customers?

M. Survey data, customer visit data, number of customers severely affected by defects.

Q5. How do we compare with the competition?

M. Survey data, QFD

Q6. How many problems are affecting the customer?

M. Incoming defect rate

M. Open critical and serious defects

M. Break/fix ratio (count of defects introduced versus count of defects fixed)

M. Postrelease defect density

Q7. How long does it take to fix a problem (compared to customer expectation and commitments)?

M. Mean time to acknowledge problem

M. Mean time to deliver solution

M. Scheduled versus actual delivery

M. Customer expectation (by severity level) of time to fix

Q8. How does installing a fix affect the customer?

M. Time customer s operation is down

M. Customer's effort required during installation

Q9. How many customers are affected by a problem? (and how much?)

M. Number of duplicate defects by severity

Q10. Where are the bottlenecks?

M. Backlog status, time spent doing different activities

GOAL: MINIMIZE ENGINEERING EFFORT AND SCHEDULE

Q11. Where are the resources going? Where are the worst rework loops in the process?

M. Engineering months by product/component/activity

Q12. What are the total life-cycle maintenance and support costs for the product (and how distributed by time and organization)?

M. Engineering months, product/component/activity

M. Engineering months by corrective, adaptive, perfective maintenance

Q13. What development methods affect maintenance costs?

M. Prerelease records of methods and postrelease costs

Q14. How maintainable is the product as changes occur? When do I give up and rewrite?

M. Incoming problem rate

M. Defect density

M. Code stability

M. Complexity

M. Number of modules changed to fix one defect

Q15. What will process monitoring cost and where are the costs distributed?

M. Engineering hours and cost

Q16. What will maintenance requirements be?

M. Code stability, complexity, size

M. Prerelease defect density

Q17. How long does it take to respond to (fix) a defect? Historically? With new processes? With resource changes? With complexity and severity variations? For each activity in process?

M. Calendar time, process and module records

Q18. How can we predict cycle time, reliability, and effort?

M. Calendar time

M. Engineering time

M. Defect density

M. Number of defects to fix

M. Break/fix ratio-historical averages

M. Code stability

M. Complexity

M. Number of lines to change

Q19. What practices yield best results?

M. Correlations between prerelease practices and customer satisfaction data

Q20. How much do the maintenance phase activities cost?

M. Engineering time and cost

Q21. What are major cost components? What aspects affect the cost?

M. Engineering months by product/component/activity

Q22. How do costs change over time?

M. Track cost components over entire maintenance lifecycle

GOAL: MINIMIZE DEFECTS

Q23. What are key indicators of process health and how are we doing?

M. Release schedules met, trends of defect density, serious and critical defects

Q24. What are high-leverage opportunities for preventive maintenance?

M. Defect categorization

M. Code stability

Q25. Are fixes effective? Are unexpected side effects created?

M. Break/fix ratio

Q26. What is the postrelease quality of each module?

M. Defect density, critical and serious defects

Q27. What are we doing right?

M. Defect removal efficiency (ratio of prerelease defect density to postrelease defect density)

M. Break/fix ratio

Q28. How do we know when to release?

M. Predicted defect detection based upon prerelease records and postrelease defect densities

M. Branch coverage

Q29. How effective is the development process in preventing defects?

M. Postrelease defect density

Q30. What can we predict will happen postrelease based on prerelease data?

M. Correlations between prerelease complexity, defect density, stability, FURPS+, and postrelease defect density; ability to make changes easily; customer survey results

Q31. What defects are getting through? What caused those defects?

M. Defect categorization[11]"

Bibliography

1. Bollinger, Terry B. and McGowan, Clement. "A Critical Look at Software Evaluations." IEEE Software. Vol. 8, No. 4, July 1991, pp. 25-41.
2. Boar, Stephen P. and Rush, Tony W. "Rigorous Software Engineering: A Method for Preventing Software Defects." Hewlett-Packard Journal, December 1991, pp. 24-31
3. Brooks, Frederick P. Jr. The Mythical Man Month. Reading: Addison-Wesley Publishing Company, 1982
4. Brooks, Frederick P. Jr. "No Silver Bullet: Essence and Accidents of Software Engineering." Computer, Vol. 20, No. 4, April 1987, pp. 10-19.
5. DeMarco, Tom. Controlling Software Projects: Management Measurement and Estimation. Englewood Cliffs: Prentice Hall, 1982
6. DeMarco, Tom, and Lister, Timothy. Peopleware: Productive Projects and Teams. New York: Dorset House Publishing, 1987
7. Freedman, Daniel P., and Weinberg, Gerald M. Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products. New York: Dorset House Publishing, 1990
8. Gause, Donald C., and Weinberg, Gerald M. Are Your Lights On? New York: Dorset House Publishing, 1990
9. Gause, Donald C., and Weinberg, Gerald M. Exploring Requirements: Quality Before Design. New York: Dorset House Publishing, 1989
10. Grady, Robert B. "Measuring and Managing Software Maintenance." IEEE Software. Vol. 4, No. 9, September 1987, pp. 35-45.
11. Grady, Robert B. Practical Software Metrics For Project Management and Process Improvement. Englewood Cliffs: Prentice Hall, 1992
12. Grady, Robert B. and Caswell, Deborah L. Software Metrics: Establishing a Company Wide Program. Englewood Cliffs: Prentice Hall, 1987
13. Humphrey, Watts S. Managing the Software Process. Reading: Addison-Wesley Publishing company, 1989.
14. Rabbitt, John T. and Bergh Peter A. The ISO 9000 Book: A Global Competitor's Guide to Compliance and Certification. White Plains: Quality Resources, 1993
15. Rankos, John J. Software Project Management For Small To Medium Sized Projects. Englewood Cliffs: Prentice Hall, 1990
16. Yeh, Hsiang-Tao. Software Process Quality. New York: McGraw-Hill, 1993
17. Yourdon, Edward. Decline & Fall of the American Programmer. Englewood Cliffs: Prentice Hall, 199
18. Yourdon, Edward. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Englewood Cliffs: Prentice Hall, 1979