THE PROCESS GROUP

POST

Vol. 8, No. 1

HELPING YOU IMPROVE YOUR SOFTWARE ENGINEERING PROCESS

February 2001

Editor's corner

Improving software development organizations

The Process Group has written a draft manuscript for a book, describing the critical steps needed to plan and implement process improvement within a software development organization. It is based on our work with more than 3,000 software professionals in 100 companies around the world. We have seen what works and what does not. We have included stories and examples from companies using these ideas for improvement.

The book has been organized into four chapters based on the Shewhart cycle for planning and managing improvement (Plan, Do, Check, Act).

PLAN - In Chapter one, you will develop an improvement plan based on the business goals and problems of your organization. This approach addresses the frustration that many people experience when improvement programs do not relate to the product development work being done.

DO - Chapter two describes techniques for deploying new ideas across the organization. These techniques address frustrations such as resistance, unwieldy solutions and slow deployment.

CHECK - Chapter three describes techniques for checking the progress of your improvement program. Checking improvement is an essential activity to provide the organization with feedback when pursuing business goals and solving problems. The resulting data allows for early problem detection, early correction and visibility to management on improvement progress.

(Continued on page 2)

Getting product requirements – sorting the information

by Neil Potter

If you are a software developer or business analyst, you probably have had good and bad experiences of eliciting and defining product requirements.

When developers and business analysts elicit product requirements, they often ask the user to describe the work tasks he or she currently performs manually. The user often responds with a mix of tasks, data definitions, whims from memories of using other products, frustrations he or she has had with previous vendors and comments on the need for speed and reliability. This mixed bag of data is confusing and can lead to the creation of an unfocused and confusing product requirements definition. This, in turn, leads to products with deficiencies and bugs.

One effective way to start the requirements elicitation process is to have a predefined set of categories for sorting the data collected during a user interview session. A category scheme is also a good reminder of the topics you need to ask about in order to obtain a comprehensive set of requirements.

An example category scheme is shown in figure 1, based on the book "Software Requirements," by Karl Wiegers.

Information collected in an interview session with a user is tagged with one of the category names. For example, when the user mentions that the business-to-business Internet solution you are creating needs to replace all company transactions currently done on paper, put

(Continued on back page)

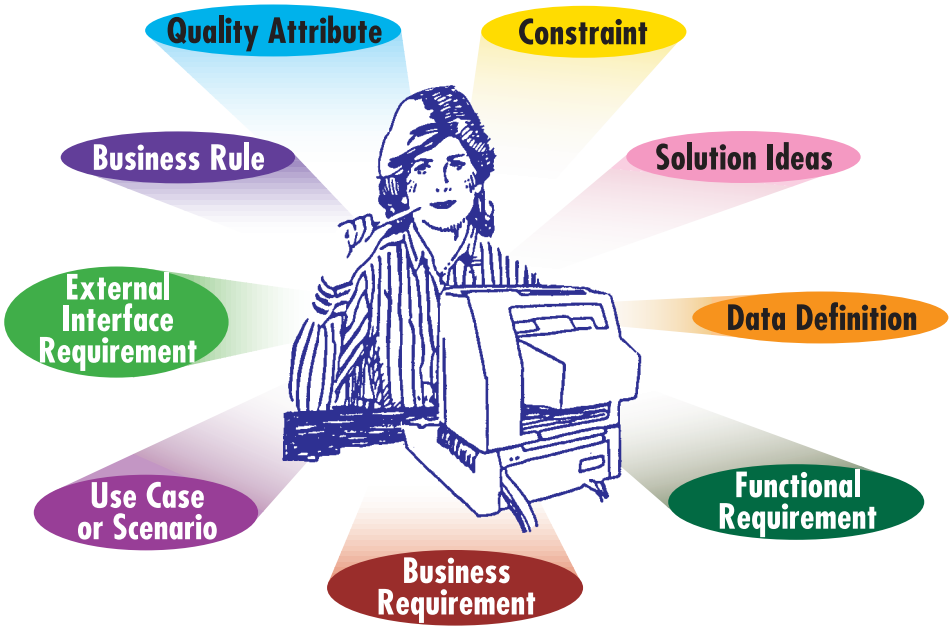

Figure 1 — Categories for sorting user requirements

Software development problems? We have answers now.

- ❑ **Understand customer needs. Clarify product requirements early.**
  In this workshop, IN SEARCH OF EXCELLENT REQUIREMENTS, software engineers, managers, requirements analysts and user representatives learn how to gather, document, analyze and manage customer requirements for software applications.

- ❑ **Decrease product development time-to-market.**
  In this workshop, ACCELERATING PRODUCT DEVELOPMENT FOR SMALL SOFTWARE PROJECTS THROUGH CYCLE TIME REDUCTION, project managers and their teams learn how to accelerate delivery through specialized schedule optimization techniques.

- ❑ **Launch projects effectively. Meet project deadlines and reduce risks.**
  In this three-day SOFTWARE PROJECT PLANNING AND MANAGEMENT workshop, project managers and their teams learn how to meet deadlines through better estimation, reduce surprises using risk management, schedule work for better optimization, understand and negotiate project trade-offs, and track progress.

- ❑ **Meet project deadlines. Scope and estimate the project work.**
  This one-day SOFTWARE ESTIMATION workshop (a subset of Software Project Planning and Management) helps teams develop more accurate estimates.

- ❑ **Avoid schedule delays caused by needless product rework. Find defects rapidly.**
  This two-day INSPECTION (PEER REVIEWS) workshop teaches teams to efficiently find defects in code and documentation. (Includes moderator skills.)

- ❑ **Hands-on SEI CMM/CMMI. Perform a mini-CMM gap-analysis.**

  The following workshops are available:

  ❑ SEI LEVEL 2 (one day), SEI LEVEL 3 (two days), SEI LEVEL 4 (one day).

  ❑ SEI CMMI—Overview of CMMI-v1.0 (one half-day presentation).

- ❑ **Identify critical changes to improve organizational results. Benchmark against the CMM.**
  A SOFTWARE PROCESS ASSESSMENT examines your organization's software practices and generates a focused list of the critical areas for improvement. Our SEI authorized Lead Assessors conduct customized CMM-based appraisals.

- ❑ **Goal/problem-based improvement.**
  This two-day SOFTWARE ENGINEERING PROCESS IMPROVEMENT workshop provides a systematic approach for organizations to improve their development capability. It includes: getting management support, focusing the organization on the critical issues, planning the improvement and effecting change.

- ❑ **Tailored assistance. Dedicated phone-based assistance.**
  This service consists of customized education and coaching on your specific problems (e.g., meeting deadlines, quality and cultural change.)

- ❑ **Audio cassettes:**
  * The Role and Focus of a Software Engineering Process Group (SEPG)
  * Making Change Happen—a 10-Piece Tool Box

*Detailed information on our services is available at **www.processgroup.com.***

*Contact us at **972-418-9541** or **help@processgroup.com** to discuss your needs.*

Editor's corner

Improving software development organizations

(Continued from page 1)

ACT - Chapter four provides examples of corrective action to get an improvement program back on track using the experience gained so far. It also discusses changing your process improvement approach based on lessons learned, making future executions of the Plan-Do-Check-Act cycle more effective.

The book is applicable to anyone wanting to start or refine a process improvement effort. The improvement model used as an example is the SEI CMM. If you would like to read this manuscript and provide feedback, please visit this web site: www.processgroup.com/tpgbook.htm

Mary Sakry

"Successful people keep moving. They make mistakes, but they don't quit."

—Conrad Hilton—

The Process Group

Mailing address: The Process Group
P.O. Box 700012
Dallas, TX 75370

Telephone number: 972-418-9541

Fax number: 972-618-6283

E-mail: help@processgroup.com
Web: www.processgroup.com

POST back issues are on line

Accelerating product development through cycle time reduction

By Dennis J. Frailey (Senior Fellow, Raytheon; Adjunct Professor, SMU; Process Group associate)
Edited by Mary Sakry

Your software is far from ready, but it's almost time to ship. You work overtime to rush through the most important functions — you skimp on testing, gloss over the quality assurance, and somehow get it out the door. Then you resign yourself to customer complaints, frayed nerves and a never-ending update cycle. Why is there never enough time to do it right? Shortening cycle time would give you a competitive edge — if you could figure out how to do it.

Here are some common causes that lead to lack of time and suggestions on what to do.

Delays

Seemingly innocuous things can cause unnecessary delays in software projects. For example, in one organization that we studied, the top three causes were: incompatible tools and data formats, inefficient approval procedures, and failure to plan effectively. Incompatible data formats add complexity to the process because you have to do extra work to convert the data and this creates bottlenecks that consume more time. Approval procedures are natural bottlenecks and often introduce variability due to unpredictable waits. On one maintenance project, for example, the average customer complaint took three months to resolve, and actual resolution times varied from one to six months. About 40 percent of the average delay was traced to an arcane approval procedure. We fixed the problem by creating a more efficient electronic approval procedure. The result was a reduction of more than a month in average complaint resolution time with a variance of only about 15 days. The worst case was now two and one-half months instead of six!

Excessive backlogs or WIP (Work In Process)

Cycle time problems are often easy to see. For example, a tell-tale symptom is a backlog of work, such as designs waiting to be reviewed or code waiting to be tested. Work In Process (WIP) is a necessary part of any process, but queuing theory states that the more WIP you have, the longer your cycle time. A simple equation that shows the relationship is, Average Cycle Time = WIP / Throughput.

To shorten cycle time, you must increase throughput and/or decrease WIP. However, it's hard to increase throughput without increasing WIP, so the smart approach is to reduce the WIP — the excessive backlogs in your process. For example, suppose you notice that products are waiting to be tested, but there is not enough test capacity. The products waiting are excessive WIP. The testing process is a bottleneck. The cause of the bottleneck might be insufficient test equipment, insufficient staffing of the test process, or inadequate maintenance of the test equipment.

Rework

A more fundamental symptom of cycle time problems is rework. Doing things over increases WIP, which adds cost and introduces delays. Much rework comes from simple things: rushing the work (introducing more errors), miscommunication (resulting in doing the wrong thing) and inadequate training (resulting in wasted time making mistakes on the job). Measuring rework — and taking action to reduce it — is an important cycle time improvement technique.

What you can do

Sometimes the actions needed to reduce cycle time go against intuition. For example, consider the Cycles of Learning technique, which says it is faster to do a job in several small increments, rather than to do it all at once. Intuition might suggest that it would be faster to do it only once. But by attacking a job in small chunks, you make mistakes and learn from them on the early cycles, then perform at top speed in later cycles. Most of us recognize this as true for software development: repeated or incremental development is often the fastest approach, provided you learn to do better each time through.

The Small Batch technique is also counterintuitive. We are accustomed to the concept of economy of scale, which says that large batches are more efficient because they require less overhead. But economies of scale don't always work. Why? Because large-scale economies are only realized when the requirements do not change and the process is close to perfect. Small batches reduce the amount thrown away or reworked when things change. If we designed and coded 10 modules, learned about our mistakes, and corrected them, we would only have to re-do 10 modules. The subsequent batches would likely have fewer problems.

The payoff

By attacking the root causes of cycle time problems, you can improve your delivery schedules permanently. When you deal with causes instead of symptoms, you save money and improve product quality as well. The techniques are not hard. You simply have to apply basic principles in a methodical fashion and be open to new ways of doing your work. The biggest problem is often selling your "counterintuitive" ideas, which is where measurement really helps. If you are not sure whether to try cycle time improvement, remember that your competitors are constantly striving to be faster than you.

(Continued on back page)

Getting product requirements – sorting the information

(Continued from page 1)

that piece of information in the Business Requirements category. A Business Requirement is a statement of a large business need being supported by the product you are creating.

When the user talks about detailed tasks he or she needs to perform with your product, label that comment as a Use Case or Scenario. The user might say, "I need to read data from my website and import it directly into my customer database," or "When the network measurement system is complete, I need to see an X-graph so that I can begin my analysis." The information collected in this session is the starting point for defining a complete Use Case.

During the session, listen for Business Rules. These are characteristics of the product that state when certain operations can be performed. Examples include: "The automatic de-icer can be invoked only when the plane is on the ground," "Money can be withdrawn from account type YY only with a supervisor's approval in the system," and "The pilot should not be able to raise the undercarriage when on the runway." Overlooking Business Rules can lead to serious product flaws.

Interwoven in the user's comments will be Solution Ideas — things the user has seen in other systems that he or she thinks are essential. These ideas might not make sense in the solution you are providing and should be labeled as Solution Ideas until they have been analyzed further. "Make all menus like the ones in Mac MSWord2001," or "I think Java would be a cool language for you to program in," are ideas that need further investigation before they become requirements.

A Quality Attribute describes how well the product will perform. This could include items such as robustness or user friendliness. "The system needs to run seven days each week, 24 hours per day," and "The product will still function when reading in foreign data," are Quality Attributes.

Constraints describe the world the product developer has to target when building the product. The Constraint, "The product must work on an isolated Windows 2000 machine with a maximum of 128MB of memory," would prevent the developer from assuming that memory can be added anytime, and that a network is available for data backup tasks.

The Data Definition category is a place to store descriptions of data used by the product, such as user record definitions, allowed password formats and supported file types. Each time you hear a reference to a piece of data, label it with Data Definition, so that data types are defined in one place.

External Interface Requirements state all of the interfaces that are needed to allow the product to work correctly in its target environment.

Functional Requirements are descriptions of the product components that will be built. These descriptions help the developer build the correct functionality into the product. Examples include, "The log-in command will verify a valid log-in ID," "The log-in function will reject numeric IDs that are less than four characters," and "Expired accounts will be validated with correct entry of the mother's maiden name." Functional Requirements can sometimes come from discussions with the user. Mostly, they are written by developers to describe what the product will do, in response to the needs collected in the interview session.

If you are a business analyst or developer involved in understanding product requirements, use this category scheme to organize the information you collect. The scheme can help you sort the needs of the user and allow you to create a more clear requirements definition.

The Process Group is licensed to teach *In Search of Excellent Requirements*, a two-day workshop based on the book *Software Requirements*, by Karl Wiegers, Microsoft Press, 1999.

Accelerating product development through cycle time reduction

(Continued from page 2)

The Process Group has collaborated with Dennis J. Frailey on a program called *Accelerating Product Development for Small Software Projects Through Cycle Time Reduction.* A more detailed version of this article is available on our web site.

For that article, or for more information about this workshop, please call or visit www.processgroup.com/cycletime-info.htm

THINK IT OVER

The world is full of willing people.
Some willing to work, the rest willing to let them.
—Robert Frost—

Do not tell me how hard you work.
Tell me how much you get done.
—James J. Ling—

In order to succeed, you must know what you are doing, like what you are doing, and believe in what you are doing.
—Will Rogers—