Adopting GQM-based measurement in an industrial environment

Schlumberger RPS integrated the Goal/Question/Metric approach into their existing measurement programs to improve their program performance. Key to their success was the use of feedback sessions as a forum to analyze and interpret measurement data.

Adopting GQM-Based Measurement in an Industrial Environment

Frank van Latum and **Rini van Solingen,** Schlumberger RPS, The Netherlands
Markku Oivo, Schlumberger SMR, France
Barbara Hoisl, Dieter Rombach, and **Günther Ruhe,** University of Kaiserslautern, Germany

Although software engineers generally agree that software measurement must be goal oriented, little has been published on the results of shifting to goal-orientation and still less on how to systematically make that transition. Thus, when we at Schlumberger RPS decided to adopt the Goal/Question/Metric approach, we resolved to document each step and record our experiences at each stage of the transition. This article is both a summary of our experiences and a brief description of how the GQM approach has greatly improved our measurement programs.

We have tried to describe the activities in our measurement programs in a way that will help more organizations shift to goal-oriented measurement. We hope that others will also document their efforts so that the software engineering community can better understand the benefits of the GQM approach.

0740-7459/98/$10.00 © 1998 IEEE

Steve McConnell's most recent book, *Software Project Survival Guide,* starts with a drawing from *Winnie-the-Pooh* by A.A. Milne and the following caption: "Here is Edward Bear, coming downstairs now, bump, bump, bump on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it…."

This new section, From the Trenches, is about sharing experience stories with colleagues in the software industry. From my perspective, one of the most effective ways to avoid bumping my own head repeatedly against the same problem is to learn from others. Case studies can teach us valuable lessons. Whether the outcome is positive or negative, they provide a wealth of practical knowledge representing years of experience and hard work.

In this section, we offer software professionals a forum to share experiences. Its pages will present practical experiences using some new methodology or technology, or give update reports about established tools and methodologies. Think about your own war stories. If you feel that your experiences could benefit others, please submit them.

Your story may just help others from bumping their heads the same way you did.

—Wolfgang Strigel, editor

Wolfgang B. Strigel is president of the Software Productivity Centre Inc., Vancouver, Canada; wstrigel@spc.ca; http://www.spc.ca.

Motivation and background

Schlumberger RPS, the Retail Petroleum Systems division of Schlumberger, manufactures and services systems for self-service petrol stations, such as fuel dispensers, point-of-sale systems, and electronic funds-transfer systems. Since Schlumberger began a software process improvement program in 1989,[1] the RPS division has achieved a level 2 on the SEI's Capability Maturity Model, Version 1.1. It has also introduced reviews and inspections, structured project planning and tracking, defect tracking, and product quality specification and evaluation. Processes have been certified to ISO 9001 and TickIT.

In late 1993, we began a systematic evaluation of the RPS division's measurement databases to assess the effectiveness of measurement practices. The databases contained data from most projects from the late 1990s to the time of the evaluation, representing more than 6,000 data points. Our evaluation revealed several weaknesses: Data points were often inconsistent and incomplete. Because each measurement was being done in isolation from the others, the measurements tended to lack context and an explicit purpose. We found that we could not even draw conclusions about the effectiveness of our practices.

Experiences with goal-oriented measurement in other organizations[2,3] seemed to offer a solution, so in 1994 Schlumberger RPS joined the Customized Establishment of Measurement Programs (CEMP) project,[4] which was already investigating heuristics and cost–benefit information on applying goal-oriented measurement in several European companies (http://www.iese.fhg.de/Services/Projects/Public-Projects/Cemp.html). In this context, we evaluated the feasibility of implementing the GQM approach.

Elements of the GQM approach

The GQM approach[5,6] is a systematic way to tailor and integrate an organization's objectives into measurement goals and refine them into measurable values. As Figure 1 shows, it is characterized by two processes: a *top-down refinement* of measurement goals into questions and then into metrics, and a *bottom-up analysis and interpretation* of the collected data.

The approach assumes that software development is still an immature engineering discipline and thus much is unknown about its characteristics and performance. Its aim, therefore, is to create information that will help people understand, monitor, evaluate, predict, control, and improve software development. Users can identify the

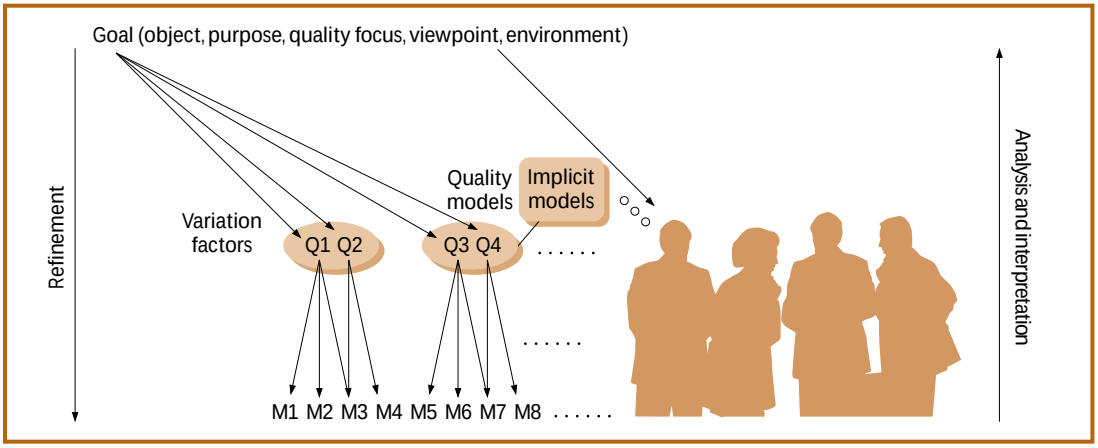

Figure 1. The GQM approach to goal-oriented measurement.

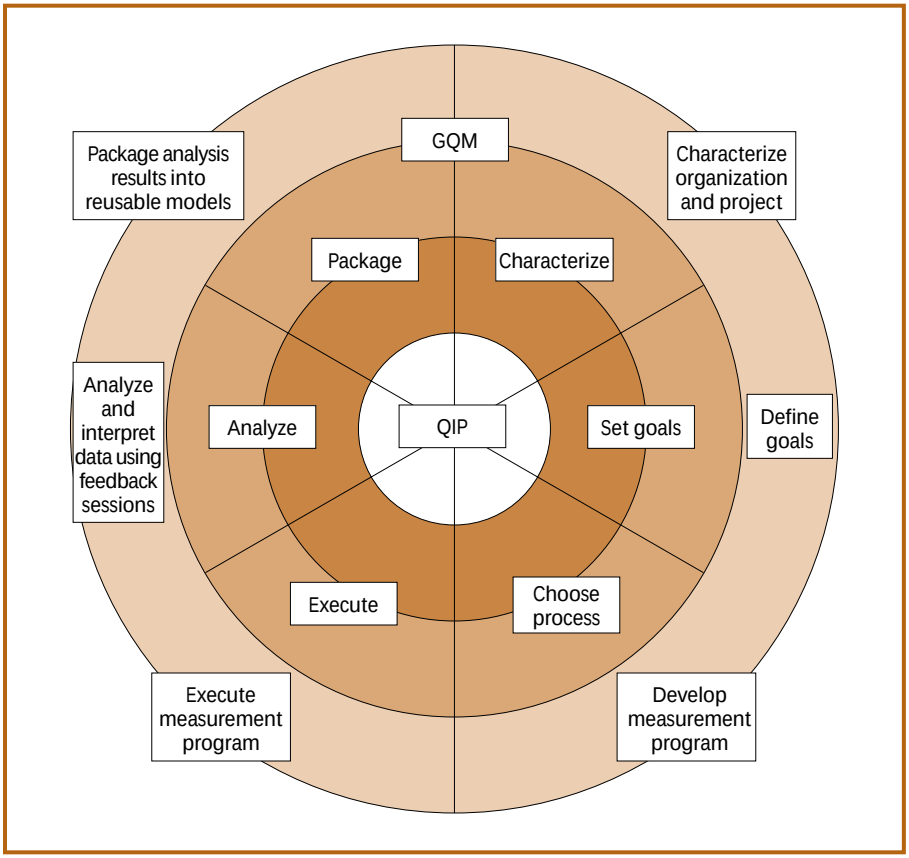

Figure 2. The activities of fitting the Goal/Question/Metric approach into a measurement program. We integrated the GQM approach into the Quality Improvement Paradigm.

information relevant to solve specific problems (goals) and represent it in a way that others can easily interpret. The basic ideas of the GQM approach[5,6] have been adopted in process improvement methods such as the Quality Improvement Paradigm[7] and AMI.[8]

Refinement

The GQM refinement steps are documented in a *GQM plan,* which has three parts:

- A *goal* that describes the measurement's purpose. By stating explicit goals, the measurement program is given a clear context. A GQM goal is described according to a template with five dimensions that express the object to be measured, the purpose of measurement, the measured property of the object (quality focus), the subject of measurement (viewpoint), and the measurement's context (environment).
- A *set of questions* that refines the goal and reflects the implicit models of the software developers. Questions related to *quality models* give a more detailed definition of the goal's quality focus. Questions related to *variation factors* describe attributes that are believed to affect the quality focus in that particular context.
- A *set of metrics* that serves to answer each question. Metric data may result from objective or subjective measurement and can be related to different scales as long as it is selected carefully.[9]

Data analysis and interpretation

The GQM approach also helps analyze and interpret results by describing the measurement data needed to answer the formulated questions. Independent of which analysis technique (statistical analysis, quantitative analysis, decision tree analysis) is applied, the results must be interpreted in close collaboration with the viewpoint team. Interpretation is generally done in a feedback session.

Fitting GQM to a measurement program

As part of the CEMP project, we established goal-oriented measurement programs in Schlumberger RPS, working closely with the teams who were running the projects the measurement programs were to support. We organized the GQM team separate from the project team to do all activities not directly related to the project, such as documenting the measurement program and preparing analysis material. Our aim was to ensure that project deadlines and stress would not endanger the continuation of quality activities. In fact, we found that close and confidential interaction between the project and GQM teams was crucial to successfully applying the GQM approach.

We have retained our initial composition and placement of the GQM team, except that during the first measurement program we had a GQM consultant acting as the coach. We have since become GQM experts ourselves so the coach is usually one of us. The GQM team is embedded in the quality assurance group and includes the quality assurance manager, a GQM coach, and a quality engineer. The team's main activities are to

- initiate measurement programs within development projects,
- carry out interviews and develop GQM deliverables,
- check data collection from the project team and handle available data,
- prepare feedback sessions by creating analysis slides,
- moderate feedback sessions,
- report progress to the project team and management, and
- disseminate results.

We have selected the first project for which we established a measurement program to illustrate our transition to a GQM-based measurement program. The project developed both software and hardware for a real-time point-of-sale system. The system was a second release and reused much of the software from the first release as well as from other projects. At the end of the project, the product contained more than 70,000 source lines of C code. The project team included a project leader, two hardware engineers, and two software engineers. Our measurement program supported the project from the design phase until a year after its release.

Figure 2 shows the sequence of six activities relative to the six steps of the Quality Improvement Paradigm. The QIP describes the improvement activities and can be applied to different tasks within software development (project level) and improvement (organization level). Each step has a corresponding measurement activity in the GQM approach, as the figure shows.

Characterize the organization and project

The primary tool for characterization is a questionnaire that includes both quantitative and qualitative questions, such as: What percentage of software people are within the organization? What life-cycle model does the project use? What is the average number of software product installations?

The GQM team has developed questionnaires, which either the project team or upper management fills in, depending on the kind of questions. If there is any confusion about the answers, the GQM team conducts a short interview to clarify points. We have used answers to these questions to define, prioritize, and select goals, as well as to build the GQM models.

For the sample project, results from the questionnaire showed that the organization's typical project lasts six months; involves real-time, embedded software products; and is developed using a waterfall model with five steps: requirements analysis and specification (REQ); high-level design (HLD); detailed design and implementation (IMP); integration; and evaluation and release. A separate quality assurance department controls the fulfillment of quality requirements.

Define measurement goals

Measurement goals can directly reflect business, project, and/or personal goals and must be based on selection criteria, such as priority to project or organization, risk, and time to reach the goal. Goals should take the form of [object to be measured], [purpose of measurement], [quality focus], [viewpoint], [environment]. In the sample goal below, the boldface words correspond to these elements:

> *Analyze the* ***delivered product*** *to* ***better understand it*** *with respect to* ***reliability and its causes*** *from the viewpoint of* ***the software project team*** *for the* ***Schlumberger RPS Project A.***

This product-oriented goal definition was the starting point in the process of understanding the product's reliability. The project team could then investigate how to describe that reliability quantitatively and determine what influenced it.

Within a brainstorming session, a project team can define many goals, but they may not know which are the most critical to their organization's business requirements. To help us prioritize goals, we devised a set of seven questions for the project team:

- What are your organization's strategic goals?
- What forces have affected your strategic goals?
- How can you improve your performance?
- What are your major concerns (problems)?
- What are your improvement goals?
- How can you reach your improvement goals?
- What are possible measurement goals and their priorities?

For the sample project, we selected measurement goals that emphasized reliability (in keeping with the characterization obtained in step 1) and reuse. The goal just described, for example, would give us a better grasp of reliability issues. Other goals gave us information about the cost and benefits of applying the GQM approach.

We consulted all project team members in defining our measurement goals and checked their degree of understanding and their motivation to reach them. We also involved management by explaining how the measurement goals would support their business activities. This involvement is critical. Without management support and commitment, the measurement program may ultimately fail.

To help us prioritize goals, we devised a set of seven questions for the project team.

Develop the measurement program

The major activity here is to refine selected goals into questions and metrics. The refinement itself has two main parts. The first is *knowledge acquisition*, in which the GQM team tries to capture the project team's current knowledge and represent it in quantitative or qualitative models. The second part is *measurement planning*, in which the team documents the GQM refinement and corresponding measurement procedures.

Knowledge acquisition

We conducted structured interviews to make the engineers' implicit knowledge explicit, capturing the project team's definitions, assumptions, and implicit models related to the goal. These interviews helped us

Object: Delivered product	Purpose: Better understanding	Quality focus: Reliability and its causes	Viewpoint: Software project team	Environment: Schlumberger RPS Project A

Quality focus	Variation factors
Number of failures • By severity (minor, major, fatal) • By detection (engineer, test group...) **Number of faults** • By life-cycle phase of detection • By modules **Cost for fixing faults (effort in hours)** • Cost by activity	**Process conformance** • Adherence to coding standards • Are the reviews done as prescribed in the process model? **Domain conformance** • Experience level of engineers **Attributes** • Complexity
Baseline hypotheses	**Impacts on baseline hypotheses**
Distribution of failures by severity • Minor 60% • Major 30% • Fatal 10% **Failure detection** • Engineer 10% • Test group 30% • OPCO 60% • Customer 0% **Faults by life-cycle phase of detection** • REQ: 5% • HLD: 10% • DD&IMP: 15% • Test: 70% **Top six fault-containing modules (ranked)** • DCT, DSS, MFT, MCF, MDS, TOT **Distribution of effort for fixing faults per introduced activities** • Requirements analysis/spec: 9.5 hours • High level design: 3.6 hours • Design and implementation 1.8 hours • Integration 1.8 hours • Evaluation and release: 3.6 hours	• Better process control results in: - Fewer failures - Fewer faults slipped through code review - Lower percentage of coding faults • Higher experience of software engineers results in fewer faults introduced • Better adherence to coding standards results in - Fewer faults in general - Less effort to locate and fix faults • Complex modules have more faults

Figure 3. Abstraction sheet for the goal "Analyze the delivered product to better understand it with respect to reliability and its causes from the viewpoint of the software project team for Schlumberger RPS Project A."

see discrepancies in different team member's views and gave us a way to discuss and clarify views. The interviews also helped us achieve buy-in for the measurement program because people realized the data to be collected is related to the very process and product issues they have raised.

During each interview, we recorded information on an abstraction sheet—a one-page presentation of the main entries, relationships, and related hypotheses for a goal in the GQM plan. Figure 3 shows an example for the goal described earlier. The abstraction sheets provide a structured approach that helps the GQM team focus on subjects relevant to the goal and to identify issues they may have overlooked.

As the figure shows, the sheet contains four quadrants:

♦ *Quality focus.* What are the measured properties of the goal's object? In the figure, the properties of interest are the number of failures and faults, and the cost. These are taken from reliability models that the project team defined.

♦ *Baseline hypotheses.* What is the current knowledge of measured properties? The baseline hypotheses are the project team's expectations of, in this case, the distribution of failures and faults, and fault handling. Again, these are based on the team's own reliability models. The failure classification and expectations are based on interviews of the team members or on reuse of former project results.

♦ *Variation factors.* Which factors are expected to most influence the models defined in the quality focus? An example is the level of code reviews. If the team plans to hold many code reviews, they are likely to find fewer faults during tests.

♦ *Impacts on baseline hypotheses.* How do the variation factors influence measurements? What is the principal kind of dependence assumed? For example, the more complex the module, the greater the likelihood of faults.

We found several uses for the abstraction sheet. The most frequently used one is to fill it in with the engineer, starting with the quality focus and baseline hypotheses and moving to the variation factors and corresponding impacts. After the GQM team and the engineer fill in all the quadrants, they repeat the process until the sheet reflects a reasonable balance between what is considered essential and what the organization can realistically measure and analyze.

Another way to use the sheet is to train engineers to fill it in themselves. This approach requires some training investment. Many people find it is hard to distinguish between the quality focus and the variation factors. Others find it hard to form hypotheses.

A third way is to fill in the abstraction sheet before the interview so that you have a "draft" version for the interview. The interview then serves to validate or invalidate the draft. However, this approach risks bias from the interviewers who are recording their implicit models, and requires the interviewers to have sufficient knowledge about the goal's context and subject.

Finally, you can use abstraction sheets as a guide when the team analyzes and interprets results. We found that they helped structure the presentation and interpretation of measurement data during feedback sessions (described later) because they reflect the most important questions.

Measurement planning

The aim of this stage is to produce a well-defined measurement program, documented in the GQM, measurement, and analysis plans. As we described earlier,

the GQM plan defines top-down what must be measured, moving from goal to question to metric. It is derived from the abstraction sheets for each goal. Figure 4 gives a sample question and the corresponding metrics derived from the abstraction sheet in Figure 3.

Both the GQM and project teams must review the GQM plan thoroughly to arrive at a set of standard terms for the process and product models and to eliminate obscurities and ambiguities. We used several short follow-up interviews and an extensive team review to refine the plan. We also used the plan to interpret and analyze the measurement data. Although the plan is strongly associated with top-down refinement, it is critical to the bottom-up interpretation activity as well.

The *measurement plan* describes the metrics, procedures, and media needed to report, collect, and validate data for each goal—basically embedding data collection into product development. The measurement plan contains

- a name and definition for each unique metric;
- the classification for each metric;
- an association point in product development that identifies when and how data is to be collected;
- definitions of the data collection forms;
- the procedures for data reporting, collection, and validation; and
- references to the metrics in the GQM plan.

The measurement plan let us integrate metrics into current development practices, which was key to successfully integrating the measurement program into our existing environment. We can often reuse data, measurement procedures, and measurement tools, which lowers the development overhead. We combined GQM-based metrics with existing metrics and data collection procedures using both automated tools and manual data collection forms.

The last deliverable in measurement planning is the *analysis plan*, which describes how to analyze and aggregate measurement data into presentation formats, such as charts and tables, that others can easily interpret and use to answer questions in the GQM plan. Much of the analysis plan time is spent describing how to compare actual data with the hypotheses defined in the interviews. There are many ways to analyze measurements, but not all will reflect key issues. We use the GQM plan during interpretation to help us focus on what the project team deems essential, which makes our analysis more meaningful.

Execute the measurement program

In this step, the GQM team collects the measurement data according to the procedures defined in the measurement plan and prepares the analysis material in line with the analysis plan.

Before presenting the measurement data to the project team, the GQM team must thoroughly verify and validate it, checking at a minimum for the data's completeness, timeliness, and accuracy, and conformance to the specified range, and ensuring that each case is classified uniquely. As they did in other steps, the GQM team worked closely with the project team while performing these checks.

The project team used an existing defect tracking tool to do most of the data collection. We also designed paper forms to record several additional metrics on fault handling, using Microsoft Excel to aggregate the measurement data and represent it in charts and tables.

Analyze and interpret results using feedback sessions

Regular and well-prepared feedback sessions have been key to the success of our measurement programs.[10] During these sessions, the GQM team meets with the project team to analyze and interpret measurement results. The session's main objectives are to discuss the measurement program results, have the project team interpret the data, and define actions for the next measurement period.

The feedback sessions must reflect the main principles of goal-oriented measurement. That is, the measurement program must address the interests of those providing the data and must be based on the project team's knowledge because they are the ones who best understand the measurement goals and the only ones who can accurately interpret the collected data.

We generally schedule feedback sessions every six to eight weeks. We found that this interval is long enough to have new and interesting results for the next session and short enough to keep people's interest and momentum. A typical session lasts about two to three hours; the main results are typically presented in 15 to 25 slides. Because we cannot present all measurement results in one session, we usually select a subset of slides to present and distribute hard copies of the full set.

All feedback sessions lead to slight or significant updates in the analysis, measurement, and GQM plans. The feedback sessions give us a better understanding of the underlying models, often prompting more ques-

Q_1	What is the distribution of failures by severity and detection mechanism?
M_1.1	for each detected failure: date, time, and unique number
M_1.2	for each detected failure: classification by severity (minor, major, fatal)
M_1.3	for each detected failure: classification by detection mechanism (engineer (provide name), test group (provide name), acceptance test group (provide name), OPCO (provide country), customer (provide name), other)

Figure 4. A sample question and corresponding metrics for the abstraction sheet in Figure 3.

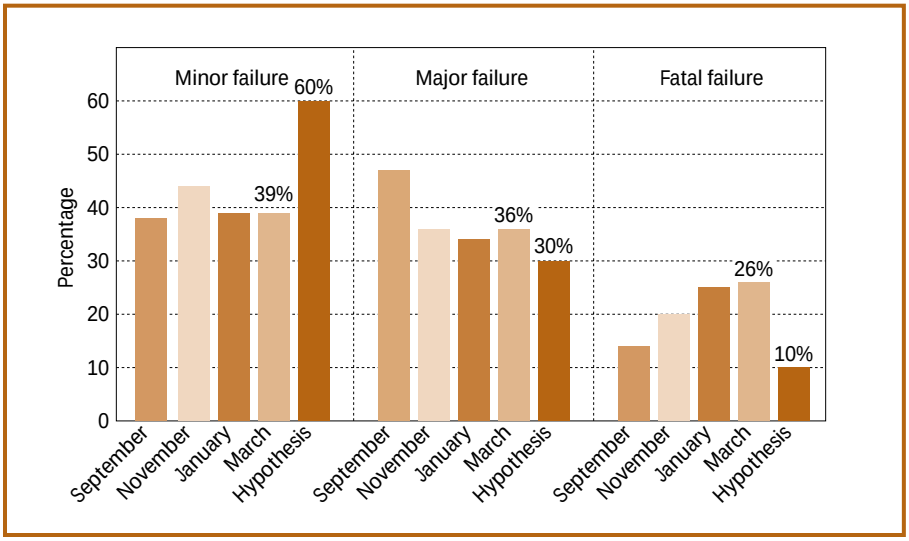

Figure 5. Trend of the severity of failures compared to baseline hypotheses.

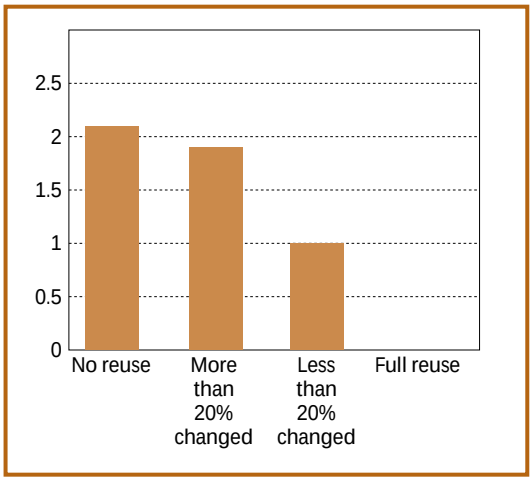

Figure 6. Fault density for the reuse categories. The bars represent faults per thousand lines of source code.

tions and occasionally even a new goal. For example, in the sample project, the project team defined a new GQM goal to analyze testing effectiveness.

The project team collected the measurement data using data collection forms. The GQM team imported the results into the quality assurance department's metrics database. For the sample project, we held nine feedback sessions to examine the validity of the hypotheses.

Figure 5 shows the distribution of the severity of failures within a seven-month period. The percentage of minor failures was actually 39 percent as opposed to the baseline hypothesis of 60 percent (see Figure 3). On the other hand, fatal failures appeared more frequently than expected (26 percent versus 10 percent).

Graphs such as Figure 5 can be used to identify a number of trends. In the sample project, we graphed the distribution of faults over the top eight faulty modules. The project team used the graph to select the modules they needed to review. In fact, they decided to rewrite one module because of the measurement results. We also graphed the distribution of failure reports over time, which gave the project team insights into process-related aspects, such as tests, releases, and holidays. In response to the data showing a higher than expected fatal failure rate (26 versus 10 percent), the team conducted additional investigations on the effectiveness of defect detection.

Armed with the measurement data, the project team was able to develop several rules of thumb:

- The average fault density is 1.9 per thousand lines of source code.
- The fault density for management functions is three times the fault density for dispensing functions.
- The fault density for console functions is two times the fault density for dispensing functions.
- The average effort needed to correct a failure in dispensing software is five times the effort needed to correct a failure in management functions.

These rules served as baselines, which let the project teams truly understand what "high" and "low" meant. They knew when corrective action was required and could better plan for the next project phases. Moreover, management could see that planning was accurate because the team had derived the baselines from systematic and accurate measurements in the same environment.

Package results into reusable models

An important benefit of the GQM approach is that it lets you reuse results and experiences in future projects. To do this, however, you must be able to package the results. This step is difficult because future needs for measurement information are generally unknown. Our project teams have been able to reuse certain packages, but we can offer no general guideline for reuse except that the more precisely you can describe your concrete experiences and their specific context, the greater your chances for adaptive reuse.

Our packaging effort consists mainly of developing the measurement database, updating development process definitions with the results of the GQM plan, and disseminating the measurement program results.

We store all measurement data in our measurement information system, which is available for all Schlumberger measurement programs. Entering new data in

the database automatically updates the feedback material. Every development process task also describes the metrics that can be collected after (or during) the task. This additional description came about as a direct result of the information in our measurement plan.

We present measurement program results on company bulletin boards and through monthly updates and project legacy reports to management. A frequently used package is the effect of reuse on product reliability. Although the project team considered the effect of reuse on productivity as important, their main concern was product reliability. We thus characterized all modules according to their reuse level and attributed every fault to new or reused software.

We classified all modules into four categories: no reuse, more than 20 percent changed, less than 20 percent changed, and full reuse. Figure 6 shows the fault density for these categories. When we presented the results during the feedback session, the project team saw at a glance that the average number of faults introduced in new software is significantly lower when the module is reused than if a module is developed from scratch. They also realized that existing faults in a fully reused module are simply inherited from the original. The figures helped them quickly identify trends and needed changes. For example, because of Figure 6, they decided to define stricter criteria for a module to qualify as reusable.

Results

The GQM-based measurement approach has been implemented as part of an organization-wide process improvement program. As a direct result of the measurement program, interest in software reuse has increased and several other projects have renewed their efforts to develop reusable modules. The reuse package described earlier is still frequently used within the organization to show the benefits of reuse.

The GQM approach helped our organization avoid many of the pitfalls we experienced in former measurement activities:

- By characterizing the measurement context, we avoid defining irrelevant (too ambitious, not of interest, or unimportant) measurement objectives and more effectively consider accompanying organizational and project-related context factors as part of the measurement program.
- By formulating an explicit measurement purpose, we avoid data "cemeteries" and more effectively exploit measurement results.
- Through regular feedback of the measurement results, we avoid misinterpreting data and more synergistically integrate human expertise and formally derived analysis results.
- By actively involving the project team, we avoid including irrelevant attributes and metrics and increase the project team's motivation (because they understand the program's purpose).

We can't say enough about the importance of the feedback mechanism. It is essential to any successful measurement program. The feedback sessions let the project team really evaluate the critical project issues, forgetting little details for the moment. They were able to define actions and investigate problems. These sessions let us initiate real process improvement on the project level. Because our project teams met frequently, they could evaluate progress and define improvements to their daily processes—which is what software process improvement is really about.

Schlumberger RPS now has five measurement programs in its software engineering department, and they are considered key to the division's overall software process improvement program.

We believe others can adopt a similar approach and customize it to their own organizations. We also believe they can achieve results similar to ours, and we look forward to reading about additional experiences with the GQM approach. ❖

Acknowledgments

We thank the Schlumberger RPS project team for their support and the participants of the CEMP consortium for all the useful discussions and suggestions during the project. CEMP was funded by the European ESSI program as ESSI project 10 358.

References

1. H. Wohlwend and S. Rosenbaum, "Schlumberger's Software Improvement Program," *IEEE Trans. Software Eng.*, Nov. 1994, pp. 833-839.
2. R. Grady, *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, Upper Saddle River, N.J, 1992.
3. "Software Measurement Guidebook," tech. report, NASA Goddard Space Flight Center, Greenbelt, Md., 1994.
4. "CEMP: Customized Establishment of Measurement Programs," IESE-Report 001.96/E, FhG IESE Kaiserslauern, 1996.
5. V. Basili and D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Trans. Software Eng.*, June 1988, pp. 758-773.
6. V. Basili and D. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Trans. Software Eng.*, Nov. 1984, pp. 728-738.
7. V. Basili, G. Caldiera, and D. Rombach, "Experience Factory," in *Encyclopedia of Software Eng.: Vol. 1*, J.J. Marciniak, ed., John Wiley and Sons, New York, 1994, pp. 469-476.
8. K. Pulford, A. Kuntzmann-Combelles, and S. Shirlaw, "AMI, A Quantitative Approach to Software Management," *The AMI Handbook*, Addison Wesley Longman, Reading, Mass., 1992.
9. N. Fenton and S. Pfleeger, *Software Metrics: A Rigorous Approach*, 2nd ed., Chapman & Hall, London, 1996.
10. B. Hoisl et al., "No Improvement without Feedback: Experiences from Goal-Oriented Measurement at Schlumberger," *Proc. European Workshop Software Process Technology*, *Lecture Notes in Computer Science*, Vol. 1149, Springer-Verlag, Berlin, 1996, pp. 168-182.

Frank van Latum is research and development manager at Dräger Medical Technology. The Netherlands, on leave from Schlumberger RPS. His main interest is the professional management of product development with emphasis on product and process quality. Other interests include real-time system software development for electronic test systems, retail petroleum systems, and medical systems.

Van Latum received master's degrees in mathematics and computer science from the University of Nijmegen. He is a member of the IEEE Computer Society.

Rini van Solingen is a quality engineer at Schlumberger RPS, where he is involved in the quality measurement and improvement of embedded products. He is also a researcher at Eindhoven University of Technology, where he is a PhD candidate in quality improvement of embedded products.

Solingen received an MSc in computer science from the Delft University of Technology.

H. Dieter Rombach is a professor of computer science at the University of Kaiserslautern, where he also holds a chair in software engineering, is codirector of a basic engineering research institute (SFB), and director of the Fraunhofer Institute for Experimental Software Engineering (IESE). His research interests are software methodologies, modeling and measuring the software process and resulting products, software reuse, and distributed systems. He also heads several research projects and consults on quality improvement, software measurement, software reuse, process modeling, and software technology in general. He has written more than 70 articles and papers. In 1990, he received the Presidential Young Investigator Award for his research in software engineering. He is an associate editor of Empirical Software Engineering and is a member of the GI, IEEE and ACM.

Rombach received a BS in mathematics and an MS in mathematics and computer science from the University of Karlsruhe, and a PhD in computer science from the University of Kaiserslautern.

Markku Oivo is a chief research scientist and head of a software engineering group at VTT Electronics. He is responsible for initiating and managing both applied research projects and industrial development projects for a broad range of clients in software engineering. His interests include software engineering, software process improvement and measurement, production of embedded software, object-oriented methods, and quality assurance and improvement. He is also a quality manager of VTT Electronics and a docent in software engineering at the University of Oulu.

Oivo received an MSc in computer technology and a PhD in software engineering, both from the University of Oulu. He is a member of the IEEE Computer Society and ACM.

Barbara Hoisl is a software architect at the OpenView Software Division of Hewlett-Packard in Germany. Her technical interests include software architecture and systematic, rigorous software development processes for distributed, object-oriented systems. Her work involves the application of these methodologies to the architecture of large-scale, globally distributed IT management solutions. When this article was written she was involved in the CEMP project as a research assistant at the University of Kaiserslautern, where she coached and consulted for the industrial project partners on introducing GQM-based measurement.

Hoisl received a master's in computer science and business administration from the University of Kaiserslautern.

Günther Ruhe is deputy director of the Fraunhofer Institute for Experimental Software Engineering, Kaiserslauern. His software engineering interests include experimental analysis and design, knowledge discovery and data mining, software measurement, and industrial software improvement. He has published one monograph and more than 50 papers.

Ruhe received a master's in mathematics with emphasis on operations research from Leipzig University, a PhD in operations research from Freiberg University, and an habilitation (Dr. Sc. Nat) from the Leipzig University of Technology. He is a member of the IEEE Computer Society, ACM, the German OR Society, and the German Computer Society.

Send questions about this article to Rombach or Ruhe at Fraunhofer Institute for Experimental Software Engineering, Sauerwiesen 6, D-67661 Kaiserslautern, Germany; {rombach,ruhe}@iese.fhg.de.