IEICE Transactions on Information and Systems, Vol.E95-D, No.9, pp.2169-2182, September 2012.

Customizing GQM Models for Software Project Monitoring

Akito Monden† **Tomoko Matsumura**†,†† **Mike Barker**†
Koji Torii† **Victor R. Basili**†††,††††

† Nara Institute of Science and Technology, 8916-5 Ikoma, Nara 630-0192 Japan.
††Presently, with Hitachi, Ltd.
†††University of Maryland, College Park, MD 20742, USA.
††††King Abdulaziz University, Jeddah, Saudi Arabia.

SUMMARY This paper customizes Goal/Question/Metric (GQM) project monitoring models for various projects and organizations to take advantage of the data from the software tool EPM and to allow the tailoring of the interpretation models based upon the context and success criteria for each project and organization. The basic idea is to build less concrete models that do not include explicit baseline values to interpret metrics values. Instead, we add hypothesis and interpretation layers to the models to help people of different projects make decisions in their own context. We applied the models to two industrial projects, and found that our less concrete models could successfully identify typical problems in software projects.
key words: *Software Process Evaluation, Software Measurement, Empirical Software Engineering, Process Improvement.*

1. Introduction

People very often use software measurement standards but they are not always easy to apply in a particular environment where the data, needs, and context are different. Without an organization's explicit goals, and the link between goals and measures and the interpretation of those measures, the organization will not get a chance to see the measures as satisfying their goals.

The aim of this work is to develop a set of Goal/Question/Metric (GQM) models [1][2] for the purpose of project monitoring, i.e., assisting project managers in controlling the software development processes. The models should take advantage of the data being supplied by a software project monitoring tool called the Empirical Project Monitor (EPM) [9][10][14][15]. The evaluation mechanisms will be applied to projects in various organizations, each with their own context and criteria for success. For this reason, the standard GQM process must be customized to take advantage of these two criteria: a given data set and abstract evaluation modes that can be determined by each organizations own needs and context.

The measurement tool EPM was developed by the EASE (Empirical Approach to Software Engineering) project of MEXT, Japan [10][11] and currently being enhanced and maintained by the Information-technology Promotion Agency, Japan [9]. To date, software companies have used EPM as a probe to record empirical data in software projects without disturbing their development activities [16]. EPM automatically collects time series data from commonly-used software development management tools such as configuration management software, issue tracking software and mailing-list software.

This paper fully exploits the potential of the GQM method [1][2] to explicitly associate the organizational goals of the EPM user companies with available data (metrics) so that the resultant GQM models become a powerful means to drive an empirical process of evaluation and improvement in companies. The GQM method is a goal-oriented measurement framework that explicitly associates goals with metrics (Figure 1). A goal is refined into a set of questions that must be answered to achieve the goal, and then into metrics to collect to provide the necessary information for answering the questions and thus evaluating goal achievement.

To build adequate GQM models, this paper attempts to balance the following two requirements.

(Req.1) Narrow down the focus of the GQM model to a particular measurement context where EPM is used.

(Req.2) At the same time, the GQM model must allow variations of project contexts for different organizations.

To satisfy these requirements, our basic idea is to build less concrete models that do not include explicit baseline values to interpret metric values, while they include a "hypothesis layer" and an "interpretation layer" to help people of different projects and/or organizations make decisions in their own context (Section 5.)

The rest of this paper describes our experience to customize the GQM models with the following Steps 1 to 4.

[Step 1: Identify Goals] Conduct a survey of potential EPM user companies to clarify their goals.

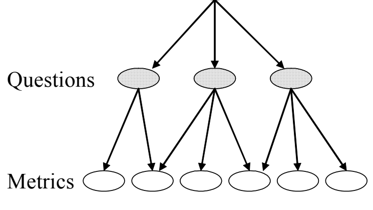

Fig 1. GQM paradigm

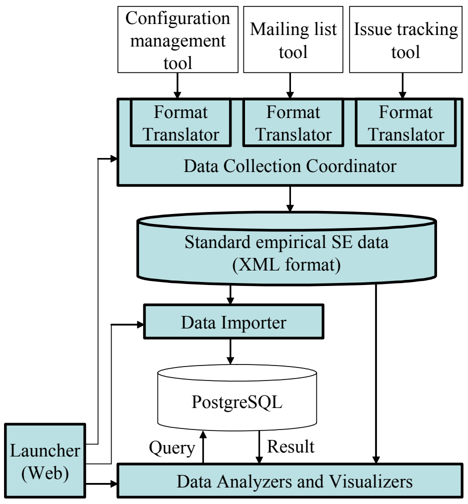

Fig 2. Overview of EPM

[Step 2: Build an Initial GQM model] Build an initial GQM models to map the goals and metrics.

[Step 3: Refine the GQM models] Conduct a preliminary analysis of empirical data collected in pilot projects (including open source software projects). This step refines the initial GQM models based on actual data.

[Step 4: Apply to project] Analyze industrial project data based on GQM models, and get feedback from companies.

2. EPM: Empirical Project Monitor

2.1 Overview of EPM

Figure 2 shows an overview of EPM. The *data collection coordinator* collects data from three types of development support tools: versioning histories from a configuration management system (e.g. CVS and Subversion), mail archives from a mailing list manager (e.g. Mailman, Majordomo and fml), and issue tracking records from a bug/issue tracking system (e.g. GNATS and Bugzilla). Because these data are accumulated through everyday development activities in common projects developers and managers do not need to do additional work for data collection.

Appendix A shows typical data collected by EPM. EPM converts these data into a XML file format (called the *standard empirical SE data format*) so that data analysis tools can deal with them in a systematic manner. This XML format excludes privacy information, such as the text of email messages, to avoid privacy violations. Data from other measurement tools can be also converted into this format by EPM plug-in modules (*data format translators*).

The XML data are then imported to a PostgreSQL database via the *data importer*. The XML data can also be imported to other database or spreadsheet software, e.g. Microsoft Access and Excel. This import can be executed every day (or every week) so that project managers can inspect the analysis results as a daily (or a weekly) task.

Data analyzers and visualizers are linked with database software (PostgreSQL). The current EPM provides simple data analysis and visualization features including a 2D graph visualizer and SRGM (Software Reliability Growth Model) tool. All the data analysis and visualization operations in EPM are performed through a web-based *launcher*.

2.2 What EPM can support

EPM can be used to collect data whenever an issue is reported or program source code is being written/modified, e.g. requirement/design review, coding, testing and maintenance phase.
Currently, EPM does not completely support project management since it does not collect data about cost management and workflow management.

2.3 Installation and administration

To use EPM in a software development project, a company only needs to install EPM on a Linux PC for data collection. The constraint in the development process is that developers must use provided tools (e.g. CVS, GNATS, Mailman, etc.) under a common operation rule, e.g. developers must check-in their (modified) source code into CVS once a day.

3. Goal Identification

3.1 EPM User Company Survey

We have been organizing an empirical software engineering workshop every six months in Tokyo. Attendees of the workshop are mainly project managers from industry including potential EPM user companies. We took this opportunity to capture industry needs (goals) as a part of top-down analysis in GQM modeling. Since EPM is supposed to be used for project monitoring, we focused on symptoms of troubles (e.g. project delay, insufficient quality or cost overrun) that can be monitored and detected early by empirical data of EPM. Specifically, we asked the attendees about the project delay and its reasons (source problems), symptoms and desirable solutions via a questionnaire sheet.

There were 20 responses from 12 companies. 18 of them answered that they had experienced a serious delay of a software project. Figure 3 shows answers to a question “In which phase of a project did you find the delay? Please answer with typical cases you have experienced.” As shown in the Figure, delays are mainly

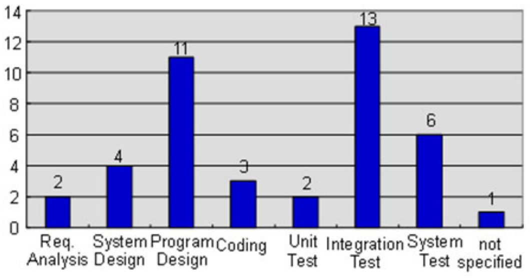

Fig 3. Phases where project delay were found

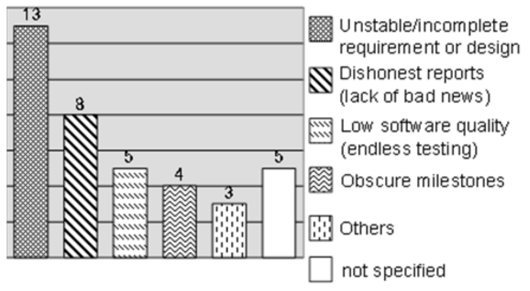

Fig 4. Unsolved problems in projects

observed in the "program design" phase and "integration test" phase. From further investigation, it turned out these delays are mainly caused by earlier phases, e.g. delays in the program design phase were due to poor requirement analysis, poor system design, frequent changes of requirements or inaccurate project estimation. Similarly, reasons behind the delay of integration testing include poor unit testing, system design and requirement analysis.

Interestingly, 17 out of 20 people answered that there were explicit symptoms of project delay. Table 1 shows what they observed as symptoms. Some of them can be easily detected by EPM. Some of them can be easily detected by EPM. For example, "too much increase of reported bugs" and "too slow bug (issue) elimination" are detectable from issue tracking data.

Figure 4 shows unsolved problems that respondents are facing in their software projects. Unfortunately, the current EPM does not cover the problem "obscure milestones." However, other problems can be detected and/or solved by EPM. For example, "unstable/incomplete requirement or design" can be detected in a review or a coding phase by CVS and GNATS data. In addition, the problem of dishonest reports can be solved by EPM since it automatically records development activities including bad news.

3.2 Related Studies on Goal Identification

The GQM Approach has been around since the early 1980s. Basili and Caldiera [4] discuss the major concepts of their approach to quality improvement, including the

Table 1. Symptoms of project delay

Symptoms	# of answers
Too much increase of reported bugs	4
Increase of design changes / insufficient design	3
It became more and more difficult to finish tasks in a scheduled time.	3
Lack of review	2
Increase of cost (effort) but no deliverables	2
Confused instructions	2
Progress report says "completed" but there is no evidence.	1
Estimated size grows every time we conduct project size estimation.	1
Too few pages of design document	1
Too slow bug (issue) elimination	1

quality improvement paradigm (QIP), goal-question-metric approach (GQM), and the experience factory (EF) for collecting, organizing, and reusing knowledge and experience. They point out that manufacturing quality improvement is based on repetitions of the same process, while software models are based on "the ability to learn from other software development projects" (p. 56). "The goal/question/metric (GQM) approach provides a method to identify and control key business processes in a measurable way" (p. 58). The goal is described as specifying the purpose, object, issue, and viewpoint of the measurement. From this, several questions are developed around the major components of the issue. Then these questions are refined to determine which metrics will allow them to be answered.

Mendonca, Basili, Bhandari, and Dawson [12] describe how attribute focusing, a knowledge discovery approach, and GQM, a measurement planning approach, combine to help "understand and structure ongoing measurement" and "discover new interesting information in the legacy data" (p. 484). They provide the template: Analyze "object of study" in order to "purpose" with respect to "focus" from the point of view of "point of view" in the context of "environment" (p. 485). This article also provides an example of how to use GQM with existing metrics, which can help identify unnecessary data collection and additional data collection which is not currently being accomplished.

Briand, Morasca and Basili [6] extend the basic GQM approach to include a measure definition process, providing "a practical guideline to design and reuse technically sound and useful measures." (p. 1107). This paper defines the template for GQM as the object of study, purpose, quality focus, viewpoint, and environment (p. 1112). They add detail to developing metrics through the steps of formalizing, identifying abstractions, instantiation and refinement of properties, definition, and validation of measures.

Business Goal: Does the EPM data support the project reports, e.g. are the requirements unstable, is the design poor or incomplete, is the product quality poor?

GQM Goal: Analyze **CVS and GNATS data for file change patterns** for the purpose of **evaluation** with respect to **requirements instability, poor design, or poor product quality** from the point of view of the **project manager** in the context of the particular project in the company

Questions: What is the level of file change over time (FCM)? What is the number of lines of code changed (LCC)? What GNATS class is being made (bug, B, or change request, CR)? How many people are changing a particular file (CVS Owner - ONR)?

Proposed Model:
If FCL > 0.5 and (LCC/file size > 5%) and (60% of GNATS reports have class = CR) then the requirements are unstable
If ((FCL > 0.5) and ((LCC/file size > 5%) or (# ONR for > 25% of files is >=2)) then there is a poor design
If (FCL > 0.5 and (GNATS class = B)/KLOC > 10 then we have a low quality product.

Assumption: All the numbers in the formula need to be verified and evolve with data over time (they are currently the best guesses).

Fig 5. One of the Initial GQM models

3.3 Business Goal

Before defining GQM (measurement) goals, we defined a "business goal" to clarify our (EASE project's) strategic target. Since the mission of EASE is to promote an empirical approach (in a narrow sense, EPM) for companies to solve (or at least detect) their problems, our business goal would be to clarify how the EPM data can support the project reports described in Section 3, e.g. are the requirements unstable, is the design poor or incomplete, is the product quality poor?

3.4 Measurement Goal

To define measurement goals, we used a goal template [6][12] as a specification of the problem we want to solve. The template is described as follows:

Analyze "object of study" in order to "purpose" with respect to "focus" from the "point of view" in the context of "environment."

In our case, "object of study" is data collected by the EPM. "Purpose" is evaluation, understanding or characterization of an ongoing (or a past) project. "Focus" is an EPM user company's problem, i.e. requirement instability, poor design, or poor product quality. "Point of view" is a project manager. "Environment" is a particular project in a company.

4. Building Initial GQM Models

Figure 5 shows one of the initial models we built. Below describes how we built them.

4.1 Questions and Metrics

After defining measurement goals (Section 3.4), "questions" are then defined as refinements of the goals. Questions should be focused on metric selection so as to interconnect goals with metrics. In our case, available metrics are pre-determined (as shown in Appendix A) since we use EPM to collect metrics. Therefore, we defined questions based on both measurement goals and EPM metrics.

There are several undefined metrics (FCL, LCC, etc.) in the questions, and there are several possible definitions for them. Below describes our initial definition for FCL (file change level).

$$\text{FCL} = \frac{\text{\# of file modifications}}{\text{\# of total files}}$$

The value of FCL is calculated at each check-in point of source files in the configuration management system (CVS). As a project progresses, the number of total files will increase (note that this is not a cumulative number). At the same time, developers may modify existing files. If this modification frequently occurs, the value of FCL will increase.

These questions essentially embrace one or more hypotheses that interconnect goals with metrics, e.g. "frequent changes are due to unstable requirements, poor design or low quality target." It can be considered that increase of FCL over time indicates unstable requirements, poor design or low quality target.

4.2 Quantitative Models

Based on questions and underlying hypotheses, we built quantitative models that explicitly interconnect goals with metrics as shown in Figure 5. These models consist of formulas with metrics and baseline values, e.g. "If FCL > 0.5 and (LCC/file size > 5%) and (60% of GNATS reports have class = CR) then the requirements are unstable." However, currently there is no basis for these baseline values. The values need to be verified and evolved continuously.

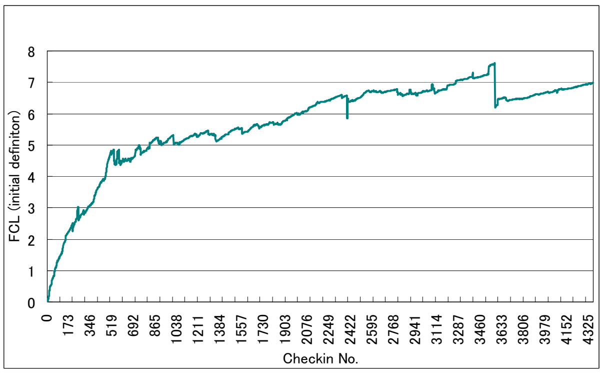

Fig 6. FCL of Azureus

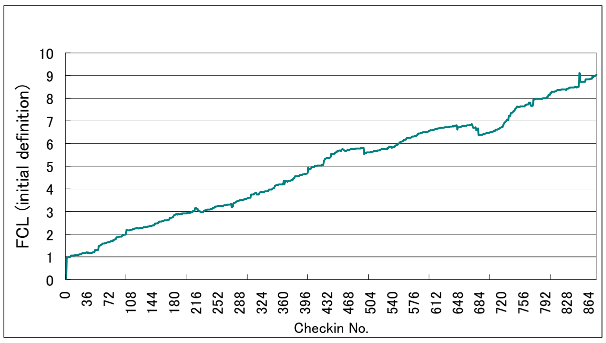

Fig 7. FCL of DoomLegacy

5. Refinement of GQM Models

5.1 Requirements

We have encountered several problems when we applied initial GQM models to pilot projects. Below describes requirements for refining initial models.

Requirement R1: Narrowing goal focus

Goals were too broad in initial models. In each initial model, we picked up multiple problems that can be detected via EPM, e.g. "unstable requirements", "incomplete design" and "poor product quality." To clarify the goal focus of a model, we decided to define a high level goal then decompose it into subgoals so that each subgoal addresses only one problem. Also, we found that some symptoms (in Table 1) are likely to be related with a specific root cause that needs to be identified as a subgoal. For example, "too slow bug (issue) elimination" and "it became more and more difficult to finish tasks in a scheduled time" are related to the problem of human resource allocation.

. These subgoals are considered "software goals" in GQM+ strategies [5][18], which are extensions to GQM paradigm to fill the gap between business goals and measurement goals.

Requirement R2: Building less concrete models

Since we are not focusing on a single project for our GQM models, we found that it is very difficult to develop a concrete model that has appropriate the baseline values for multiple projects. It is because there are too many (hidden) individual variables that may affect the baseline values (e.g. developer's skill level, customer involvement, process model being used, etc.) For example, Figure 6 and 7 shows examples of time series values of *FCL* (File Change Level) measured from two open source projects (Azureus and DoomLegacy). As we see in the Figures, the value 0.5 in the initial model's formula "if FCL > 0.5 ….. then the requirements are unstable" seems to be inappropriate for these two projects. Moreover, the baseline itself may not exist for DoomLegacy project as FCL keeps increasing all the time (Figure 7).

Moreover, in the context of project monitoring, we found that focusing on the (sudden) increase or decrease of metric values is more important than focusing on the value itself since any "change" could be a symptom of a problem.

Instead of trying to develop concrete models having baseline values, we decided to develop less concrete interpretation models that do not have baseline values. In these models we focus on the change metrics values. For example, an interpretation model would be "if FCL is high relative to the number of files then you might be concerned about requirement instability."

Requirement R3: Explicit hypothesis description

Since we employ less concrete interpretation models, which target multiple projects and/or organizations in Requirement R2, we need to clarify the underlying hypotheses that interconnect goals, questions and metrics to help people in different organizations make decisions in their own context based on the observed metrics values. If a project manager of a certain organization thinks that one of the hypotheses does not match the project context, then that part of the GQM model should not be used.

Although questions essentially embrace one (or more) underlying hypotheses that interconnect goals with questions, they must be clear in the model. For example, suppose we have a goal "evaluation of requirement stability," then an underlying hypothesis can be, "if requirements are unstable, then frequent changes of source code can be seen (i.e. unstable requirements imply that there will be frequent changes of source code)." After clarifying such hypotheses, we will be able to define a clean-cut question "what is the code change frequency?"

In our refined models, we decided to explicitly describe hypotheses that interconnect goals with questions so that the resultant models become much easier to understand. We propose hypotheses should be a combination of "cause (e.g. unstable requirements)"

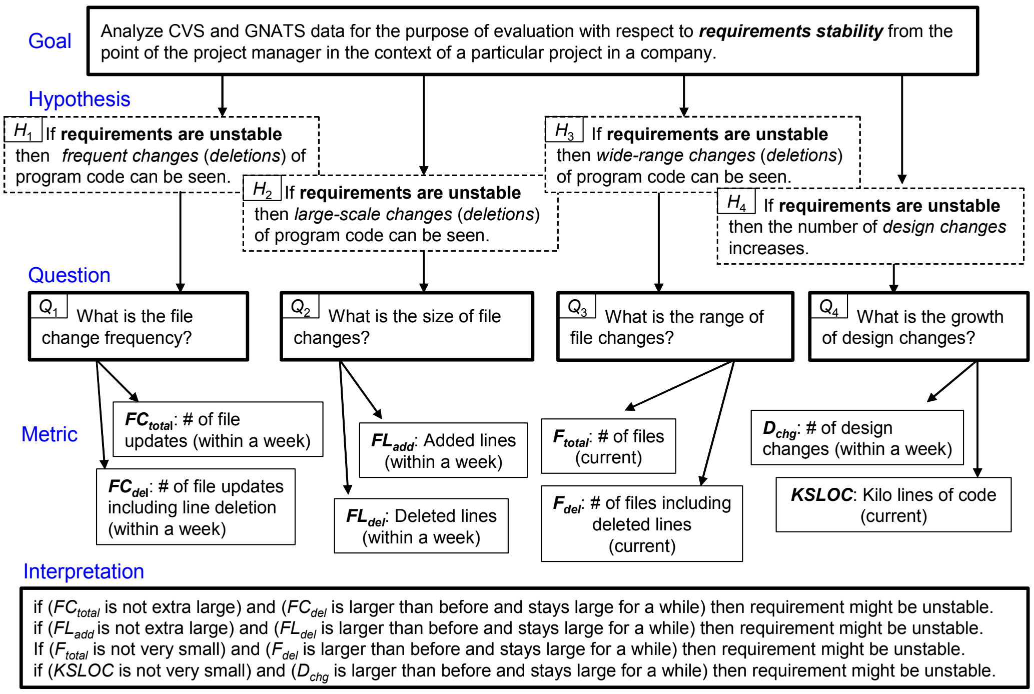
Goal
Analyze CVS and GNATS data for the purpose of evaluation with respect to requirements stability from the point of the project manager in the context of a particular project in a company.
Hypothesis
H1 If requirements are unstable then frequent changes (deletions) of program code can be seen.
H2 If requirements are unstable then large-scale changes (deletions) of program code can be seen.
H3 If requirements are unstable then wide-range changes (deletions) of program code can be seen.
H4 If requirements are unstable then the number of design changes increases.
Question
Q1 What is the file change frequency?
Q2 What is the size of file changes?
Q3 What is the range of file changes?
Q4 What is the growth of design changes?
Metric
FCtotal: # of file updates (within a week)
FCdel: # of file updates including line deletion (within a week)
FLadd: Added lines (within a week)
FLdel: Deleted lines (within a week)
Ftotal: # of files (current)
Fdel: # of files including deleted lines (current)
Dchg: # of design changes (within a week)
KSLOC: Kilo lines of code (current)
Interpretation
if (FCtotal is not extra large) and (FCdel is larger than before and stays large for a while) then requirement might be unstable.
if (FLadd is not extra large) and (FLdel is larger than before and stays large for a while) then requirement might be unstable.
If (Ftotal is not very small) and (Fdel is larger than before and stays large for a while) then requirement might be unstable.
if (KSLOC is not very small) and (Dchg is larger than before and stays large for a while) then requirement might be unstable.

Fig 8. Refined GQM model 1 for evaluating requirements stability

and "effect (e.g. frequent change of source code)."

Our explicit hypothesis is a cross between the assumptions concept in GQM+Strategies [5] and the variation factor in the GQM abstraction sheet [3][8]. The GQM+Strategies assumptions are estimated unknowns affecting the interpretation of the data. The abstraction sheet was used in the initial stage of model building to resolve conflicts and inconsistencies among participants of a GQM building team. The variation factors are environmental factors that have an impact on the quality focus (question) of a particular goal [8]. In the abstraction sheet, an assumed relationship between variation factors and the quality focus is described as a hypotheses (for example, "an increase in variation factor VF1 will reduce quality focus QF1".) By looking at the hypotheses, a participant can recognize the gap among involved people.

Requirement R4: Using base metrics instead of derived metrics

FCL (file change level) defined in Section 4.3.2 is a "derived" metric calculated from two base metrics "the number of file modifications" and "the number of total files." We found that, when we focus on the change of values, such derived metrics are not easy to interpret. For example, in Figure 6 and 7, it is difficult to figure out whether file modifications are considered too frequent or not in these projects from *FCL* values only since *FCL* is dependent on the current number of files. In addition, *FCL* is too much sensitive in the beginning and becomes less responsive later on because its denominator (# of files) is small in the beginning and increases as the project progresses. This problem also occurs in other density metrics, e.g. bug density, although these are useful to evaluate the entire project. Therefore, we decided to use base metrics instead of using derived metrics.

Nevertheless, for file-based or day-based metrics, we still need to use derived (averaged) metrics, e.g. the average number of file owners per file and the average days a bug remains open, to represent project status.

Requirement R5: Using week-wise metrics

In Figure 6 and 7, x-axes are check-in numbers in CVS repositories; however, from the perspective of project management, these x-axes should be elapsed days or dates of a project. While daily updates of metrics values are not always available, we decided to use week-wise metrics to capture any anomaly between weeks. For example, the number of bugs found could be measured within each week so that comparison between weeks becomes feasible.

But still, for some basic metrics, e.g. SLOC, we should record cumulatively every day throughout a project to visualize the progress of a project.

5.2 Refined GQM Models

Based on requirement R1, we defined our high level goal as follows.

Analyze EPM data in order to capture any symptoms of project delay from the point of the project manager in the context of a particular project in a company.

Next we identified four root causes (requirement instability, design incompleteness, bad resource allocation and bad coding quality) that might result in the

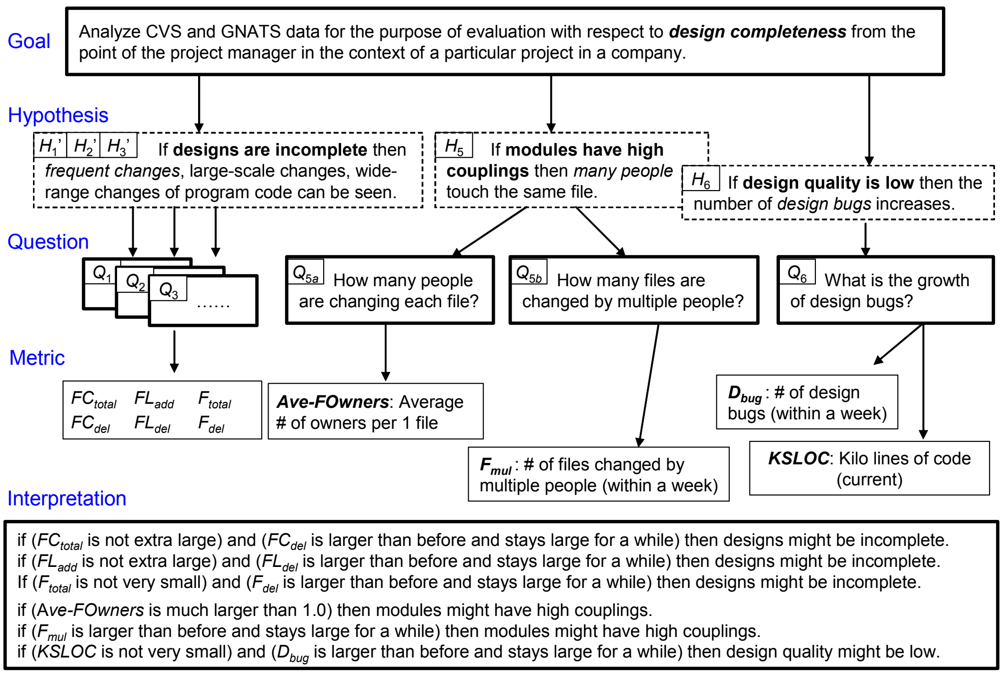
Goal
Analyze CVS and GNATS data for the purpose of evaluation with respect to design completeness from the point of the project manager in the context of a particular project in a company.
Hypothesis
H1' H2' H3' If designs are incomplete then frequent changes, large-scale changes, wide-range changes of program code can be seen.
H5 If modules have high couplings then many people touch the same file.
H6 If design quality is low then the number of design bugs increases.
Question
Q1 Q2 Q3 ……
Q5a How many people are changing each file?
Q5b How many files are changed by multiple people?
Q6 What is the growth of design bugs?
Metric
FCtotal FLadd Ftotal FCdel FLdel Fdel
Ave-FOwners: Average # of owners per 1 file
Fmul : # of files changed by multiple people (within a week)
Dbug : # of design bugs (within a week)
KSLOC: Kilo lines of code (current)
Interpretation
if (FCtotal is not extra large) and (FCdel is larger than before and stays large for a while) then designs might be incomplete.
if (FLadd is not extra large) and (FLdel is larger than before and stays large for a while) then designs might be incomplete.
If (Ftotal is not very small) and (Fdel is larger than before and stays large for a while) then designs might be incomplete.
if (Ave-FOwners is much larger than 1.0) then modules might have high couplings.
if (Fmul is larger than before and stays large for a while) then modules might have high couplings.
if (KSLOC is not very small) and (Dbug is larger than before and stays large for a while) then design quality might be low.

Fig 9. Refined GQM model 2 for evaluating design completeness

project delay. We defined four subgoals each focusing on one of these causes.

Then, for each subgoal, we built a new GQM model that refines the initial GQM models based on requirement R2 to R5.

Model 1: Evaluation of requirements stability

Figure 8 shows the first GQM model. The goal focus is "evaluation of requirements stability" in this model. Note that "unstable requirements" does *not* always mean "project is in trouble" because user requirements are essentially unstable in many projects (e.g. prototype based development). An important thing here is to *be aware of* the requirements instability in the ongoing software project based on empirical data.

In this model, we introduced a new layer "Hypothesis," which interconnects "Goal" and "Question" layer (Requirement R2). Also, "Proposed Model" layer was replaced with "Interpretation" layer, which describes a less concrete model without baseline values.

In the Hypothesis layer, we defined 4 hypotheses each associated with one question. We expect these hypotheses will help EPM user companies to understand the underlying concept of model construction, and to recognize the necessity of collecting metrics. To clarify the assumed cause-effect relations, all the hypotheses were written in the form of "if cause, then effect." Hypotheses H_1, H_2 and H_3 are related to configuration management data (CVS), assuming that frequent changes, large scale changes and wide range changes of program code can be seen if requirements are unstable. Hypothesis H_4 is related to issue tracking data (GNATS), assuming that frequent changes of designs can be seen if requirements are unstable.

In the "Metric" layer, we refined the metrics definitions based on new questions. To evaluate the "change" of program code with respect to requirement instability, we considered that "deletion" of existing code is the change. This is because programmers inevitably delete lines when a change of requirements occurs in already written source lines (note that CVS considers a "change" as a delete and add). On the other hand, "addition" of lines is not a good indicator for requirement changes since addition occurs all the time in daily programming even though requirements are not changed.

For each question, one of more base metrics were defined based on Requirement R4. For example, for the question "what is the file change frequency?" two metrics FC_{total}, (the number of file updates) and FC_{del}, (the number of file updates including line deletion) were defined. These metrics are supposed to be calculated every week (Requirement R5) and shown as a time series graph to a project manager.

Finally, in the Interpretation layer, we defined how to interpret these metrics without using baseline values. For example, if FL_{del} (deleted lines) is relatively larger than before and they do not decrease for a while, we might be concerned about requirement instability. Nevertheless, if FL_{add} (added lines) is much larger than FL_{del}, we may not need to care about FL_{del}. Therefore, we defined one of the interpretations as "if (FL_{add} is not extra high) and (FL_{del} is larger than before and stays large for a while) then requirement might be unstable."

Model 2: Evaluation of design completeness

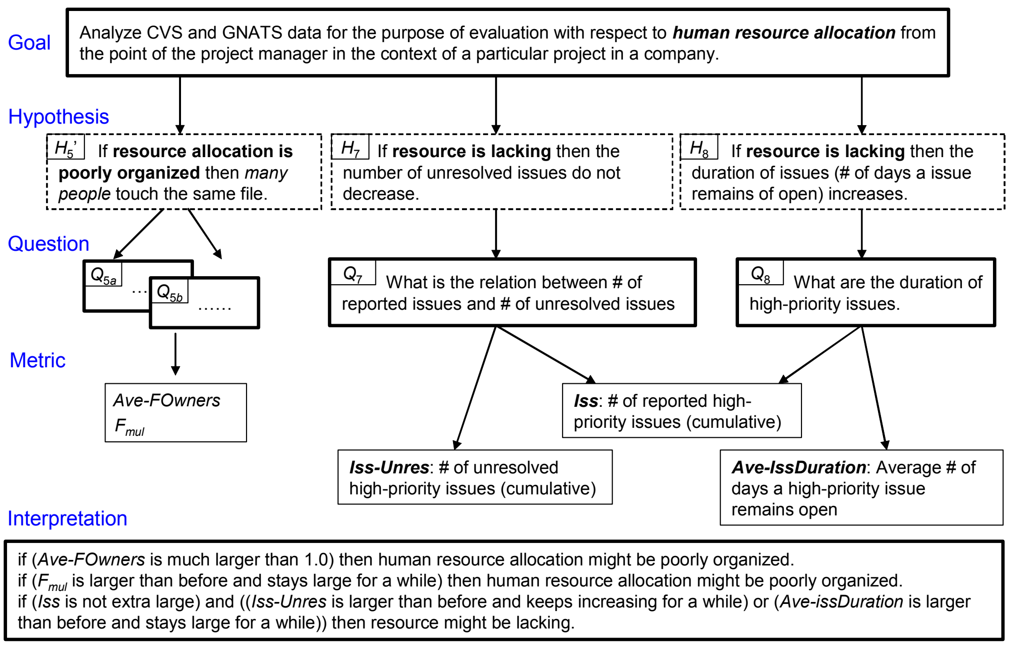
Goal
Analyze CVS and GNATS data for the purpose of evaluation with respect to human resource allocation from the point of the project manager in the context of a particular project in a company.
Hypothesis
H5' If resource allocation is poorly organized then many people touch the same file.
H7 If resource is lacking then the number of unresolved issues do not decrease.
H8 If resource is lacking then the duration of issues (# of days a issue remains of open) increases.
Question
Q5a …
Q5b ……
Q7 What is the relation between # of reported issues and # of unresolved issues
Q8 What are the duration of high-priority issues.
Metric
Ave-FOwners
Fmul
Iss: # of reported high-priority issues (cumulative)
Iss-Unres: # of unresolved high-priority issues (cumulative)
Ave-IssDuration: Average # of days a high-priority issue remains open
Interpretation
if (Ave-FOwners is much larger than 1.0) then human resource allocation might be poorly organized.
if (Fmul is larger than before and stays large for a while) then human resource allocation might be poorly organized.
if (Iss is not extra large) and ((Iss-Unres is larger than before and keeps increasing for a while) or (Ave-issDuration is larger than before and stays large for a while)) then resource might be lacking.

Fig 10. Refined GQM model 3 for evaluating resource allocation

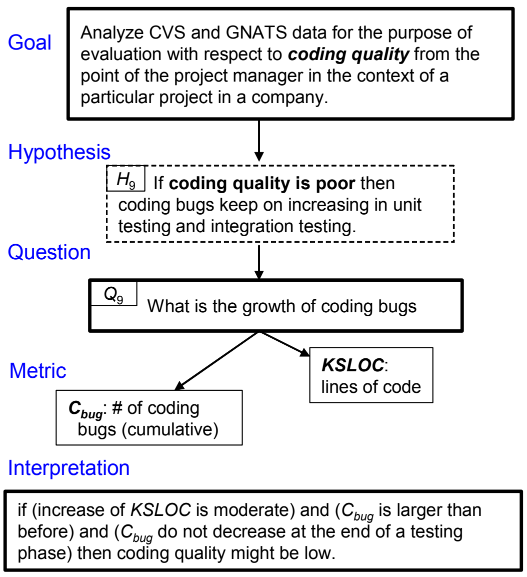
Goal
Analyze CVS and GNATS data for the purpose of evaluation with respect to coding quality from the point of the project manager in the context of a particular project in a company.
Hypothesis
H9 If coding quality is poor then coding bugs keep on increasing in unit testing and integration testing.
Question
Q9 What is the growth of coding bugs
Metric
KSLOC: lines of code
Cbug: # of coding bugs (cumulative)
Interpretation
if (increase of KSLOC is moderate) and (Cbug is larger than before) and (Cbug do not decrease at the end of a testing phase) then coding quality might be low.

Fig 11. Refined GQM model 4 for evaluating coding quality

Figure 9 shows a GQM model for evaluating design completeness. Hypotheses H_1', H_2' and H_3', and questions Q_1, Q_2 and Q_3 were borrowed from Model 1, by replacing "requirement" with "design". We assume that requirements stability and design stability both correlate with the stability of (already written) program code.

Hypothesis H_5 — many people touch the same file if modules have high couplings — is a new viewpoint of this model. It is based on an idea that more than two people need to change one module if the module is dependent on many other modules owned by different people. If many such modules exist, it indicates that the module designs are poor.

For hypothesis H_5, there are two questions Q_{5a} and Q_{5b}. Question Q_{5a} focuses on (the number of) people who changed each file, while question Q_{5b} focuses on (the number of) files changed by multiple people. For these questions, two metrics *Ave-FOwnner* and F_{mul} were defined in the model. These metrics are easily measured from CVS data since CVS records the *event owner* who checked-in a particular file.

The last hypothesis H_6 focuses on design quality. It is directly measured as the number of (reported) design bugs D_{bug} from GNATS data.

Model 3: Evaluation of resource allocation

Figure 10 shows a GQM model for evaluating human resource allocation. Hypothesis H_5' and its related questions Q_{5a} and Q_{5b} were borrowed from Model 2. In Model 2, we hypothesized that multiple owners touching one module can be observed if the module designs are poor. In Model 3, we assume it can also be observed if human allocation is confusing (or uncontrolled).

Hypotheses H_7 and H_8 are related to the issue tracking data (GNATS). We assume both the number of unresolved issues and the (average) duration of unresolved issues increases if human resources are lacking.

Model 4: Evaluation of coding quality

From the EPM user company survey, there exist projects whose coding qualities are poor and testing becomes endless. Figure 11 shows a GQM model for evaluating coding quality. In this model we describe a simple hypothesis "coding bugs keep on increasing if

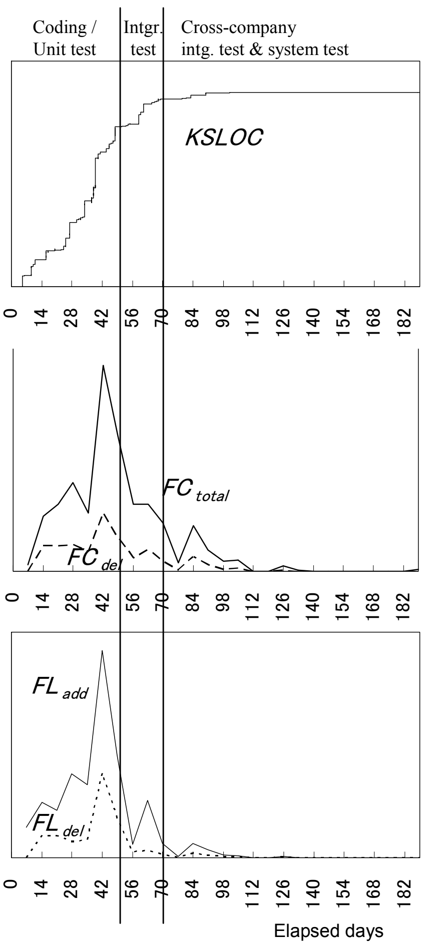

Fig 12. Metrics related to GQM Model 1 and 2 (Project A)

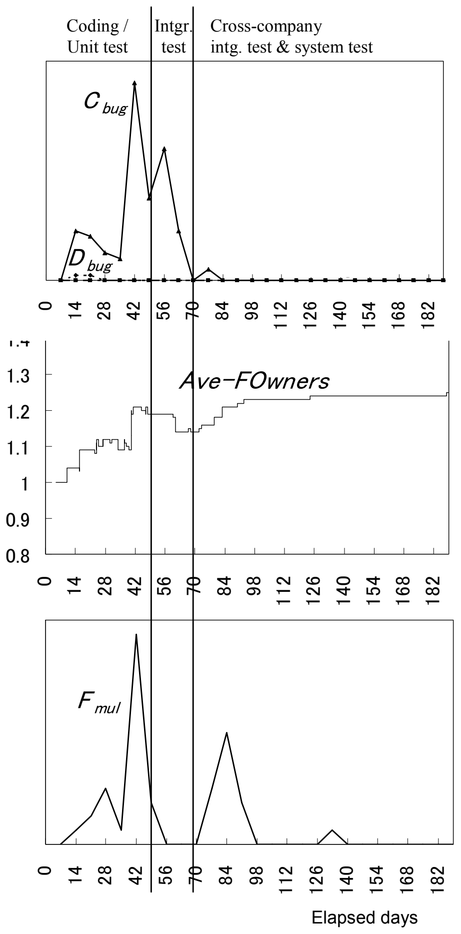

Fig 13. Metrics related to GQM Model 2, 3 and 4 (Project A)

coding quality is poor."

The model highlights both the number of coding bugs C_{bug} and *KSLOC* of source files. Note that measuring C_{bug} alone is not sufficient because in some projects, unit testing for a particular module (a set of files) starts even if other modules are still under construction. In such cases, we need to be aware of not only the growth of C_{bug} but also *KSLOC*.

Other models

Besides the four models described above, traditional GQM models related to design review quality and testing quality are also available [2][17] using EPM data. While our four models focus on the changes of time-series data to detect the symptoms of troubles in daily activities, these quality models focus on a phase-wise evaluation (e.g. design review phase).

6. Applying GQM Models to Industrial Projects

6.1 Project descriptions

This paper targets a multi-vendor development of an information system, carried out by members of the COSE[1] with the support of Japan's Ministry of Economy, Trade and Industry (METI). Six COSE companies participated in the development: one company was engaged with project management, and the other five with development. Development was carried out using a waterfall process. A user company defined requirements, and development companies each developed subsystems under the supervision of a project-management company. After each company conducted an intra-company unit test and integration test, an inter-company integration test was performed, followed by an inter-company system test.

The size of the developed system was approximately 330K steps (SLOC), and almost all of the source code was written in C/C++ language. The number of subsystems was 38 and the number of files including shell script, batch, etc. was approximately 1400. The project duration was ten months.

Researchers from the EASE[11] and the SEC[2] created a data-collection scheme using the EPM and analyzed

[1] COSE : COnsortium for Software Engineering

[2] SEC: Software Engineering Center, Information-technology Promotion Agency, http://sec.ipa.go.jp/

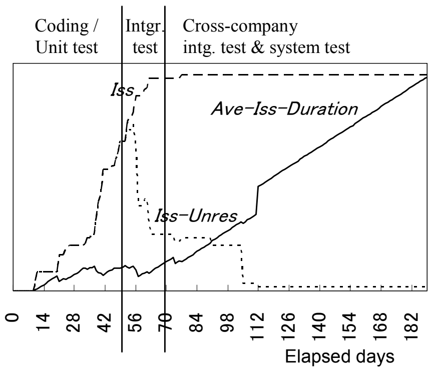

Fig 14. Metrics related to GQM Model 3 (Project A)

the obtained data based on GQM models. From the beginning of the project, meetings were held with each company to give feedback on analysis results and to conduct interviews with developers and a project manager. Data were collected from coding through system testing.

Problem symptoms were identified every week by the GQM analysis team (of the EASE project) based on the refined GQM models, and reported to project managers in the feedback meeting. The managers judged whether the reported symptoms are actually problems or not, and conducted remedial actions or process improvement as needed.

This paper describes data analysis of two development processes (namely, project A and B.) One limitation in applying EPM was that, in project B, EPM was used after the coding phase. That is, each file was checked-in to CVS individually only when it became ready for the unit test. Hence, growth of KSLOC in project B does not directly reflect the progress in file size. On the other hand, project A used EPM from the beginning of the coding phase. Note that there was no clear date line between coding and unit test in both projects.

For these projects, we had feedback meetings to show the result of GQM analyses and to get feedback from companies. These meetings helped us to relate what happened in projects and what was observed in collected data.

6.2 Analysis of Project A

6.2.1 GQM Model 1

Figure 12 shows some of the requirement stability metrics (which are also design completeness metrics) of project A. Design changes were not observed in this project. As we see the changes of four metrics values FC_{total}, FC_{del}, FL_{add} and FL_{del}, this project did not match the expressions in the interpretation layer of GQM Model 1. Therefore, there was no symptoms for

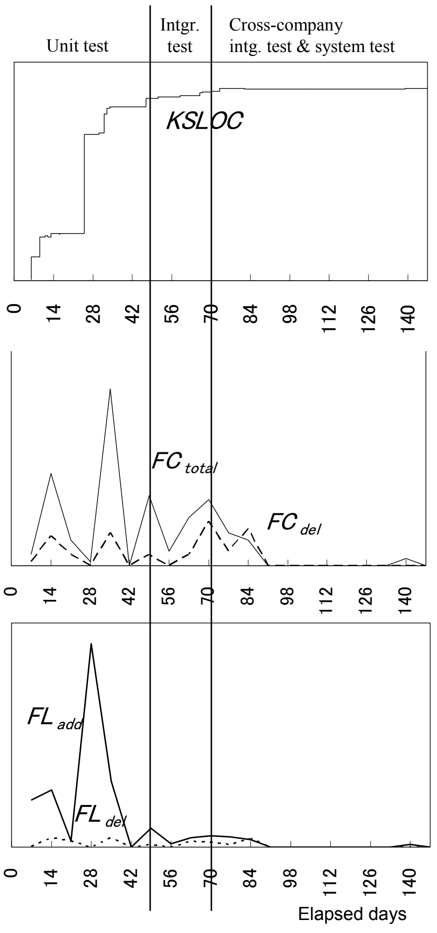

Fig 15. Metrics related to GQM Model 1 and 2 (Project B)

requirement instability (and design incompleteness) observed in this project. Although both FC_{del} and FL_{del} grew large just before the (intra-company) integration test (around 42nd day), FC_{total}, and FL_{add} also went very large; and all four metrics went very small in the integration test and cross-company tests. This indicates that unit tests were quite actively executed and successfully finished.

One remarkable observation in this project was that deletions of code lines frequently occurred throughout the project as FC_{del} and FL_{del} covered about one-third of FC_{total} and FL_{add} respectively. An interview with the engineers indicates that a static code-analysis tool was regularly used and changes to source code were made based on the tool in this company.

6.2.2 GQM Model 2 and 4

Figure 13 shows design completeness metrics and coding quality metrics of project A. As shown in the Figure, a few design bugs were reported. On the other hand, just before the integration test, *Ave-FOwners* became around 1.2 and F_{mul} became large (although it soon went small). These could be the suspected symptom of design incompleteness from model-based interpretation; however, an interview with the engineers indicated that there was a change of an engineer at this time. In addition, F_{mul} also became large in around 84th day; and, it turned out that a certain engineer made code clean up (mostly revising comments in source code) at this time. Therefore, no serious design problem was present in this project.

As for coding quality (GQM Model 4), unit test and integration test phases found several coding bugs. However, bugs were soon reduced to zero in the end of the integration test (Figure 13) and FL_{del} was small in this phase (Figure 12); thus, there was no serious coding problem.

6.2.3 GQM Model 3

Figure 14 shows resource metrics of project A. Notably, *Iss* and *Iss-Unres* had the same value until around the 56th day. This means, in general, all the reported issues (bugs) were not resolved (fixed) at all for more than a month. However, an interview with the engineers indicated that issues were mostly resolved soon but not recorded in EPM until around the 56th day.

Another anomaly was that *Ave-IssDuration* kept increasing after the integration test because one issue remained open until the end of the project. This was also due to the recording problem (i.e. resolved but not recorded).

As a result, the resource allocation problem itself was not properly monitored, but recording problems were clearly observed in this project.

6.3 Analysis of Project B

6.3.1 GQM Model 1

Figure 15 shows requirement stability metrics (which are also design completeness metrics) of project B. Design changes were also not observed in this project. Notably, FC_{del} became larger in the integration test and cross-company tests than in the unit test, which could be the suspected symptom of requirement instability and/or design incompleteness from model-based interpretation. The large FC_{del} indicates that broad range modifications were made to software after the integration. In this project, requirements were stable, but design incompleteness was revealed after integration (see next Section 6.3.2) and this made the increase of FC_{del}.

6.3.2 GQM Model 2 and 4

Figure 16 shows design completeness metrics and coding quality metrics of project B. Obviously, a lot of design bugs and coding bugs were reported in the integration test, which suggested both design incompleteness and low coding quality. It turned out from an interview that design bugs were due to insufficient design review; and, many of coding bugs were due to incomplete bug fix in the unit test and the integration test. A symptom of design incompleteness was also seen in the previous phase (unit test), i.e. design bugs were found throughout the unit test, while in project A, design bugs were found only in the beginning of the unit test.

Another anomaly was seen around the 120th day where one design bug was reported. This was related to a performance problem of a software system.

Increase of *Ave-FOwners* and F_{mul} was also observed in this project around the 35th and 70th days. These were due to a change of engineers just as in project A. It can be said that F_{mul} is a good indicator to capture the change of engineers.

6.3.3 GQM Model 3

Figure 17 shows resource metrics of project B. Notably, *Iss-Unres* did not decrease in the cross-company tests; and, this made *Ave-Iss-Duration* keep growing. These strongly suggest the presence of resource allocation problems from the model-based interpretation. From an interview, we found that unresolved issues were related to the performance problem, which was not easy to fix. Also, it turned out that the delay of bug fix was partly because an engineer was changed around the 70th day as shown in Figure 16. Therefore, we could conclude that the resource allocation problem was properly captured by GQM model 3.

6.4 Discussion

Our main finding is that our less concrete models could successfully identify typical problems in multiple projects, that is, the models are potentially reusable in many companies that use EPM environment.

Although not all the changes of metrics values were directly related to GQM goals, we did find several symptoms of project problems as follows:

(1) In project B, design incompleteness was observed from D_{bug} and FC_{del} of GQM model 2.
(2) In project B, resource allocation problem was observed from *Ave-Iss-Duration*, *Iss-Unres* and F_{mul} of GQM model 3.

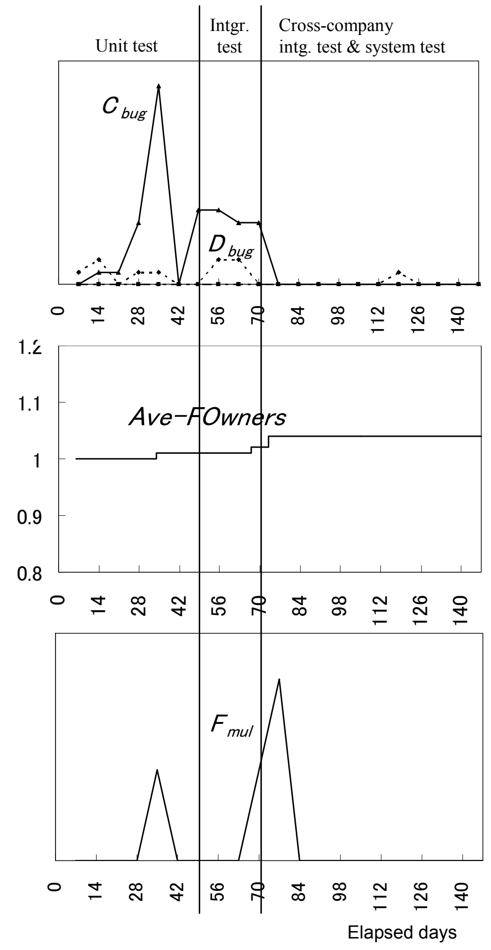

Fig 16. Metrics related to GQM Model 2, 3 and 4 (Project B)

(3) In project B, low coding quality was observed from C_{bug} of GQM model 4.

(4) In project A, resource allocation problem was suspected from *Ave-Iss-Duration* and *Iss-Unres*, but it turned out that this was due to the improper use of EPM (GNATS).

In the feedback meetings, some of project managers told us that they were already aware of issues identified by our GQM-based analysis. However, they also told us it was very useful to quantitatively confirm what they have observed by the models. Managers also suggested that we conduct a module-based analysis because problems are often bound to a particular module. The module-based analysis will be an important future refinement.

The hypotheses were useful to explain engineers how goals are related to questions and metrics. We could clearly explain the "cause-effect" relations we assumed.

During GQM-based analyses, we found some mismatches between GNATS data and CVS data. For example, there was a closed bug in GNATS data but there was no history of fixing the bug in CVS comments.

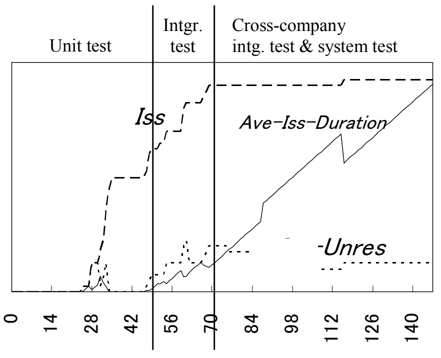

Fig 17. Metrics related to GQM Model 3 (Project B)

Automated detection of such mismatches would be useful to engineers.

We conducted interviews with managers in succeeding projects next year using our less concrete GQM models for different goal focuses. From the interviews, we found that 17 out of 28 reported problem symptoms were actually considered problems; and, the companies were not aware of 7 problems. Therefore, we confirmed that the idea of less concrete models were actually useful to control the projects. On the other hand, 11 problem symptoms were reported but they were not problems indeed. Since we use less concrete interpretation model, it is natural that some interpretation does not fit the project context, and managers could successfully judge whether a symptom fits their context or not. Therefore, we consider that mangers can make decision based on less concrete models.

7. Conclusion

The goal of this paper is to customize GQM models for a particular environment where a software project monitoring tool EPM is used, which means measurement context is clear while variations among projects are still allowed. Our basic idea is to build less concrete models that do not include explicit baseline values to interpret metrics values, while they include a "hypothesis layer" and an "interpretation layer" to help people of different projects make decisions in their own context.

We built initial GQM models applicable to EPM user companies. Then we refined the models into 4 new models based on 5 requirements, 1) narrowing goal focus, 2) explicit hypothesis description, 3) building less concrete models, 4) using base metrics instead of derived metrics 5) using week-wise metrics, which are derived from an analysis based on initial GQM models.

We applied the models to two industrial projects (A and B). Our main finding is that our less concrete models could successfully identify typical problems in multiple projects, that is, the models are potentially reusable in many companies that use EPM environment.

These case studies of GQM model-based analyses suggest that models are useful to quantitatively confirm what the project managers have observed. We could identify problems such as (1) design incompleteness in project B, (2) resource allocation problem in project B, (3) low coding quality in project B, and (4) improper use of EPM in project A.

These case studies and the development of the GQM models using EPM-collected data indicate the usefulness of an empirically-based process for modeling, observing, and providing real-time feedback to industrial projects. This allows project managers to support their intuitive observations about problems with factual data, reducing the risks associated with these problems. Since EPM collects data from commonly used software development tools, this approach does not add additional data collection efforts even though it provides significantly better understanding of the projects. This means project managers and others can more easily and accurately manage their projects.

Acknowledgments

The part of this work was conducted in the EASE project, the Comprehensive Development of e-Society Foundation Software program of the Ministry of Education, Culture, Sports, Science and Technology. Also, part of this work was conducted under Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (C) (22500028).

References

[1] Basili, V. R. and Weiss, D. M., "A methodology for collecting valid software engineering data," *IEEE Trans. on Software Engineering*, Vol.SE-10, No.6, pp.728-838, 1984.

[2] Basili, V. R., "Software modeling and measurement: the Goal/Question/Metric paradigm," *Computer Science Technical Report Series*, CS-TR-2956 (UMIACS-TR-92-96), University of Maryland, College Park, MD, Sep. 1992.http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/84.73.pdf

[3] Basili, V., Caldiera, G., and Rombach, D., "Goal question metric paradigm," In J. Marciniak, editor, Encyclopedia of Software Engineering, Volume 1, pp.528–532. John Wiley & Sons, Inc., New York, 1994.

[4] Basili, V. R., and Caldiera, G., "Improve software quality by reusing knowledge and experience," *Sloan Management Review*, Vol.37, No.1, pp.55-64, 1995.

[5] Basili, V. R., Lindvall, M., Regardie, M., Seaman, C., Heidrich, J., Münch, J., Rombach, D., and Trendowicz, A., Linking software development and business strategy through measurement, IEEE Computer, Vol.43, No. 4, pp.57-65, 2010.

[6] Briand, L. C., Morasca, S., and Basili, V. R., "An operational process for goal-driven definition of measures," *IEEE Trans. on Software Engineering*, Vol.28, No.12, pp.1106-1125, 2002.

[7] Cusumano, M., MacCormack, A., Kemerer, C. F., and Crandall, B., "Software development worldwide: the state of the practice," *IEEE Software*, Vol.20, No.3, pp.28-34, 2003.

[8] Dybå, T., Stålhane T., and Palmstrøm R., "Experience of goal-oriented measurement using ami and GQM," In Proc. 8th European Software Control and Metrics Conference (ESCOM'97), 1997.

[9] EPM Enhanced Version, Information-technology Promotion Agency, http://sec.ipa.go.jp/tool/epm.html

[10] EPM Online Manual, http://empirical.jp/EPM/EPM.html

[11] The EASE Project, http://www.empirical.jp

[12] Mendonca, M. G., Basili, V. R., Bhandari, I. S., and Dawson, J., "An approach to improving existing measurement frameworks," *IBM Systems Journal*, Vol.37, No.4, pp.484-501, 1998.

[13] Mitani, Y., Barker, M., Torii, K., and Tsuruho, S., "An experimental framework for Japanese academic-industry collaboration in empirical software engineering research," *Proc. Int'l Symposium on Empirical Software Engineering (ISESE2004)*, Vol.2, Aug. 2004.

[14] M. Ohira, R. Yokomori, M. Sakai, K. Matsumoto, K. Inoue, M. Barker, and K. Torii, "Empirical Project Monitor: a system for managing software development projects in real time," *Proc. Int'l Symposium on Empirical Software Engineering (ISESE2004)*, Vol.2, pp.37-38, Aug. 2004.

[15] Ohira, M., Yokomori, R., Sakai, M., Matsumoto, K., Inoue, K., and Torii, K., "Empirical Project Monitor: A tool for mining multiple project data," *Proc. Int'l Workshop on Mining Software Repositories (MSR2004)*, pp.42-46, May. 2004.

[16] Oka, Y., Tamura, K., Sekiguchi, J., Suzuki, K., Murayama, T., Tsunoda, F., Tamori, T., "Advantages and disadvantages of "mieruka" confirmed by the EPM evaluation working group of Japan Information Technology Services Industry Association," SEC journal, Vol.5, No. 6, pp.355-361, Dec. 2009.

[17] Solingen, R. and Berghout, E., "The Goal/Question/Metric method – A practical guide for quality improvement of software development," McGraw-Hill, 1999.

[18] Trendowicz, A., Heidrich, J. and Shintani, K., "Aligning software projects with business objectives," Proc. Joint Conference of 21st Int'l Workshop on Software Measurement and 6th Int'l Conference on Software Process and Product Measurement (IWSM/MENSURA2011), pp.142-150, Nov. 2011.

Appendix A: Metrics collected by EPM

	Configuration Management Data (CVS)		Mailing List Management Data (Mailman)	Bug Tracking Management Data (GNATS)
	Process	Product		
PROJECT ID	host name:project name (e.g. se.naist.jp:EASE_Project)			
PROJECT NAME	project name (e.g. EASE_Project)			
LACATION	host name:input file name or directory name (e.g. se.naist.jp:/tmp/cvsroot)			
EVENT TYPE	CVS	PRODUCT	MAIL	how to fix an issue)
EVENT SOURCE	person who operated CVS	person who created a XML file	line number or file name in MailBox	bug report file name in GNATS
EVENT OWNER	person who operated CVS	person who created a XML file	sender's e-mail address	bug reporter
EVENT TIME	time when CVS was operated	time when a XML file was created	time when an email was sent	time when a bug report was posted
EVNET TARGET	file name or directory name	project name	mail subject excluding "Re:"	category (e.g. pending, bug, documentation, feature, patch)
EVENT DETAIL	event type (e.g. checkout, export, add, modify, remove, tag, release, update, delete,	file name	message ID (in mail header)	bug type (e.g software-bug, documentation-bug, support, change-request, mistaken, duplicate)
	version		references (in mail header)	bug state (e.g open, analyzed, suspended, feedback,closed)
	remote working directory			severity (e.g critical, serious, non-critical)
			original subject	prioritry (e.g. high, medium,low)
	module name	version		person in charge of the bug
				bug reporter's name
			lines of message excluding mail header	bug reporter's address
				reporter's type (e.g User, Developer,
	number of added lines			mail address list for notification
			sender's name	time when a bug issue raised
		lines of code (in the file)		time when a bug report last modified
	number of deleted lines			time when a bug report was closed
			sender's e-mail address	subject of bug report
				bug report number
	change log message		receiver's name	synopsis (summary of a bug)
				bug description (in detail)
		time when the file was modified	receiver's e-mail address	how to reproduce an issue
	cvs tag			how to fix an issue
			cc receiver's name	confidental (e.g. yes or no)
				report modification history
	sticky (additional information)		cc receiver's e-mail address	release information
				related data