
Interner Bericht

**Measurement-Based Feedback in a
Process-Centered Software Engineering Environment**

Christopher M. Lott

283/96

Fachbereich Informatik

Universität Kaiserslautern • Postfach 3049 • D-67653 Kaiserslautern

Measurement-Based Feedback in a Process-Centered Software Engineering Environment

Christopher M. Lott

Internal Report 283/96

Herausgeber: AG Software Engineering
Leiter: Prof. Dr. H. Dieter Rombach

Kaiserslautern, Mai 1996

Abstract

Software development organizations measure their real-world processes, products, and resources to achieve the goal of improving their practices. Accurate and useful measurement relies on explicit models of the real-world processes, products, and resources. These explicit models assist with planning measurement, interpreting data, and assisting developers with their work. However, little work has been done on the joint use of measurement and process technologies.

We hypothesize that it is possible to integrate measurement and process technologies in a way that supports automation of measurement-based feedback. Automated support for measurement-based feedback means that software developers and maintainers are provided with on-line, detailed information about their work. This type of automated support is expected to help software professionals gain intellectual control over their software projects.

The dissertation offers three major contributions. First, an integrated measurement and process modeling framework was constructed. This framework establishes the necessary foundation for integrating measurement and process technologies in a way that will permit automation. Second, a process-centered software engineering environment was developed to support measurement-based feedback. This system provides personnel with information about the tasks expected of them based on an integrated set of measurement and process views. Third, a set of assumptions and requirements about that system were examined in a controlled experiment. The experiment compared the use of different levels of automation to evaluate the acceptance and effectiveness of measurement-based feedback.

Acknowledgements

I would like to thank my dissertation advisor, Victor R. Basili, for offering me the chance to work with one of the pioneers in empirical research in software engineering. I greatly enjoyed my years at Maryland, and was especially glad to have cooperated with members of the Software Engineering Laboratory at NASA's Goddard Space Flight Center.

Next I would like to thank H. Dieter Rombach for the opportunity to work with him, the Research Group in Software Engineering, the Software Technology Transfer Initiative, and the Fraunhofer Institute for Experimental Software Engineering in Kaiserslautern. The environment at Universität Kaiserslautern made this research possible, and also improved my German skills considerably.

I owe a great debt to my students at Universität Kaiserslautern for their ideas and assistance with defining, designing, implementing, and repeatedly extending the process engine and the user interface for MVP-S. In alphabetical order they are: Bernd Freimut, Oliver Flege, Ralf Kempkens, Barbara Hoisl, Enno Tolzmann, and Jürgen Wüst. Further, I would especially like to thank my colleagues and their students who participated in the experimental evaluation of MVP-S.

Portions of this dissertation were previously published in journals and conference proceedings, and the copyright was transferred accordingly. In alphabetical order, I thank Elsevier Science Ltd., IEEE, Springer-Verlag, and World Scientific Publishing for their permission to republish work of mine to which they hold the copyright.

Thanks to all the people who made my life better over the years in Maryland and Kaiserslautern. Bill Regli was my partner in throwing aerobic, making "drano" salsa, skiing the Poconos, traveling the world, biking around Washington, and sharing some fantastic food. Martin Verlage smoothed the rocky road of starting anew in a foreign country and made some great dinners on the grill. Alfred Bröckers answered a thousand questions about teaching, the university, statistics, German grammar, and also showed us the finer parts of life in Germany, including good beer, conversation, and hospitality.

Last but certainly not least, I would like to thank my wife, Kristen, and our parents. They encouraged me to embark on this course of study, patiently supported me through good times and bad, and provided badly needed encouragement when my motivation failed. Tausend Dank!

Contents

1	Introduction	1
1.1	Context of the Dissertation	1
1.2	Statement of the Problem	2
1.3	Contributions	3
1.4	Integration of Measurement and Process	3
1.4.1	Example 1: Describing the status quo	4
1.4.2	Example 2: Improving existing practices	6
1.5	Automation of Measurement-Based Feedback	7
1.5.1	Level 1: No explicit process nor measurement view	8
1.5.2	Level 2: Explicit process view and measurement view	8
1.5.3	Level 3a: On-line process view, off-line measurement view	10
1.5.4	Level 3b: Off-Line process view, on-line measurement view	11
1.5.5	Level 4: Process-centered environment	12
1.6	How to Read the Dissertation	13
2	Survey of Related Work	15
2.1	Overview	15
2.2	Measurement Frameworks	16
2.2.1	Quantitative Measurement of Software Quality	17
2.2.2	Software Quality Characteristics	17
2.2.3	Factors in Software Quality	18
2.2.4	Software Quality Metrics	18
2.2.5	Goal Question Metric Paradigm	19
2.2.6	Software Quality Attributes	20
2.2.7	Quality Function Deployment	21
2.2.8	Summary of measurement frameworks	22
2.3	Process Modeling Notations	22
2.3.1	Requirements-level notations	25
2.3.2	Design-level notations	26
2.3.3	Code-level notations	29
2.3.4	Summary of process modeling notations	31
2.4	Measurement Support in Process-Centered SEEs	32
2.4.1	GQM plan for the survey of process-centered SEEs	33
2.4.2	Survey of existing systems	33
2.4.3	Summary of process-centered SEEs	39
2.5	Empirical studies of software engineering environments	42
2.6	Summary	43

3	Foundations of Measurement-Based Feedback	45
3.1	Overview	45
3.2	Integration of Measurement and Process Views	46
3.2.1	Requirements for integrating measurement and process views	47
3.2.2	Reuse of existing formalisms	48
3.2.3	Information that links the measurement and process views	50
3.2.4	Limitations and open issues	53
3.2.5	Definition of consistency properties	55
3.3	Algorithm for Integrating Measurement and Process Views	58
3.3.1	QIP step 1	58
3.3.2	QIP step 2, integration task 1	58
3.3.3	QIP step 3, integration tasks 2–6	59
3.3.4	QIP step 4	61
3.3.5	QIP step 5	61
3.3.6	QIP step 6	62
3.4	Example of Developing and Integrating Views	62
3.4.1	Task 1: Start the measurement view	62
3.4.2	Task 2: Develop the process view	62
3.4.3	Task 3: Complete the GQM plan	65
3.4.4	Task 4: Integrate the views	65
3.4.5	Task 5: Check consistency of views	70
3.4.6	Task 6: Develop the measurement plan	70
3.4.7	Summary of the example	71
3.5	Evaluation of the Integration Approach	71
3.5.1	Planning an industrial measurement program	71
3.5.2	The case study	72
3.5.3	Summary of the evaluation	73
3.6	Requirements for Automated Support	73
3.6.1	User interactions for collecting data	74
3.6.2	Generating feedback for technical roles	77
3.6.3	Capabilities to support managerial roles	79
3.7	Summary	80
4	Automated Support for Measurement-Based Feedback	81
4.1	Overview	81
4.2	MVP-S, A Prototype Process-Centered SEE	82
4.2.1	Capabilities of the prototype	82
4.2.2	Possible extensions to the prototype	90
4.2.3	Example of providing measurement-based feedback	90
4.3	Assumptions About Automated Support	95
4.3.1	Summary of the MVP-S user interface	95
4.3.2	Assumptions about the use of MVP-S	96
4.4	Summary	99
5	Empirical Evaluation of Measurement-Based Feedback	101
5.1	Overview	101
5.2	Characterization of Experiments	103
5.3	Hypotheses and GQM Plan	103
5.3.1	Hypotheses	103

5.3.2	Goal	105
5.3.3	Questions	105
5.3.4	Metrics	106
5.4	Experiment Plan	107
5.4.1	Experimental design	107
5.4.2	Exercises	109
5.4.3	Instruments	110
5.4.4	Subjects	111
5.4.5	Data collection and validation procedures	112
5.4.6	Data analysis procedures	112
5.5	Experiment Procedures	112
5.5.1	Training activities	112
5.5.2	Conducting the experiment	113
5.6	Results	114
5.6.1	Data and interpretations	114
5.6.2	Discussion	124
5.7	Critique and Future Work	126
5.7.1	Improving the experiment	126
5.7.2	Possible improvements to MVP-S	127
5.7.3	Future work	128
5.8	Summary	129
6	Conclusion	131
A	Appendix: Instruments	133
A.1	Script used to introduce subjects to MVP-S	134
A.2	Instructions for “functional testing” as used off-line	135
A.3	Instructions for “functional testing” as used on-line	137
A.4	Question sheet “functional testing” as used off-line	140
A.5	Question sheet “functional testing” as used on-line	141
A.6	Building equivalence classes	142
A.7	Specification of program “cmdline”	143
A.8	Source code of program “cmdline”	145
A.9	Sample solution for “functional testing”	151
A.10	Instructions for “structural testing” as used off-line	153
A.11	Instructions for “structural testing” as used on-line	155
A.12	Question sheet “structural testing” as used off-line	157
A.13	Question sheet “structural testing” as used on-line	158
A.14	Explanation of coverage criteria	159
A.15	Specification of program “tokens”	160
A.16	Source code of program “tokens”	161
A.17	Sample solution for “structural testing”	164
A.18	Observer’s data sheet	167
A.19	Debriefing questionnaire	168
	Bibliography	171

List of Tables

2.1	Overview of measurement frameworks	23
2.2	Summary of example notations	31
2.3	Summary of software engineering environments	40
3.1	Compatability of a metric and an attribute	56
3.2	Summary of the example process view	64
3.3	Questions to describe the objects of interest (set O)	65
3.4	Questions to describe the quality aspect of interest (set Q)	65
3.5	Metrics that help answer the questions	66
3.6	Summary of the example measurement view	66
3.7	Mapping of metrics and objects/attributes for example DCT1	68
3.8	Mapping of questions and objects for example DCT1	68
3.9	Mapping of metrics, events, and personnel for example DCT1	69
3.10	Summary of the example integration information	70
5.1	Hierarchical characterization schema for software engineering experiments	104
5.2	Size measures and fault counts for the code objects	111
5.3	Basis for quantitative quality models supplied to subjects	111
5.4	Groups used to partition subjects	113
5.5	Subjective data concerning acceptance (hypothesis 1)	115
5.6	Subjective data concerning comprehension (hypothesis 2)	115
5.7	Subjective data concerning use of quantitative data (hypothesis 3)	116
5.8	Comparison of data from two experiments on exercise "FT/cmdline"	117
5.9	Data concerning reporting data in a timely way (hypothesis 4)	118
5.10	Descriptive statistics for time and failure data (hypothesis 5)	118
5.11	Results for effect of exercise (functional – structural)	120
5.12	Results for effect of guidance technique (off-line – on-line)	122
5.13	Results for effect of order (trial 1 – trial 2)	124

List of Figures

1.1	Product flow in example DCT1	5
1.2	Illustration of the real and model worlds	7
1.3	No process or measurement view (level 1)	8
1.4	Explicit, off-line process and measurement views (level 2)	9
1.5	On-line process view, off-line measurement view (level 3a)	10
1.6	Off-line process view, on-line measurement view (level 3b)	11
1.7	A process-centered software engineering environment (level 4)	12
2.1	Refinement levels in a GQM plan	20
2.2	Activities supported by process model notations	22
2.3	Basic MVP-L entities and relationships	28
2.4	MVP-L process model for "Write_design" from DCT1 example	29
2.5	Abstraction levels of work activities	32
2.6	Feedback provided by Xceilidh	38
2.7	SEE components: construction versus measurement	41
2.8	SEE components: process definition versus measurement	41
2.9	SEE components: construction versus process definition	42
3.1	Linking metrics and attributes from measurement and process views	46
3.2	MVP-L project plan for the DCT1 example	63
3.3	MVP-L product model for "Requirements_document," before integration	64
3.4	MVP-L resource model for "Developer," before integration	64
3.5	Scope of activities for a process-centered SEE	74
3.6	Forms-based data collection tool	77
4.1	Coverage of the solution space for measurement-based feedback	82
4.2	Viewing the project state in a role-specific work context window	84
4.3	Context of activity "Major Design Review"	85
4.4	Viewing the object model for process "step_major_design_review"	86
4.5	E-mail request generated by the process engine	87
4.6	Control-flow view of the ISPW activities (from [CML ⁺ 90])	91
4.7	Data-flow view of the detailed example	92
4.8	Work context for activity "Modify Design," no quantitative feedback	92
4.9	Role-specific work context for a QA engineer, no quantitative feedback	93
4.10	Work context for activity "Modify Design," with quantitative feedback	94
4.11	Role-specific work context for a QA engineer, with quantitative feedback	95
5.1	Experimental design	108
5.2	Steps in the functional-test exercise	109
5.3	Steps in the structural-test exercise	110

5.4	Raw data for total time required	119
5.5	Raw data for percentage of failures detected	119
5.6	Difference in times, FT – ST	120
5.7	Difference in percentage of failures, FT – ST	121
5.8	Difference in rate of failure detection, FT – ST	121
5.9	Difference in times, off – on	122
5.10	Difference in percentage of failures, off – on	123
5.11	Difference in rate of failure detection, off – on	123
5.12	Difference in times, trial 1 – trial 2	124
5.13	Difference in percentage of failures, trial 1 – trial 2	125
5.14	Difference in rate of failure detection, trial 1 – trial 2	125
A.1	Menu bar to start MVP-S and a question sheet tool	138
A.2	MVP-S work-context window that supports functional testers	138
A.3	MVP-S process-context window for functional testing step “schritt_1”	139
A.4	On-line question sheet used by functional testers	141
A.5	MVP-S process-context window for structural testing step “schritt_2”	156
A.6	On-line question sheet used by structural testers	158

Chapter 1

Introduction

Software in the mid 1990's involves far more than payroll systems, spread-sheet packages, and flight simulator games. Software is first and foremost a tremendous productivity tool: it helps us compose and format documents, make decisions based on data analyses, and access enormous stores of information. Thanks to its unparalleled flexibility, the use of software in embedded systems is growing rapidly. To list a few examples, software has assumed a leading role in telephone switching offices, civilian and military jet aircraft, airport baggage-handling systems, and radiation therapy machines. In brief, software has become the value-added technology that companies use to distinguish their products from the competition. As a result, society is increasingly forced to rely on the orderly and reliable functioning of software.

1.1 Context of the Dissertation

Software development and maintenance practices have not kept pace with the demand for reliable systems that meet their user's expectations [Gib94]. All of the embedded-system examples mentioned above have experienced catastrophes, some that resulted in loss of life, due to faulty software. Large projects in the civilian and military domains regularly overrun schedules and cost budgets, buyers of shrink-wrap software become *de facto* beta testers, and consumer products such as automobiles increasingly malfunction due to software defects rather than manufacturing flaws. One reason is that software development and maintenance practices resemble a craft-oriented cottage industry more than an engineering field. The lack of engineering discipline can be seen in the following problems:

- Processes are implicit.
A process is a set of partially ordered steps intended to reach a goal [FH92]. Software organizations generally have a poor understanding of the processes followed by their development and maintenance staff. Definitions of the actual processes exist only implicitly in the minds of the staff, leading to redundant work and poor communication. Further, it is impossible to perform cause-effect analyses when a project fails, and difficult to duplicate work practices when a project succeeds.
- A quantitative understanding is missing.

Few organizations possess a quantitative understanding of process and product qualities such as cost or reliability. Even the most rudimentary measures frequently recommended for processes such as the number of defects found are not regularly collected [Dem86]. Further, quality goals are seldom expressed in operational, quantitative terms.

- Projects have no predictive capabilities.

Models are not commonly used to track or predict the attainment of project goals. A model may be as simple as an equation that predicts growth in lines of code based on the percentage of schedule expended. Without predictive capabilities, software development organizations cannot make sound judgements about their projects.

- Techniques are not validated.

Vendors of new tools and techniques made wild claims such as “900% productivity improvement” but have no evidence to support their claims. Even worse, these new tools and techniques are often applied in revenue-producing projects before they can be assessed, validated, and tailored for the local context.

- History is ignored.

Past experience is more frequently discarded rather than packaged for learning and reuse. Thus software projects are condemned to repeat the same basic mistakes. Also, because so few models exist for predicting the attainment of project goals, the models cannot be improved based on information gained from new projects.

To summarize, basic principles known in other engineering fields are not being followed by the majority of software professionals. It is clear that software professionals have not attained the needed intellectual control over their development and maintenance projects.

1.2 Statement of the Problem

This dissertation addresses the first two problems from the previous list, namely implicit processes and a missing quantitative understanding. We hypothesize that it is possible to integrate measurement and process technologies in a way that allows automated support for assisting software development personnel. Assistance based on measurement and process technology is named *measurement-based guidance and feedback*. The long-term goal of automated support for measurement-based guidance and feedback is to improve communication and reduce problems among teams who cooperate on a lengthy project.

The research described in this dissertation encompasses three goals. First, a foundation for measurement-based feedback must be established that supports automation. Second, a software system must be built to provide personnel with on-line, measurement-based feedback. Third, the acceptance and effectiveness of such a system must be evaluated empirically.

This work is done in the context of a comprehensive approach for gaining intellectual control over software projects. The Quality Improvement Paradigm (QIP) calls for treating software development as an experimental activity from which continuous learning is possible [Bas85]. Continuous learning is supported by *measuring* and *modeling* the processes of the project organization. The aspect of measurement entails gaining a quantitative understanding of factors such

as cost, defects, etc. The aspect of modeling entails gaining an explicit understanding of the processes, products, and resources in the project organization. Although the measurement and modeling approaches are a natural fit with each other, and some recent work has recognized that fact [dBS95, Pfl95], little work has been done on integrating them.

1.3 Contributions

The major contributions of this dissertation are the following. First, a framework for integrated measurement and process modeling was constructed. This framework explores how process aspects may be incorporated with measurement, supports the joint application of these two improvement approaches, and reveals some of the implications of using them together. Use of this framework results in an integrated view that can be used to provide measurement-based guidance and feedback. Second, a process-centered software engineering environment was developed to automate some aspects of measurement-based guidance and feedback. Personnel who use this system are provided with information about the tasks expected of them based on the integrated view. An important part of the system's description is a statement of assumptions and requirements for such a process-centered software engineering environment. Third, the acceptance and effectiveness of that system were evaluated in a controlled experiment. The experiment compared the use of on-line and off-line guidance and feedback techniques to test the assumptions and requirements concerning automated support for measurement-based feedback.

1.4 Integration of Measurement and Process

We suggest integrating measurement and modeling technologies for the following reasons.¹ There is a growing understanding that useful plans for software projects must be based on explicit models of processes, products, and resources (see for example [HB92, HF94]). However, even the best models describing "how" a process should be performed are not helpful without an operational, measurable description of "how well" it should be performed. Similarly, the best plans for measurement programs are useless if they are based on incorrect assumptions regarding the processes, products, and resources to be measured [RUV92]. Therefore, both measurement and process modeling technologies are needed.

Measurement technology is needed to define target data values for the goals to be achieved by a project, to make the status of a project visible, to gather data that supports the performance of project team members, to predict future project performance, and to establish baselines against which improvement claims can be judged [BCR94b]. Experience tells us that useful plans for measurement activities can be established and significant improvements can be achieved based on goal-oriented measurement approaches in local environments [McG90].

Process modeling technology is needed to capture project goals explicitly, to build explicit models of real-world processes, and to integrate these models with other elements into compre-

¹Portions of Section 1.4 and subsections are reprinted from *Information and Software Technology*, 35(6/7), C. M. Lott and H. D. Rombach, "Measurement-based guidance of software projects using explicit project plans," pp. 407-419, Copyright 1993 with kind permission from Butterworth-Heinemann Journals, Elsevier Science Ltd., The Boulevard, Langford Lane, Kidlington OX5 1GB, UK. Not to be further copied without the publisher's specific permission.

hensive process views. This technology can be used to track project performance using explicit descriptions, to specify and plan changes in a process, and to improve process descriptions both on-line (i.e., while a project is being performed) and off-line (i.e., for future projects) [CKO92].

The integration of measurement and modeling technologies brings together two views of a single software development project. Essentially, implicit overlaps and relationships between measurement activities and development processes are made explicit. In this research, a "measurement view" describes the goals, questions, metrics, plans, data, results, etc. concerning the measurement activities in some development project. A "process view" describes the processes, products, resources, etc. involved in the project's work.

The integration of measurement and modeling views yields synergy effects during the project planning phase as well as during project execution [LR93]. During planning, the development and use of integrated views ensures that the measures are based on sound assumptions about the processes, and that an operational definition of quality goals has been established for the processes. During project execution, integrated measurement and modeling views support guidance, data collection, tracking, feedback, and replanning. Guidance means informing personnel in advance about their goals and tasks. Data are collected about those goals and tasks. Tracking involves comparing collected data with expected values to evaluate whether goals are being met. Feedback means notifying personnel of the results of tracking, possibly notifying them of problems. Finally, replanning involves deciding on steps to take when tracking reveals problems. The approach described in this work includes guidance, data collection, tracking, and feedback. Replanning is not addressed.

Examples of using integrated views for describing the status quo and for improving existing practices are discussed next.

1.4.1 Example 1: Describing the status quo

Any improvement effort must have a starting point, or a baseline for comparison. To describe the status quo (develop a baseline), an organization essentially performs descriptive modeling. Descriptive modeling means capturing existing processes, regardless of how well or poorly structured those processes (and the resulting models) are. The organization also builds quality models that describe how well the processes are performed. When describing the status quo, the integration of views helps identify the method, responsibility, and appropriate time for data collection, improves the identification of possibilities for collecting required data [Vis94], encourages process models that state quantitative goals rather than algorithms, and guarantees the correspondence of the measurement view with the process view. For example, Carr et al. built a baseline by modeling a set of existing, highly complex processes [CDP95]. The models were used to gain an understanding of the status quo and ultimately to restructure those processes.

A simple example is presented here. Consider a project named "DCT1" that designs, codes, and tests pieces of code. Describing the status quo of DCT1 would begin by modeling the products and then processes, resulting in a simple process view. The product-flow view of this example project appears in Figure 1.1. Product flow is modeled as inputs and outputs among the three processes "write_design," "write_code," and "test_code." Each process consumes and produces a single product. Forward product flow is modeled as a single document produced by each process and consumed by its logical successor. Backward product flow is produced and consumed similarly, but in the opposite direction. Not shown in the figure is control flow; the

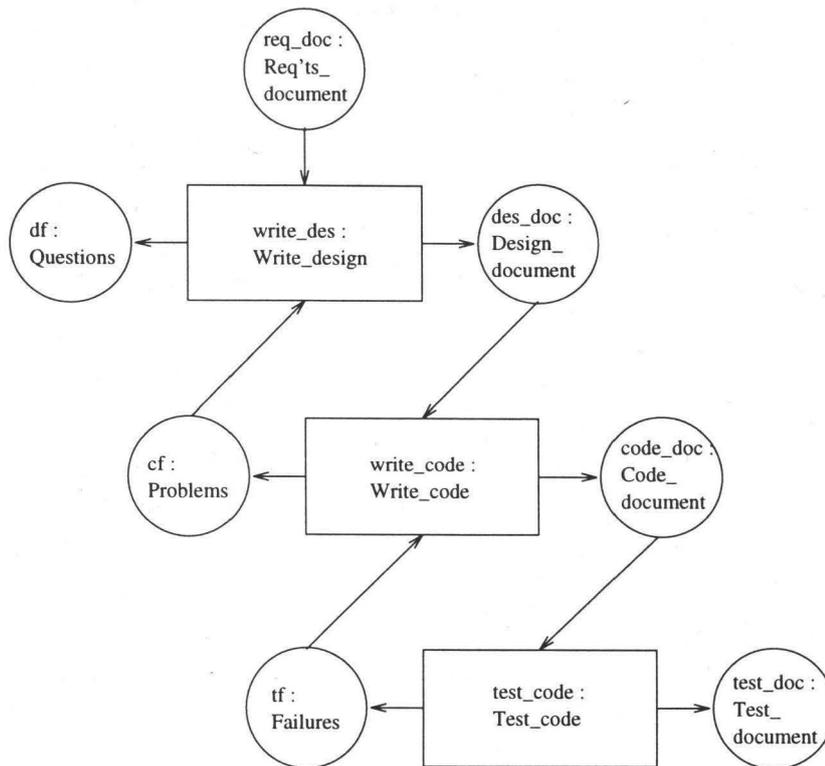


Figure 1.1: Product flow in example DCT1

flow of control in this project might be modeled as iterative cycles through the set of processes.

Next, assuming that the organization has no understanding of resource use and defect rates within the DCT1 project, it would develop a plan for measuring and interpreting this data (a “measurement view”). Rough consistency must then be achieved between the measurement and process views. “Rough consistency” simply means that some agreement has been reached among personnel about what processes and products actually exist, are important, and should be both modeled and measured.

An example measurement goal for the DCT1 project is “to analyze the design process, in order to understand it, with respect to the cost and detection of requirements defects, from the development team’s viewpoint, in the development organization.” This goal is tracked using data that are collected from the three processes. The personnel will measure resource and defect data. The project chooses to collect resource and defect data based on the staff-hours charged to each process and the defects found in each process. Once the personnel have agreed on what processes should be modeled and measured, the final part of integration involves deciding upon the responsibility (who), the procedure (how), and the triggers (when) for collecting the data.

After the integration is accomplished, the planners check various consistency properties of the resulting views on the DCT1 example. First, analyzing consistency lets them be certain that the two views of the project refer to the same products (e.g., the requirements document), processes, and resources. Second, the planners can judge whether all information needs from the measurement view (e.g., requirements defects) can be satisfied based on the information sources identified in the process view. Third, the consistency check guarantees that metrics specified in the measurement view (e.g., effort) and attributes in the process view match. Finally, the consistency

check ensures that the goal of gaining an understanding has been followed. Specifically, the data are collected during this project only to gain an understanding and to document the status quo. Although the consistency checks are straightforward in this example, the example demonstrates the necessity of checking for a one-to-one mapping between products, processes, and resources identified in the two views and for consistency among the data requirements and data sources.

1.4.2 Example 2: Improving existing practices

Once a baseline is established (i.e., both process and quality models exist), an organization can evaluate improvements against that baseline. The activity of creating models solely to inform and to direct personnel is labeled “prescriptive modeling.” The integration of measurement and process views makes it possible to provide automated assistance for data-collection activities and to supply measurement-based feedback to the people responsible for a software development project.

An existing baseline can be used to support guidance, data collection, tracking, and feedback. Guidance informs a project team that they are expected to perform some specific processes, data collection involves measuring those processes, and tracking evaluates those data against a baseline. Finally, feedback is used to warn the team that they are approaching a state from which they cannot recover. However, feedback does not entail preventing such problems. Prevention and recovery actions are considered to be the sole responsibility of the project team. For example, feedback may be generated to inform a project team that they have failed to find the expected number of defects while testing a module. The team has at least three possible alternatives. First, they can ignore the fact that they have violated the plan and continue work as before. Second, they can redo the testing process; perhaps additional testing will detect more defects. Third, the team can replan the project; essentially this recognizes that the goals as stated are unattainable.

In the context of providing guidance and feedback, data (e.g., “4.2 per week”) must be distinguished from information (e.g., “current defect rate exceeds the baseline by 30%”). Data becomes information after some evaluation has been performed. Only information, not data, is appropriate for use as guidance and feedback.

Consider an example instantiation of the measurement and process views described above for a new project called “DCT2.” In this case, the existing views will be used to provide guidance about the actions of the new project. The goals of DCT2 include using fewer than 1,000 staff-hours. The process “write_code” receives 40% of the project budget, so not more than 400 staff-hours total should be expended. Further, the project would like to detect a significant number of defects during design inspections and thereby reduce the number of design defects detected during coding and testing activities. These target values express quantitatively what is expected of the project in terms of both resources and defect densities. Because it is expected that the use of quantitative data will have an impact on the flow of the processes, data that is collected must be compared with baseline quality models. Based on the result of the comparisons, feedback will be generated for the developers. An additional consistency check is necessary to decide whether this aspect has been included in the integrated views.

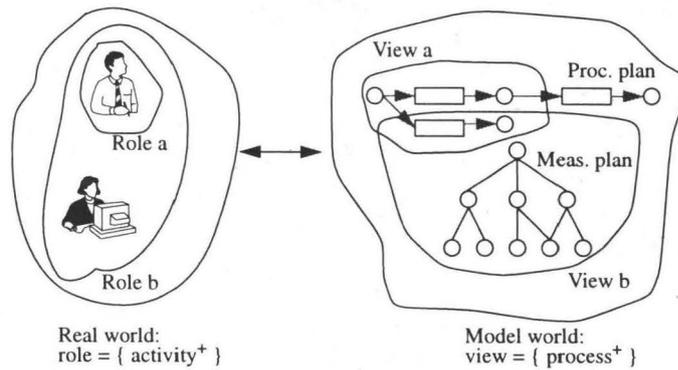


Figure 1.2: Illustration of the real and model worlds

1.5 Automation of Measurement-Based Feedback

Different levels of automation may be provided for measurement-based guidance and feedback.² If no automation is available, feedback is limited to off-line documents such as a project handbook. Alternately, documents may be provided on-line, for example in a software engineering environment. To illustrate automated support for measurement-based feedback, Fig. 1.2 shows the distinction between concepts in the real world (workplace) and the model world (computer), and the mapping between them. In the real world, a person assumes some *role* that determines the activities he or she will perform. In the model world, a person's activities are mirrored by a view [Ver94].

The following questions address the differences between no automation and extensive automation. These questions explain the advantages and required investments involved in using integrated measurement and process views to generate feedback:

- How can personnel be informed about the work expected of them?
- How can deviations from the integrated measurement and process views be detected?
- How is inter-role communication supported?
- To what extent can actions be performed automatically?
- To what extent can reliable measurement data be collected?
- What quality of feedback can be provided?

The rest of this section introduces five levels of automation for measurement-based guidance and feedback. These levels of automation range from the state of the practice to idealized systems. A conventional environment consisting of software construction tools is assumed as the foundation for all of the following five levels:

1. No explicit process nor measurement view (predominant state of the practice).

²Portions of Section 1.5 and subsections are reprinted with permission; please see the notice regarding copyright in Section 1.4. Not to be further copied without the publisher's specific permission.

2. Explicit process and measurement views (off-line documents).
- 3a. On-line process view and off-line measurement view.
- 3b. Off-line process view and on-line measurement view.
4. On-line, integrated measurement and process views (a process-centered environment that supports measurement).

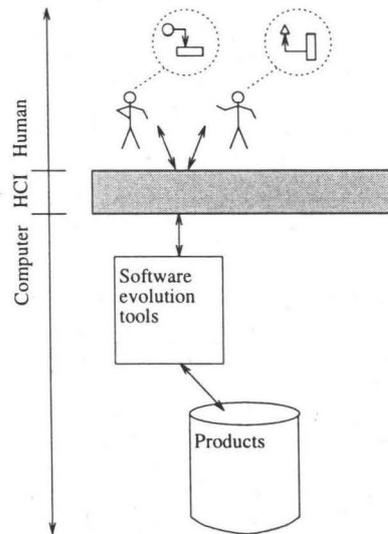


Figure 1.3: No process or measurement view (level 1)

1.5.1 Level 1: No explicit process nor measurement view

This level of technology, as illustrated in Figure 1.3, reflects state-of-the-practice software engineering environments (SEEs). Activities performed by individual project members are shown at the top, and actions carried out by a machine are shown below. The interface between project personnel and the software engineering environment is based on the invocation of software tools. Personnel may have dramatically different ideas about how they are expected to evolve software products. Also, they measure nothing about their processes, products, and resources. A clear understanding of project goals and requirements is necessary for the tools to be used effectively. However, these goals and requirements are only implicitly defined in the minds of the engineers. There is no possibility of measurement-based feedback at this level.

1.5.2 Level 2: Explicit process view and measurement view

The next technology increment, as illustrated in Figure 1.4, reflects current trends in software organizations. Activities are represented using a process view that describes the constructive and analytic process steps. Project goals are defined quantitatively using a measurement view. Project personnel may detect deviations from the goals and prescribed actions by invoking measurement tools and comparing the results with the goals. The measurement view clearly defines what data is required and when it should be collected, but the physical collection of measurement data

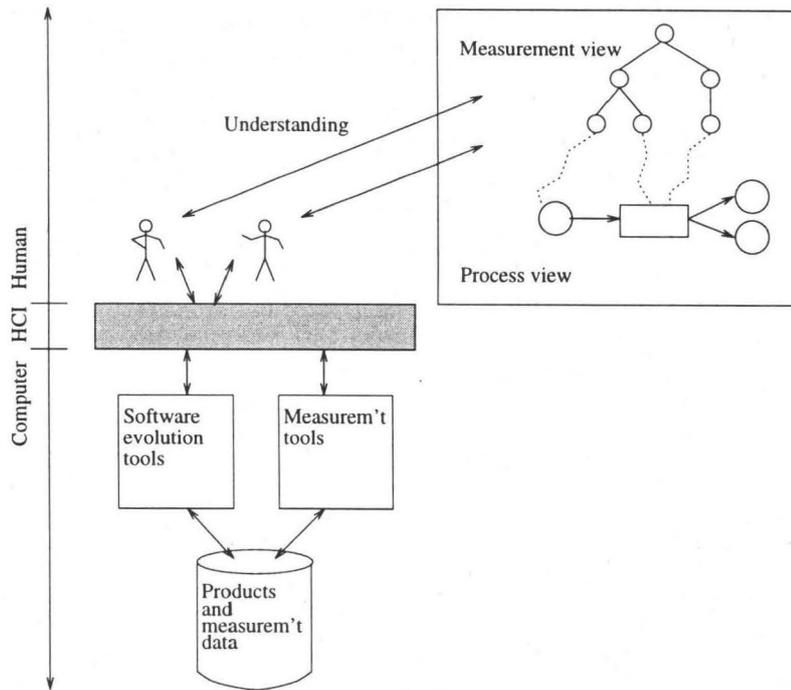


Figure 1.4: Explicit, off-line process and measurement views (level 2)

requires effort on the part of project personnel to fill out forms. Data-collection activities may be postponed or otherwise delayed until the end of the week or month to reduce their overhead, resulting in wide variances in data quality. Finally, quality feedback may only be provided to the project through the efforts of individuals who track the project state, read the process and measurement views, and monitor the measurement data. There is no guarantee that the project proceeds in accordance with the views, which makes static use of the views even more difficult.

Still, by using the process and measurement views off-line, a project may realize the advantages of improved understanding of their processes, quantitative definition of the criteria necessary for project success, and a clearly defined method by which the project state can be monitored. Reaching this level requires an organization to devote considerable effort towards defining both a process and a measurement view. An example of writing and using explicit process views appears in [HSW91].

Two real-world examples of off-line use of explicit process and measurement views are described next. NASA's Software Engineering Laboratory conducted a project to gain a quantitative understanding of a maintenance process [RU89b, RUV92]. The first attempt defined only a measurement view of the maintenance processes. Measurement failed to yield meaningful results because the process view implicit in the minds of the leaders of the study was not consistent with the existing processes. After process views of maintenance personnel were captured, modeled formally, and reviewed until conflicts were resolved (i.e., integrated with the measurement view), data collection efforts yielded useful results.

The second example is a case study performed with TRW [KNMS⁺92]. TRW personnel used a natural-language description of a reuse-oriented development process to build a process view using the MVP-L language [BLRV95]. They analyzed the MVP-L representation for completeness and consistency, changed the representation to fix problems and improve it, and finally

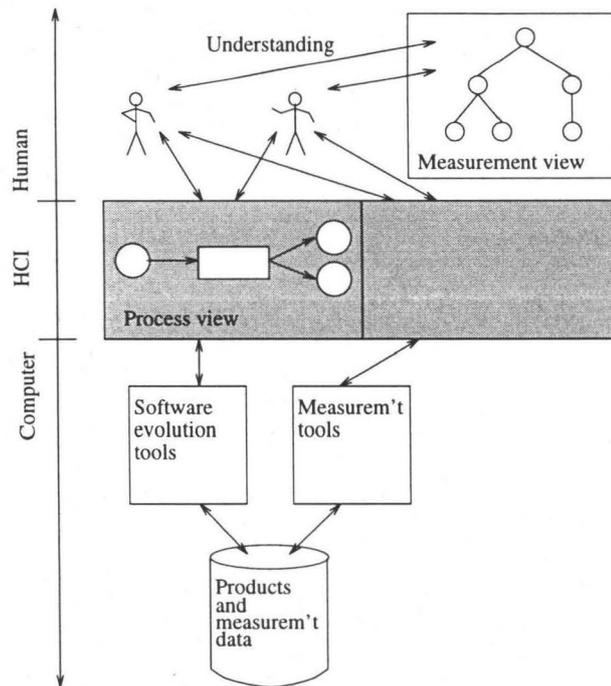


Figure 1.5: On-line process view, off-line measurement view (level 3a)

rewrote an improved natural-language description. The benefits for TRW included an improved understanding of the process and the generation of an internally consistent process view.

1.5.3 Level 3a: On-line process view, off-line measurement view

Figure 1.5 illustrates a process-centered environment in which tool use is mediated by a process view, but without measurement support. Such a system is the first step towards offering project personnel a role-specific, process-centered interface through which they accomplish their work. The process view is essentially limited to evaluating process entry and exit criteria in terms of product flow. Deviations from the process view can be detected when products are changed or not produced as expected. Inter-role communication is provided in the sense that a version control system tracks user's actions and prevents conflicts. However, this level of technology offers no on-line support for representing and using the quantitative criteria that define project success, so we claim that this level represents no significant progress over using a sophisticated version control system. Some actions can be performed automatically exactly when users interact with the revision control system. The collection of reliable measurement data is not supported on-line, so it is subject to the same problems discussed in Section 1.4. Finally, the quality of guidance and feedback that can be offered based on product availability is mostly limited to informing project personnel of processes that can begin and notifying them of changes in input products.

Nonetheless, this technology increment offers many advantages over an off-line process view (level 2). Some conflicting actions can be detected automatically, deviations from the process view can be reported, and rudimentary feedback is possible. Further, because essentially all of these benefits accrue through the use of version control technology, the investment required of an organization to achieve this level is moderate. Version control systems such as RCS [Tic85] are well understood and freely available.

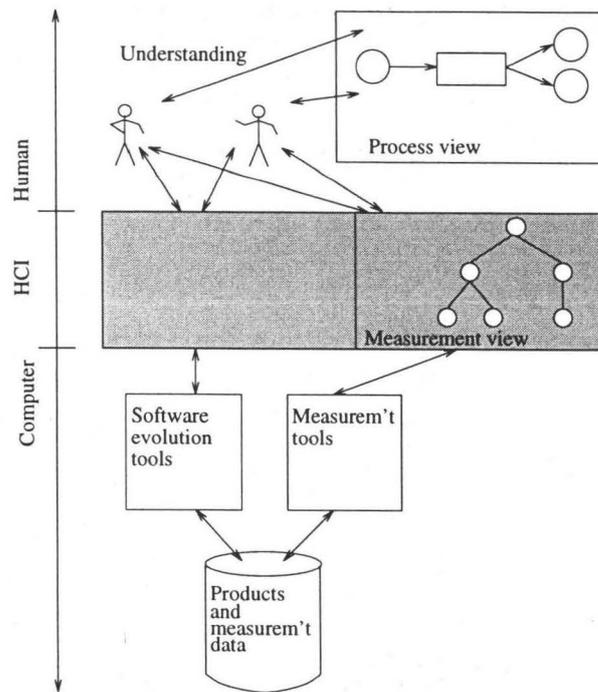


Figure 1.6: Off-line process view, on-line measurement view (level 3b)

1.5.4 Level 3b: Off-Line process view, on-line measurement view

An environment that supports on-line collection and interpretation of measurement data, but does not incorporate a process view, is depicted in Figure 1.6. Measurement will be successful if the project follows the processes assumed by the measurement view. Although the process view is not automated, it is explicitly defined. Only the diligence and attention of users can detect deviations in project performance from the plan. In this level of technology, there is no support for inter-role communication. Many activities pertaining to measurement may be automated, especially the routine collection of measurement data. Automated support for data collection can be expected to yield much higher quality data as compared to data that is collected off-line. Finally, limited feedback can be provided if empirical models for the process and environment are available.

The main advantage of this level of technology is that many trivial activities involved in gathering measurement data can be automated using existing tools. The hypothesis here is that this increased automation will dramatically improve the validity of the collected data. The immediate return on investment to an organization that achieves this level of integration is higher than at level 3a, primarily because the extra work required by measurement activities can be partially offloaded onto automated tools.

This level of technology was implemented by NASA's Software Engineering Laboratory in a system called the Software Management Environment (SME) [DV92]. SME demonstrates the on-line use of a measurement view. The SME system accesses large quantities of data from ongoing projects, and compares that data to models of the typical and target values in the NASA environment. When current project data deviates from the baselines, the system can offer possible interpretations for the deviations.

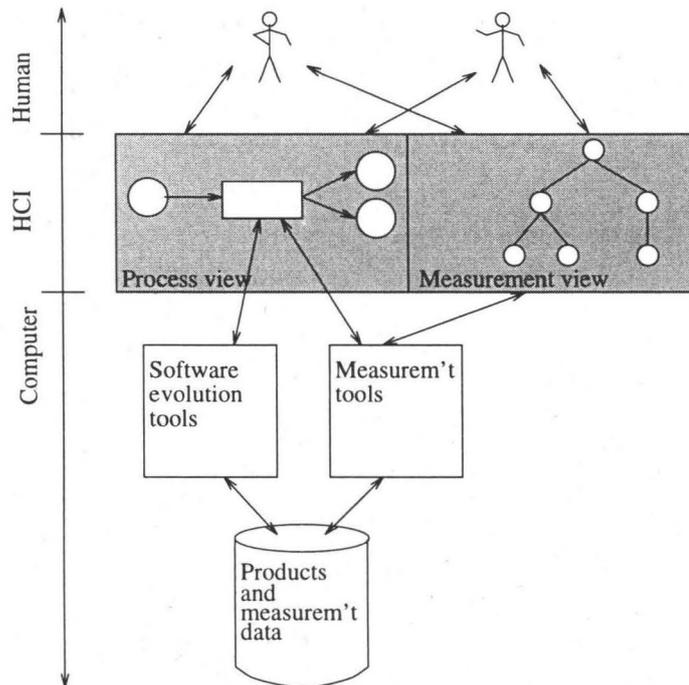


Figure 1.7: A process-centered software engineering environment (level 4)

1.5.5 Level 4: Process-centered environment

The use of an integrated measurement view and process view is shown in Figure 1.7. This level of integration represents an idealized system that goes beyond current practice. In this level, quantitative criteria for project success are integrated with an explicit process view and used to provide feedback to personnel. The interface in such an environment offers indirect access to software evolution tools, and represents a paradigm shift from unmonitored access of software evolution tools to managed, process-centered access of the same tools. Lines in the figure between the measurement and process views represent the integration of quantitative targets and baselines with aspects of the process views. The system will gather quantitative data and offer feedback according to the collected data. By providing such an interface, a process view serves as an integration mechanism for a software engineering environment [WF91].

Users may query the project state at will, and deviations will be detected and reported to project personnel automatically. Project personnel are kept informed about which activities are possible, which activities are currently being performed, and which activities have been affected by the unexpected results of a related activity. Activities as used here are not fine-grained tasks such as compiling a code module, but coarse-grained tasks such as designing a subsystem or conducting a design review. This type of feedback helps teams coordinate their actions and detect when independent actions have come into conflict.

Actions such as periodic data collection may be performed automatically. Such actions may be triggered by changes in project state initiated by project personnel (e.g., starting a process) or by the tracking system (e.g., passage of time).

Data can be collected on a timely basis, and an on-line support system can further guarantee the consistency of the project state with the model world (i.e., views maintained by the idealized

system). Some types of measurement lend themselves to automatic collection. A simple example is static source-code metrics, which can be collected from a version-control system. Other measurements can only be collected from project personnel. Examples are resources consumed by review activities, characterizations of defects removed, and final completion dates of processes. In all of these cases, automated support of data collection can be provided using something like a forms-based tool to reduce paperwork and encourage timely collection of data. It cannot be overemphasized that automated support for data collection must work harmoniously with project personnel to reduce the costs of this activity, not to check up on personnel.

Finally, given that a deviation from the integrated views has been detected, and the project team has not chosen to ignore the deviation, the system can assist with replanning and recovery. For example, backward chaining can be used to estimate how far work must be rolled back in order to restore consistency with the original goals. The ideal system would give project personnel relevant feedback well in advance; i.e., before an inconsistent project state is ever reached.

The DCT1 example is again used to illustrate the capabilities of a process-centered environment. We show how a team member responsible for maintaining the consistency of the on-line views with the current project state requests information and enters changes in a system. Assume that the members of the DCT1 project have completed the "Write_design" process, and intend to begin the "Write_code" process. To keep this example simple, assume that the two processes do not overlap. The responsible team member indicates to the system that the design process is complete. The system responds by collecting information. Data for the design quality aspects of coupling and information hiding is collected using a design-measurement tool, and data regarding resource consumption for the design activity and requirements defects found during the design process is requested from the team member using a forms-based tool. The data are then used to evaluate the exit criteria for the design process. In this case, assume that the design team has consumed 105% of their resource allocation. Defect detection data is substantially below the baseline value.

The exit criteria are not fulfilled due to the excess resource consumption. The team member can ignore the problem, repeat the process, or replan. The resource limit was only exceeded by 5%, so the team member ignores the problem and marks the process as complete. The completion of the "Write_design" process fulfills the entry criteria of the "Write_code" process, meaning that it can be started. This fact is immediately reported to the team. In response, the team member initiates a second change in project state by notifying the tracking system that the team intends to start the writing code process. That process's entry criteria are true, so this change is accepted. The system's project state is thus made consistent with the fact that team members are proceeding with the "Write_code" process.

1.6 How to Read the Dissertation

Each chapter begins with an overview section that summarizes its content. The dissertation may be skimmed by concentrating on the overview sections.

Chapter 2 offers a survey of existing measurement frameworks, process modeling notations, and process-centered software engineering environments. This survey identifies work that can be reused and hypotheses that others have tested. Chapter 3 then gives the framework for integrating

measurement views with explicit process views. That foundation is used by a prototype software system to supply measurement-based guidance and feedback, as described in Chapter 4. The prototype system implements portions of automation level 4. Finally, the use of that system is evaluated in a controlled experiment as discussed in Chapter 5.

Chapter 2

Survey of Related Work

This chapter surveys work on measurement frameworks, process modeling notations, and process-centered software engineering environments. Section 2.1 presents an overview. Section 2.2 surveys research on measurement frameworks, and Section 2.3 surveys process modeling notations. These bodies of work form the foundation for providing measurement-based feedback, either manually or with automated support. Work on process-centered software engineering environments is surveyed in Section 2.4 to identify support for measurement and process technologies. Finally, empirical evaluations of software engineering environments are discussed in Section 2.5 to identify the hypotheses that others have tested.

2.1 Overview

A measurement framework is an approach for deciding what to measure and how to interpret the resulting data. Early measurement frameworks in software engineering focused purely on source code. Later work supported measuring processes as well as products. The survey discusses “Quantitative Measurement of Software Quality” (published by Rubey and Hartwick in 1968), “Software Quality Characteristics” (first published by Boehm in 1973), the Rome Air Development Center extension of Boehm’s work (first published by McCall et al. in 1977), “Software Quality Metrics” (first published by Murine et al. in 1980), “The Goal Question Metric Paradigm” (first published by Basili et al. in 1981), an extension to the Rome Air Development Center’s work (done by Bowen et al. at Boeing Aerospace and published in 1985), and “Quality Function Deployment” (first applied to software around 1988). The survey shows that all existing frameworks work in a goal-oriented manner. All begin with a statement of goals, then refine the goals in an operational, tractable way into measures for which data can be collected. Reuse of those frameworks that offer predefined goals can be simple, provided that the user’s goals match the framework’s goals exactly. Reuse of more flexible frameworks is challenging because the user must set goals and refine them for his or her particular needs.

A software process modeling notation is a formalism for creating models of the processes, products, and resources involved in a software development or maintenance project. Nine different formalisms are surveyed, namely E-L, SADT, STATEMATE, HFSP, MELMAC, MVP-L, APPL/A, MARVEL, and MERLIN. All are intended to help people understand and improve processes by making them explicit. The specific goals pursued by the various formalisms include improved

communication among personnel, assistance with on-line tasks, and control of the tasks to be performed. Some notations are paired with execution support that offers guidance and feedback to software developers about their tasks. The granularity of the tasks for which each formalism is best suited varies from extremely fine (analogous to a formal programming language) to highly abstract (analogous to a requirements description notation).

A process-centered software engineering environment is a computer system that supports software developers and maintainers in their work through its awareness of the processes that they are expected to perform. All of the process-centered systems considered here include some aspect of measurement. The systems surveyed are Ceilidh, Ginger, Amadeus, ES-TAME, Provence, and SynerVision. Many of these systems represent concrete steps towards integrating software construction, process definition, and measurement capabilities into a single system. However, the survey shows that none permit the flexible definition of processes and measures in a way that supports measurement-based feedback.

Finally, a study that compared two software engineering environments (process enactment systems) is discussed. Extremely little empirical work has been done in this area, and this study illustrates some of the difficulties.

2.2 Measurement Frameworks

It is generally accepted that measurement is not an end in itself but a means to an end. Measurement should be viewed as an infrastructure technology that is necessary to achieve systematic improvement [BCR94b]. However, deciding precisely what to measure and interpreting the resulting data are not simple tasks.

Many frameworks have been developed that assist software engineers in deciding what to measure and in interpreting the data. Early frameworks focused on source code. Later work addressed the measurement of other software development products such as requirements and designs, as well as the processes in which these products are developed and used. All of the frameworks surveyed here advocate a goal-oriented approach. This means that measurement goals are defined first, and then the goals are refined into measures that can be gathered. However, each framework takes a different approach towards stating and refining goals.

A brief description is given for each framework, and each is additionally characterized by answering the following questions.

- When did the first publication appear about applying the framework to software?
- What is the scope of the framework (what is it good for)?
- What was the original application of the framework?
- What support is provided for defining custom goals?
- What techniques are defined for refining goals into measurable quantities?
- What is the purpose of measurement?
- What types of objects are measured?

- What aspects of those objects are measured?
- Whose point of view is considered?
- What guidelines are provided for interpreting the resulting data?

2.2.1 Quantitative Measurement of Software Quality

Rubey and Hartwick studied software used in spacecraft and derived an extensive list of quality attributes [RH68]. They defined seven high-level attributes, each of which is refined into six to twelve more specific attributes. Complex formulas are used to assign a number to each attribute. The scope of this framework entails assessing the source code of programs that are in development. The original application of the framework is source code, but no specific implementation language is targeted. No support for defining custom goals is offered, and the refinements of the seven high-level attributes are fixed. The purpose of measurement is to determine the current “level of quality” (the authors’ words) of a program and to use that information to direct the program’s development. Only source code is measured. The aspects measured include the correctness of mathematical calculations, the usage of time and memory resources, and the intelligibility of the code. The user’s (purchaser’s) point of view is primarily considered. They provide interpretation guidelines in terms of a system for ranking the results on a scale of 1 to 100.

2.2.2 Software Quality Characteristics

Boehm, Brown, and Lipow define an approach for evaluating the quality of software quantitatively using “Software Quality Characteristics” (SQC) [BBL76, BBK⁺78]. This approach, first defined in 1973, uses a tree to relate a predefined set of software quality characteristics (i.e., goals) to each other. Ultimately the characteristics are refined into measures that can be used to evaluate the predefined quality characteristics. For example, the characteristic “portability” would be evaluated in part based on measuring the number of assumptions made in the code about the floating-point accuracy of the machine. They additionally discuss developing checklists based on the data and building tools that will gather data for the measures automatically. The predefined set of qualities is both a strength and a weakness. They offer a cookbook approach that can be applied with little effort. However, the approach is highly specialized and no mention is made of extending the tree for quality characteristics outside their predefined set.

The scope of the quality characteristics tree approach is limited to determining various qualities of a software product, and the original application was to evaluate FORTRAN source code. No support is provided for defining goals that are tailored for a user’s specific needs. Instead, a predefined set of qualities (portability, reliability, . . .) is defined. The implicit goal is to evaluate the software product in terms of those qualities. A predefined refinement is provided for the set of goals. No techniques are provided for refining the goals according to the user’s special requirements. The purpose of measurement is to detect anomalies in the software product that might lead to problems, and thereby to improve the process by shortening the life cycle. Approximately 160 measures are defined strictly for FORTRAN source code. Various source code aspects including size, structure, and commenting are measured. Both developer’s and user’s views are mingled in this framework. For example, both reliability and testability of the product

appear in the predefined tree of characteristics. The predefined set of goals and measures makes interpretation of the data relatively straightforward.

2.2.3 Factors in Software Quality

McCall, Richards, and Walters synthesized previous work done by Boehm et al. and named it Factors in Software Quality (FSQ) [MRW77, WM79]. Boehm's work was tailored for use in a formal software procurement environment, namely the U.S. Department of Defense, as sponsored by the Rome Air Development Center (RADC). McCall et al. also extended Boehm's work with the use of quantitative measures for early phases of development, and attempted to validate some of the concepts formally. They defined 11 factors (correctness, efficiency, reliability, etc.), and discussed the refinement of those factors into criteria (accuracy, commonality, conciseness, etc.). Each factor is refined into one or more criteria, and criteria are in turn refined into metrics.

The scope of the work done at the Rome Air Development Center extends that of Boehm's work by removing the constraint of a single programming language. The original application was systems in the military software contracting domain. No support is provided for defining new factors (i.e., goals). Factors are refined via criteria into metrics. Although a refinement for each factor is predefined, the framework specifically allows the addition of new or revised metrics. The purpose of measurement is to allow a procurement office to judge the overall quality of a software system, both during development and at delivery. All documents from the software life cycle may be measured, and a number of different aspects of those objects are measured. The point of view is strongly focused on that of a procurement organization. Finally, some guidelines are given for interpreting data.

2.2.4 Software Quality Metrics

Murine adopted significant portions of both Boehm's and RADC's work, and marketed his approach under the name "Software Quality Metrics" (SQM) [Mur80, CM83, SAY85]. SQM instructs users to evaluate the quality of software by choosing high-level factors, refining those factors into criteria, and refining those criteria in turn into metrics. The SQM approach defines a fixed set of 12 factors, 23 criteria, and over 400 metrics. Some emphasis is placed on scoring techniques that will yield a single number representing the "goodness" [author's terminology] of an artifact. Murine states that the data gathered for the metrics must be interpreted in light of the criteria and factors, an argument for viewing SQM as a top-down approach to measurement. However, he also states that the appropriate life-cycle phase and documents must be considered above and beyond the factors and criteria in order to choose metrics, which suggests that the factors and criteria alone are insufficient for the selection of metrics.

The scope of the SQM approach is limited to assessing the conformance or nonconformance of products with respect to the predefined factors, and the original application was on software. Because the set of factors is fixed, SQM does not support the definition of user-specific measurement goals beyond limiting the number of factors to be considered. The implicit goal is to evaluate the quality of the software in terms of the factors. Again similar to Boehm's approach, the predefined refinement of the factor, criteria, and metric hierarchy precludes any tailoring of the refinement hierarchy. The purpose of measurement is to evaluate many quality aspects of a software artifact and to produce a single score. The types of objects to be measured are strictly

source code. The aspects of those objects that are measured include correctness, testability, and maintainability. Similar to Boehm's work, no single point of view is singled out; both user's, developer's, and manager's views are mingled. A fair amount of instructions are provided for interpreting the data by graphing and scoring the results.

2.2.5 Goal Question Metric Paradigm

The Goal Question Metric Paradigm (GQM Paradigm) was developed in response to the need for a framework that embodies a top-down approach to measurement in software engineering [BW81, BW84, BR88, RU89b, Rom91b, Bas92, Dif93, BCR94a, DHL96]. The GQM Paradigm supports defining the goals behind measuring software processes and products, offers guidance for refining the goals into measures, and gives assistance in interpreting data. GQM reuses the idea embodied in the Boehm and Murine approaches of using multiple levels to refine high-level goals into measurable items. However, GQM removes all restrictions of those approaches on the high-level goals. Definition of goals is supported by templates that prescribe what aspects must appear in a goal, including purpose, perspective, and context. The top-down refinement of goals is accomplished first by defining questions and ultimately by selecting metrics that will answer the questions. Guidance for choosing the set of questions that refines a high-level goal is provided in the form of two structures, one for product-related goals and one for process-related goals. Recent work has defined a number of techniques that support the definition and use of GQM hierarchies [DHL96].

The scope of the GQM Paradigm is any process, product, or quality model involved in software development or maintenance. The definition of tailored goals is strongly supported, actually required. Definition of a hierarchy for refining goals into metrics is also required by GQM. The purpose of measurement is freely definable by the user, and may include characterization, prediction, evaluation, improvement, etc. Any objects (processes, products, resources, quality models, etc.) can be measured, and similarly any aspects of those objects may be selected for measurement. Stating the explicit point of view (the person who is most interested in the results) is also required, and may include the developer, the manager, the researcher, etc. Finally, data are interpreted by using the hierarchy of questions. Data collected for the metrics are used to provide answers to the questions, and these answers in turn are used to evaluate the goals.

The term "GQM plan" refers to a single goal plus supporting questions and metrics. Figure 2.1 illustrates the refinement levels in a GQM plan. The single goal identifies the *object of interest*, the *quality aspect of interest* with respect to that object, the *viewpoint* taken in the study, the *purpose* of the study, and the *context* of the object and study. The goal is refined using three, top-level sets of questions, as follows. Question set O defines the objects of interest and variation factors that influence those objects. Question set Q refines the definition of the quality aspect as stated in the goal. Question set F defines the feedback relating to the quality aspect of interest. The three sets of questions may in turn be refined by further questions; and finally the leaf questions are mapped onto metrics.

This refinement hierarchy is a directed acyclic graph. A question can depend on any other question or any metric for its refinement, but cycles are prohibited in a GQM plan (e.g., questions refined in terms of themselves). The left arrow in Figure 2.1 points downwards to represent how objects are refined when following a top-down refinement approach. Similarly, the right arrow points upwards to represent how people use a GQM plan to interpret the data using the questions, and to track and evaluate whether the goal has been met.

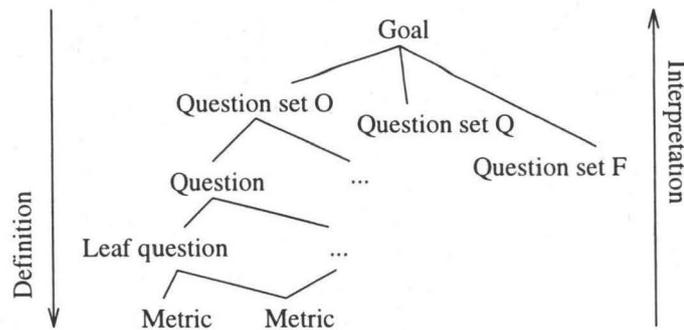


Figure 2.1: Refinement levels in a GQM plan

The depth of refinement of metrics is an important issue for the purposes of this dissertation. As used here, GQM metrics are deliberately not refined to encourage the use of questions to document the refinement. However, for the purposes of this research, all GQM metrics must be directly collectible. For example, an example GQM metric might be “design effort” and would presumably be directly collected by asking a person. A metric such as “productivity” would be a GQM metric only if a person is directly requested to provide data about productivity; e.g., whether their productivity on some project was higher, the same, or lower than expected. If the measure of productivity was intended to be refined into units of time and work, then it must be defined as a question, not as a GQM metric. Keeping all refinements in the GQM plan guarantees that the rationale behind each metric is explicitly documented, and thereby supports the interpretation activity.

However, a GQM metric will not necessarily include a precise statement of the when, how, and who of data collection. This remaining bit of abstraction allows a metric to denote the collection of a *set* of data values. For example, the metric “experience of the developers with the programming language in years” is a directly collectible metric, but it will result in a set of data values, one for each of the developers (i.e., one for each of the multiple instantiations of this aspect in the project). The additional information necessary to indicate who collects data, when, and how is captured in a measurement plan.

2.2.6 Software Quality Attributes

Bowen, Wigle, and Tsai of Boeing Aerospace Company enhanced and extended the previous RADC work to produce “Software Quality Attributes” (SQA) [BWT85]. Their work consists of an enhanced version of the quality framework, and a methodology suitable for use by a U.S. Air Force software acquisition manager. The methodology integrates the quality framework into a software acquisition management process. This process is expected to be applied within the context of a (then) proposed standard for the software life cycle. Many of the enhancements and extensions performed on the framework were made to achieve consistency with other U.S. Department of Defense work on measurement and evaluation. Another large part of the work focused on reworking the data collection mechanisms (e.g., forms, tables, and worksheets) that support the quality framework. As in the RADC quality framework and the Murine approach, a number of user-level quality factors (i.e., goals) are defined. These high-level factors are refined via criteria into concrete metrics. The extended framework includes 13 predefined quality factors, 29 criteria, and 73 metrics. Bowen et al. also introduced the idea of refining metrics into subparts (called elements), and accordingly defined over 300 metric elements for their set of metrics. The

methodology includes support for choosing the appropriate factors to be evaluated. Based on the need to accommodate evolving user concerns, a major extension to prior work supports the definition of new factors, criteria, and metrics.

The scope of the SQA work is similar to the prior FSQ work, namely on military command and control systems. The original application is the evaluation of embedded software systems procured for the U.S. Air Force. Limited support is provided for defining new factors (i.e., goals). Factors are refined via criteria into metrics, and metrics can further be refined into elements. Refinements are provided for all predefined factors, but the refinements can be extended, as can the sets of factors, criteria, metrics, and metric elements. Identical to prior work, the purpose of measurement is to allow a procurement office to judge the overall quality of a software system, both during development and at delivery. All documents from the software life cycle may be measured, and a number of different aspects of those objects are measured. The point of view is strongly focused on that of a procurement organization. Some predefined guidelines are given for interpreting data.

2.2.7 Quality Function Deployment

The Quality Function Deployment (QFD) approach is a mixture of a top-down measurement framework and a general process-improvement framework [KA83, AG88, Kin89]. Kogure developed QFD in the late 1960s in Japan to improve the design process for manufactured products. Zultner later extended QFD for use in the software engineering domain [Zul93]. The QFD approach strongly advocates customer-centered design as well as listening to the “voice of the customer.” As a consequence, QFD includes tools for formatting customer information, determining priorities of company responses to customer demands, encouraging members of the organization to work in teams to meet customer demands, and tracking quality characteristics through design and manufacturing process. The last aspect has the strongest measurement nature, and is the focus here.

The scope of the QFD approach is the process of designing a product that will best satisfy a customer’s needs and expectations the first time, thus reducing the need for redesign. The original application of QFD was on design processes for manufactured goods. As tailored for the software engineering domain, QFD relates characteristics of the final software product to characteristics of earlier products. QFD does not offer the user much latitude in defining goals. The purpose of measuring (identifying quality characteristics) is to increase the product’s chances of acceptance among customers. There seems to be less need to refine high-level goals into measurable characteristics in QFD than in other frameworks, because QFD offers matrices that encourage the user to state quality characteristics and functions at a fine level of granularity. The objects measured are both the capabilities of design processes and the products. QFD directs the user to measure two types of aspects, namely qualities as perceived by the customer and the product’s functions as seen by the producer. Matrices are used to analyze how the functions satisfy the quality needs of the customer. The primary viewpoint taken is the product’s designer, but as mentioned above, the voice of the customer is expected to influence every part of the process. The QFD framework includes extensive support for interpreting data, specifically a set of some 30 charts that help the user analyze interactions between customer demands, quality characteristics, and functions.

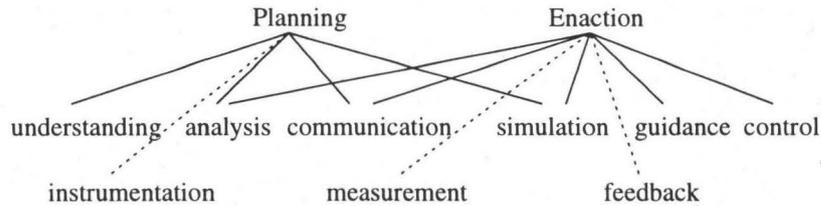


Figure 2.2: Activities supported by process model notations

2.2.8 Summary of measurement frameworks

Table 2.1 uses the classification scheme presented earlier to give an overview of the measurement approaches. A unifying theme in all of these approaches is the use of a top-down refinement approach for breaking high-level goals into measurable quantities. The approaches developed in the 1960's and 1970's are quite restrictive in terms of what could be measured, why it was measured, the method of interpreting the data, the viewpoint considered, etc. The approaches from the 1980's remove many of the restrictions on the facets; the GQM Paradigm is especially flexible.

2.3 Process Modeling Notations

A process notation is a formalism for creating models of software experience and knowledge, for example the process by which a requirements document is written. Process notations are sometimes called *software process metamodels*, because they are models for representing software process models. A number of process modeling languages have been developed [KR90], and first experiences regarding their practical usefulness are documented [Kel89, RU89b, PS91, KNMS+92, RUV92, CKO92]. Example notations are discussed in this section. Some of the examples presented also have enaction support systems. Abstraction and refinement in process notations are introduced, various goals of notations are described, levels of expressive power are discussed, and then research on process notations is surveyed.

Principles of abstraction and refinement enhance the expressive power of a process notation. The use of abstractions when representing a process helps manage the complexity of the representation and thereby improves human comprehension. Abstractions may serve as views of a process that hide certain details and expose the important relationships that may be obscured by the details.

Each process notation was designed to support a limited set of activities. Notations for modeling in-the-small focus on activities of enaction and control, while notations for modeling in-the-large include high-level activities of both planning and enaction. Figure 2.2 shows some of the relationships among the activities discussed here. At the top, planning and enaction are the abstract activities that help differentiate among notations.

Planning means preparing for a project by choosing a work model, selecting resources, defining constraints, etc. The activities supporting planning are understanding, analysis, communication, and simulation. First, the activity of promoting understanding means that the notation was designed to be comprehensible even to those unfamiliar with process modeling notations by giving an explicit, natural definition of the process activities. Second, dynamic analysis of a

Table 2.1: Overview of measurement frameworks

Facet	Approach						
	Quan. msrmt.	SQC	FSQ	SQM	GQM	SQA	QFD
Author(s)	Rubey et al.	Boehm et al.	McCall et al.	Murine	Basili et al.	Bowen et al.	Kogure et al.
First pub'd re S/W	1968	1973	1977	1980	1981	1985	1988
Scope of approach	Products	Products	Products	Products	Prod., proc., ...	Products	Prod. + proc.
Original application	Source code	Fortran	Software	Software	S/W devel.	Software	Manufac.
Define custom goals	No	No	No	No	Template	Limited	Limited
Custom refinement	No	No	No	No	Yes	Limited	Yes
Purpose of meas.	Evaluate	Evaluate	Evaluate	Evaluate	Understand, ...	Evaluate	Improve accep.
Objects measured	Code	Code	Products	Code	Any	Products	Prod. + proc.
Aspects measured	57 attributes	160 metrics	Extendable	400 metrics	Any	Extendable	Any
Point of view	User, purch.	Dev./user	Procurement	Dev./user	Any	Procurement	Customer
Interpreting data	Predefined	Predefined	Predefined	Predefined	Hierarchy	Predefined	Tables

process model depends on an enaction support system for the notation. Third, a comprehensible process definition that appears natural to process participants will aid in communicating the requirements of that process to those who must enact it. Fourth, execution of a process can be simulated by using the corresponding process model.

Enaction is the activity of executing process models to provide guidance and feedback to personnel. The activities supporting enaction are analysis, communication, simulation (all mentioned above), guidance, and control. A goal of process guidance means that the notation addresses modeling in-the-large and focuses on issues of process coordination. If control over a process is a goal of the notation, then that notation will probably limit itself to small, fully automatable processes. For example, control over recompilation activities is straightforward; control over humans, however, is a fuzzy issue.

Finally, shown at the bottom of Figure 2.2 are the three activities that are the focus of this research, namely instrumentation, measurement, and feedback. Instrumentation is done during planning. Measurement and feedback occur during enaction. Instrumentation refers to attributing process views with definitions of data. Notations that support measurement may provide ways of defining data items to be collected or may automate some data collection procedures. Last, the capability to provide feedback based on measurement data can help personnel during their projects.

Different process notations have different levels of expressive power. Notations are categorized here as either requirements-level, design-level, or code-level notations. First, requirements-level process notations describe basic process steps and provide only general guidance for their use in a project. These notations are useful for creating the Universal-level models identified by Humphrey [Hum89]. Second, design-level process notations represent task prerequisites and anticipated results, useful measures, important checkpoints, and task sequences. By leaving the lowest levels of detail up to the personnel, design-level representations permit maximum creativity because process execution is not restricted unnecessarily. These notations can be used to create Humphrey's Worldly-level models. Last, code-level notations are capable of representing enormous amounts of process detail. For example, a specific process activity may be automated using a code-level notation, or a standardized method may be described so that execution of a task can be carefully controlled. These notations must have algorithmic specification capabilities. A code-level notation may be used to create Humphrey's Atomic-level models.

A few of the notations that were presented at the Sixth International Software Process Workshop are summarized in the next few sections to show the diversity of approaches [Kat90]. Notations are placed into one of the requirements-level, design-level, or code-level categories based on the highest level of expressive power that, in our opinion, the notation can usefully provide. Like the domain of conventional programming languages, no single process notation can offer extensive support at all levels. The ideal process notation to support the needs of this research will have sufficient expressive power to create understandable models of process knowledge at least as high as the design level, and will be able to integrate models with measurement approaches. A primary consideration is whether the notation has the power to express knowledge about constraints on activities. Example notations are evaluated according to these needs [Kel89, RU89b, PS91, KNMS⁺92, RUV92].

A simple classification scheme developed by Kellner and Rombach is used in this section [KR90]. Aspects of the classification scheme are the notation's objectives, the representation formalism, the core paradigm for formal languages, specification of behavior, and specification

of functionality. Notation objectives are described using activity-words from the discussion that appeared above. The representation formalism used by process notations may be a formal language, structured English, or a graphical representation. The majority are formal languages. Formal languages have more expressive power than graphical notations, and lack the ambiguity that accompanies prose descriptions. The core language paradigm for execution of formal language notations is either the imperative paradigm or the rule-based paradigm. Imperative languages use a procedural notation for representing activities. Rule-based languages define activities based on preconditions and postconditions, and tend to be more declarative than the procedural languages. Functionality of a process refers to the inputs and outputs to and from that process. Finally, process behavior is specified by either activation and deactivation clauses for rule-based languages, or simply by the notation for procedural languages. Additional classification aspects include whether the notation supports abstraction and refinement, whether execution semantics are defined for the notation, and whether an enactment support system exists.

Finally, a notation's support for measurement is also mentioned. Measurement may be supported explicitly or may be treated as just another activity to be modeled. Explicit measurement support means that the notation provides a way of describing the collection and use of empirical data in the context of a software development project. Implicit measurement support means that the notation can be used to describe measurement activities that run in parallel with project work, but empirical data cannot be used in a substantive way to influence the project.

2.3.1 Requirements-level notations

Requirements-level formalisms support highly abstract descriptions of processes and related products. The three examples presented are either purely graphical or heavily dependent on graphical notations. Those notations classified here as requirements-level notations are E-L, SADT, and STATEMATE.

E-L

The E-L approach uses a graphical notation based on activity descriptions [KC90]. Activity descriptions capture process knowledge by identifying required activity parameters and by providing a means for refining the activity. The E-L graphical notation is based on transaction graphs that specify how activities should be coordinated. An interesting part of this approach is the concept of views, which are defined as projections from a transaction graph. This approach states the objectives of communication, analysis, and execution control. Basic components of this notation are activities, objects, and agents. The core notation paradigm is entity-relationship diagrams. Functionality is expressed using input/output relations, and behavior is expressed via control flow. Abstraction and refinement are supported, and execution semantics are based on those of transaction graphs. The authors report that a system for supporting transaction graphs is in the implementation stage.

E-L provides an understandable notation useful for representing projects at an abstract level. However, the graphical notation does not permit any way of instrumenting processes for measurement, and has no mechanisms for expressing constraints on process behavior.

SADT

The Structured Analysis and Design Technique (SADT) specifies a graphical notation for representing activities, data flow, products that control processes, and mechanisms [IMF90]. The flexibility of this notation makes it extremely expressive, and methods for abstraction and refinement are natural and easy to use. Basic components of the notation are activities and products. The products flow among the activities and can alternatively be inputs, outputs, control inputs, and mechanisms. This flexible treatment of products provides much of the expressive power of the notation. The primary objective of this visual formalism is communication. The SADT notation has no definition of execution semantics, and activity behavior can only be expressed using the labels attached to activities. The functional specification consists primarily of arrows that show data flow. Abstraction and refinement are supported.

Like the E-L approach, SADT is an abstract, understandable graphical notation. It has the same shortcomings for the purposes of this research as E-L. Measurement of products and processes must be expressed as a parallel activity to the relevant process or as an explicit measurement activity for products.

STATEMATE

STATEMATE is a commercial process-simulation system that is based on the state chart formalism. It has been used extensively at the Software Engineering Institute for process modeling [KR90]. The notation represents three modeling perspectives, namely the functional, behavioral, and organizational perspectives. These three perspectives are combined to yield a comprehensive process modeling technique. The STATEMATE system can perform analysis and prediction for models encoded using its notation. This system incorporates both graphical and textual notations. Execution of models encoded in STATEMATE is done for analysis or simulation. Objectives of the STATEMATE approach are understanding, analyzing, and planning software projects. The notation is a combination of visual formalisms and formal language, and the core paradigm is imperative execution. Behavior is specified as transitions among states, and functionality of inputs and outputs is represented by explicit "input" and "output" clauses in the notation. Abstraction and refinement are supported, execution semantics are defined, and a system for enactment exists.

The STATEMATE approach allows specification of a considerable amount of detail; it overlaps the requirements-level and design-level categories. Models created using STATEMATE are understandable and can be used to analyze projects. Also, the level of detail permits description of measurement activities.

2.3.2 Design-level notations

Design-level formalisms describe process coordination and interaction issues. Those notations classified here as design-level notations are HFSP, MELMAC, and MVP-L.

HFSP

The HFSP modeling notation tries to separate functional, behavioral, and enactment aspects of software processes [Kat89]. The basic components of the notation are activities and objects. This notation characterizes activities by their input and output, and represents activities as mathematical functions. The functions are defined hierarchically, and these hierarchical definitions are repeatedly refined until the leaves of this refinement tree represent primitive activities. This hierarchical definition is called an enactment tree. Objectives of the HFSP approach include producing clear and comprehensible representations and supporting machine-guided enactment. The HFSP modeling notation is a formal language. However, the language is not a derivation of any conventional, computer-programming language. The core execution paradigm is imperative execution. Behavior of a software process model is expressed using enactment trees, and functionality is specified using input/output relationships. Abstraction and refinement are supported, execution semantics are defined, and a prototype system for enactment is reported to exist.

The notation of enactment trees and the treatment of activities as mathematical functions doesn't produce the most understandable models. Also, the imperative paradigm used by HFSP diminishes this notation's usefulness for modeling projects, because constraints can only be expressed in a procedural way. No special measurement functions are defined.

MELMAC

The MELMAC system represents control flow, data flow, and project activities using the edges, channels, and agencies of a FUNSOFT net, an extended Petri net [DG90, Gru91]. Simulated execution of a process model involves choosing an initial marking for the net and then setting it in motion according to the rules of Petri nets. The main objectives of the MELMAC project are understanding and guiding software processes. The network formalism is called FUNSOFT nets, and the core execution paradigm is that of Petri nets. Abstraction and refinement are supported, execution semantics are defined based on Petri nets, and execution support exists.

Constraints on process execution are expressed in MELMAC using connections between places of the FUNSOFT net, which is a somewhat unnatural notation for software developers to review. A significant drawback of the MELMAC approach is the lack of an explicit, textual model of the processes, products, and resources. This deficiency makes it difficult to create and review understandable models. Although parallel activities (i.e., measurement activities) can be supported in a FUNSOFT net, the notation provides no way to use those data. Any data gained through measurement would have to be translated into a token in the Petri net, which is problematic.

MVP-L

The goals of MVP-L include gaining a sound understanding of a project's processes, as well as representing the collection and use of empirical data [RU89a, Rom91a, BLRV95]. MVP-L supports the construction of comprehensible specifications and designs of processes, products, and resources. It further supports the instantiation and enactment of these models for purposes of analysis, simulation, guidance, tracking, and feedback.

In MVP-L, a process view is the set of process models, product models, resource models, and attribute models involved in a project. The language supports:

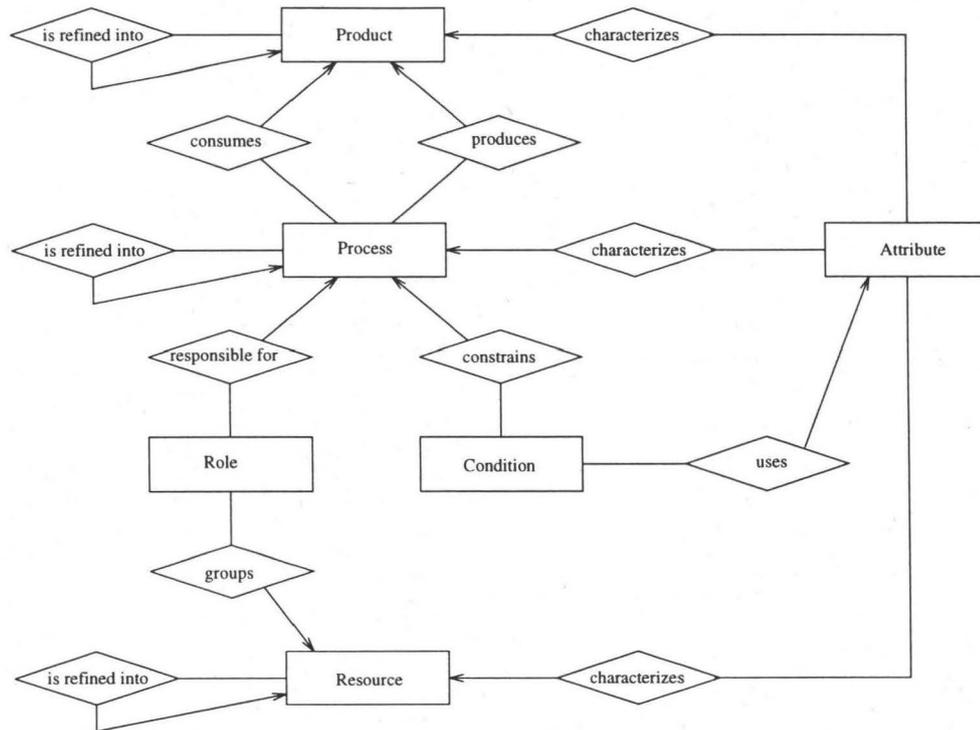


Figure 2.3: Basic MVP-L entities and relationships

1. Declarative process modeling. Each activity is explained in terms of a target condition, optimally using empirical data rather than an algorithm.
2. Modular definitions of processes, products, and resources. This allows all of these entities to be refined as needed, and encourages the reuse of models.
3. Empirical data representation. Attributes are used to characterize processes, products, and resources by representing measurable qualities of those entities.
4. Data collection. Both the “when” and “how” of collecting data values for the attributes, as well as the responsibility for doing so, may be defined.
5. Measurement-based goal definitions. Measurement data is used to define process start and end criteria explicitly and unambiguously.

Figure 2.3 uses entity-relationship notation to illustrate the constructs of MVP-L. The main entities are shown in rectangles, and their relationships are denoted by diamonds. For example, the arrow that runs from the entity “attribute” to the entity “product” through the relationship “characterizes” should be read as “(an) attribute characterizes (a) product.”

Execution of MVP-L is based on project state and state transitions. The behavior of processes is specified by entry and exit criteria (rules). Process inputs and outputs are specified by consume and produce clauses in process models. Abstraction and refinement are supported, execution semantics are defined, and an enactment support system exists. The collection and use of measurement data can be explicitly represented. Comparisons using collected empirical data can be incorporated into entry and exit criteria to influence the path followed by the project.

```

process_model Write_design () is
  process_interface
    imports product_model Requirements_document, Design_document;
    exports
    product_flow
      consume in1 : Requirements_document;
      produce out1 : Design_document;
              out2 : Feedback;
      consume_produce
    context
    entry_exit_criteria
      local_entry_criteria (in1.status = 'complete');
      global_entry_criteria
      local_invariant
      global_invariant
      local_exit_criteria (out1.status = 'complete') and
        (out2.status = 'available');
      global_exit_criteria
    end process_interface
  process_body
    implementation
  end process_body
  process_resources
    personnel_assignment
      imports resource_model Developer;
      objects Doe, Roe : Developer;
    tool_assignment
      imports
      objects
    end process_resources
end process_model Modify_Design

```

Figure 2.4: MVP-L process model for “Write_design” from DCT1 example

Figure 2.4 shows the process model for the “write_design” process from the DCT1 example that was shown in Figure 1.1 and discussed in Section 1.4.1. The products involved in this process are listed first, then the rules for beginning (“entry criteria”) and ending (“exit criteria”) the process are given. This process is not refined into any child processes, so the “process body” section of the model is empty. The model also shows that two staff members (“Doe” and “Roe,” both resources of type “Developer”) are expected to perform the work of writing the design. These are the users who would receive feedback. A user would keep a feedback system informed about his or her progress by sending a “process.start” event when he or she begins the process, and similarly would send a “process.complete” event upon process completion.

2.3.3 Code-level notations

Code-level notations support the representation and automation of fine-grained work details. This section surveys APPL/A, Marvel/MSL, and MERLIN.

APPL/A

APPL/A was created to satisfy the object management needs of process programming and to allow the execution of process programs [SHO89]. Therefore, APPL/A is often labeled a process-programming notation. A process program is a process model represented using conventional programming techniques. The stated purpose of APPL/A is representation of process programs at the code level, with the primary goals of process-program execution and process control. APPL/A extends the Ada language with features that provide for the storage and use of persistent data. The key extensions are relations, triggers, and predicates. Relations support persistent storage of data and object management. Triggers resemble Ada tasks; they act in response to the acceptance or completion of relation entry calls. Predicates are a mechanism for representing predicate logic expressions that apply to relations. APPL/A also defines a programmable interface to the persistent storage manager. The stated objective of this formal language is process enaction. The core language paradigm is an imperative execution model, behavior is controlled by triggers (rules), and functionality of inputs and outputs are expressed through the use of relations. Abstraction and refinement are supported only in the way Ada supports these principles. Execution semantics are defined, and enaction support based on translation into Ada is partially built.

As a programming language, APPL/A supports integrating models with measurement principles simply due to the flexibility of the notation. The imperative nature of the language permits the description of both collection and use of data, and constraints can be expressed using APPL/A predicates. However, the fine-grained nature of this language makes the creation of understandable process, product, and resource models extremely difficult.

Marvel Strategy Language

The Marvel approach uses the Marvel Strategy Language (MSL) to describe a rule-based system for activation of tools [KFP88]. Marvel is characterized by its authors as a software development environment, and emphasizes assistance of software processes over modeling them. Marvel is a useful tool for defining a software development environment because it can easily automate small, formal activities. The basic component of MSL is the strategy, which is a collection of knowledge that defines facilities appropriate to some target. Processes are modeled in MSL as rules that have preconditions and postconditions and appear as part of strategies. MSL is a formal language with the objective of execution control. The core language paradigm for execution is a rule-based execution model. The behavior of processes is specified using strategies, especially the rules incorporated as parts of those strategies. Functionality is described in MSL by parameters on rules. Abstraction and refinement are not supported, the execution semantics are defined, and support for enaction exists.

Marvel is a software engineering environment in which the system assists personnel according to its internal knowledge. As an assistant, it is ideally positioned to collect measurement data. However, the notation offers little support for using that data to influence the progress of the project. Also, because abstraction and refinement are not supported, it is impossible to create abstract, understandable models of complex processes using the notation.

Merlin

Merlin supports technical modeling issues such as interconnection of people and tools, resource assignment, tool invocation, and backtracking [PS90a]. Merlin provides rule-based information to users with limited tool invocation. This notation is based on Prolog and uses the forward and backward-chaining capabilities of Prolog as an execution mechanism. Forward and backward chaining provides information to users, and serves as a means for answering queries about the process model. Basic components of the Merlin notation are roles, activities, objects, and resources. Roles in Merlin represent groups of logically related activities. Software objects represent all artifacts produced during the process. Resources include both the process agents and their tools. This approach states objectives of understanding, communication, and execution guidance. Merlin's formal language notation is a modified Prolog. As in Prolog, the core language paradigm is rules. Behavior of activities is specified by the preconditions and postconditions on rules, and functionality is specified by rule parameters. Abstraction and refinement are not supported, execution semantics are the same as Prolog, and a prototype is reported to exist for enactment support.

Merlin offers similar benefits and suffers from similar problems as the Marvel approach.

2.3.4 Summary of process modeling notations

Table 2.2 summarizes the notations. In the column labeled "Measurement support" we state whether the notation supports measurement explicitly or whether measurement may be modeled like any other activity (implicit support).

Notation	Expressive power (level)	Execution semantics	Measurement support
E-L	requirements	Transaction graph	implicit
SADT	requirements	none	implicit
STATEMATE	req./design	state charts	implicit
HFSP	design	imperative	implicit
MELMAC	design	Petri nets	implicit
MVP-L	design	rules	explicit
APPL/A	code	imperative (Ada)	implicit
MARVEL/MSL	code	rules	implicit
MERLIN	code	rules (Prolog)	implicit

Table 2.2: Summary of example notations

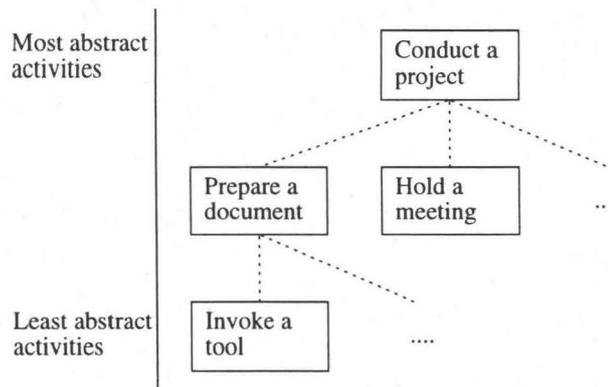


Figure 2.5: Abstraction levels of work activities

2.4 Measurement Support in Process-Centered SEEs

This section surveys research on process-centered software engineering environments (SEEs).¹ As in previous surveys [Rom89, Lot93], the trend towards collecting and using empirical data in these systems is emphasized, and details about several research and commercial systems are given.

Many of systems surveyed here work towards the ideal of integrating process definition, document construction, and measurement capabilities into a single system that can generate feedback for developers concerning their work activities. Figure 2.5 shows project work activities at three different abstraction levels, ranging from the coarse-grained activity of conducting a project to the fine-grained activity of invoking a tool on a computer. Feedback may be provided for activities at any of these abstraction levels. An ideal system might inform software developers about possible choices of methods based on a process model, or might give developers feedback about their work based on empirical data (the middle abstraction level in Figure 2.5). It may be possible to mediate access to documents and tools stored in a computer through a process model and the use of empirical data. Much activity has focused on using a SEE to help developers with in their activities on the computer (the lowest abstraction level in Figure 2.5).

We offer a scenario of how an ideal system might be used. Assume that a project has completed a portion of a software design document and that a design inspection must be conducted. Assume further that the ideal SEE has a representation of the project's work processes, the project's different design inspection methods, the criteria for applying each inspection method, and automatic data collection capability for the design document. After the SEE collects data from the design document, the data could be used to generate guidance for personnel about the most suitable inspection method. The benefits of automating data collection and other measurement activities are expected to include reducing the cost of collecting and using the data as well as increasing the validity of the data.

¹Portions of Section 2.4 and subsections are reprinted from the *Journal of Software Engineering and Knowledge Engineering*, 4(3), C. M. Lott, "Measurement support in software engineering environments," pp. 409-426, Copyright 1994 with kind permission from World Scientific Publishing Co. Pte. Ltd., Farrer Road, P. O. Box 128, Singapore 9128, Republic of Singapore. Not to be further copied without the publisher's specific permission.

2.4.1 GQM plan for the survey of process-centered SEEs

This section presents the GQM used to structure the survey of SEEs.

Goal:

Analyze **process-centered software engineering systems** for the purpose of **characterizing them** with respect to **collecting and using empirical data** from the point of view of **the researcher** in the context of **the literature**.

Questions: The following questions refine the goal.

1. What role does the SEE play in the software evolution process?
2. How sensitive is the system to work processes?
3. What role does measurement play in the system?

Metrics: Each of the previous questions is further refined into three metrics that help characterize the systems surveyed here.

- 1.1 The target user of the system (requirements engineer, designer, university student, software developer, ...).
- 1.2 The software evolution activities that are supported (requirements analysis, design, coding, testing, ...).
- 1.3 The construction tools provided by system (none, access to tools, diagram editor, compiler, ...).
- 2.1 Flexibility of work processes (assumed (implicit), fixed, variably definable).
- 2.2 The underlying enactment model for work processes (none, rules, ...).
- 2.3 Purpose for using process models (understand, control, ...).
- 3.1 Definition of measurement data (none, predefined, variable, tools, ...).
- 3.2 Objects that are measured (products, processes, depends on tools, ...).
- 3.3 Use of data (none, characterization, feedback, planning, ...).

2.4.2 Survey of existing systems

The GQM plan from the previous section is used to structure the presentation of the research prototypes and commercial SEEs given below. Static and dynamic code analysis tools such as code complexity analyzers are not covered. The systems are presented in approximate chronological order of publication.

Ginger

The Ginger system consists of a set of monitoring and feedback tools designed to record and improve programmer productivity during coding activities [MIK⁺87, KMKT90, Mat90, MKKT93, TMK95]. In Ginger, the focus is on measurement of on-line activities. Data is collected from these activities and from code products unobtrusively and automatically. The data thus collected is evaluated and stored in a database. Real-time feedback is provided to the system's users based on the collected data and historical baseline models.

Ginger assumes that users repeatedly perform a process consisting of editing source code, saving the file, compiling the code into an object file, and running the object file to test the program. The data that are collected include calendar time, terminal time, number of command executions, and CPU time consumed. Additionally, the source-code file is sampled at 5-minute intervals to capture the number of changes made since the last sample. The system treats changes as error removals, assuming that repeatedly editing the same file will result in a final, error-free version. Programmers who use the system can ask it to compare the data gathered from their work to historical baselines for their organization, and thereby obtain immediate feedback about their work. Feedback is primarily based on productivity data and baselines.

Ginger uses a fixed process model, and it provides no construction tools for its users. Ginger's target user is the programmer (metric 1.1), only the coding activity is supported (1.2), and it provides only access to construction tools (1.3). The system assumes a fixed edit-compile-test process (2.1), is unable to model work processes (2.2), and does not use a model of work processes for any purpose (2.3). Data definitions are fixed in the system (3.1), data is automatically collected by watching the programmer's code file and commands (3.2), and Ginger uses the collected data to evaluate the process and give feedback based on predefined baseline models (3.3).

Amadeus

The Amadeus system was originally developed in the Arcadia Project [SJM⁺89, SPSB91, Kad92].² The objectives of the Amadeus system include integrating measurement with process enactment by presenting an abstract interface for data collection, data analysis, and interpretation. This abstract interface attempts to isolate the primitives needed for using empirical data within a process-centered SEE. As originally developed, Amadeus made extensive use of classification trees as its basis for data analysis and interpretation capabilities [PS90b].

The objective of presenting an interface specialized for data collection and use means that the Amadeus system offers neither a process modeling component nor development tools. Instead, by working in cooperation with a process-centered software engineering environment, Amadeus can augment that system with sophisticated data collection and analysis capabilities. The process system is required to deliver notifications of process events to Amadeus. Depending on the event and its internal status, the Amadeus system reacts to the events by triggering data collection, analysis, and interpretation agents. Each event must be specified by the system's users. Possible reactions to an event include collecting data from users via forms or invoking a data-collection tool. Because of the system's independence from tools, Amadeus is not restricted to any life-cycle phase, product, or process, and its measurement capabilities are fully configurable.

²Commercially available from Amadeus Software Research Inc., 12 Owen Court, Irvine, CA 92715, USA; E-mail: amadeus-info@amadeus.com.

Internally the Amadeus system is controlled by scripts. Each script consists of an event, a guard, and an agent. Briefly, when the event occurs, the guard is evaluated, and the agent is triggered if the guard evaluated to true. The basic system can generate clock events and monitor files in a directory for changes. Additional events, especially process events, are defined in conjunction with a cooperating process system. A guard is simply any boolean expression defined using attributes of the events. The agent is any software tool or program that can be invoked on the host system. Amadeus was implemented so that it can be used in a distributed manner. A single interpreter runs on a server host. Clients can connect to the interpreter to supply data, request data, and pass events.

The commercial product extends the research prototype with tools for gathering measures of size and other statically determined properties of source code [Ama94]. The supported languages include Ada, C, and C++. Also included in the commercial product is a repository for collected data, support for importing and exporting data to/from the repository, and facilities for generating graphs and reports from data in the repository.

Amadeus provides no construction tools of its own and does not support defining a process. The target user of the system is the SEE builder (metric 1.1), any software evolution activity can be supported (1.2), and the system provides no construction tools (1.3). Work processes are assumed to be defined by a cooperating process system (metrics 2.1, 2.2, and 2.3). The system can accept any data definition (3.1), collect any data that can be collected using on-line tools (3.2), and the data use is entirely up to the person who writes the scripts that control the system (3.3).

ES-TAME

The ES-TAME system is a prototype of an expert system to support the design process for real-time software [Oiv90, OB92]. ES-TAME supports the design process in that the work processes to be performed can be represented in the system, the quality models for the products can also be represented, and these two sets of models can be integrated such that they function together. These models are linked based on metrics defined at each model's lowest refinement level. In addition, both products and processes can be measured. Although the prototype's knowledge is focused on real-time design methodologies, no design-related CASE tools are part of the system.

A sophisticated, highly structured framework supporting the Goal Question Metric Paradigm (GQM, see also Section 2.2.5) is built into ES-TAME. Templates assist the user in writing a GQM goal. The user can then select from a predefined set of questions and metrics, or can write new questions and metrics to refine the goal. Reasoning on the part of the expert system is used primarily when constructing the GQM plan. A rule-driven GQM generator uses forward chaining to guess elements of the GQM plan under construction. The system is intelligent enough to ask the user for data and to look in existing quality models to obtain those data when possible.

The GQM paradigm guides the representation of quality models in ES-TAME. Further, the GQM paradigm is also used to answer questions and interpret data collected from work products. When the user wants to evaluate a GQM plan, the system can automatically supply the user with data for those metrics where data exists. The system will further assist the user in answering the questions based on the metric values and interpreting the results in terms of the goals. Although work processes are modeled in this process-centered system, it is unclear how the system is made aware of progress made on the work represented therein.

The ES-TAME system's target user is a real-time software designer (metric 1.1), software design is the primary supported activity in the prototype (1.2), and although it provides no software construction tools (1.3), the system does provide an editor for writing GQM plans. ES-TAME supports the variable definition of processes (2.1), work processes are defined using an object-oriented knowledge representation (2.2), and the process definitions are used to guide the user during work processes (2.3). Although the prototype system focuses on the design activity, it is completely flexible about defining data (3.1) and collecting data from on-line artifacts (3.2). In addition to using the data for feedback and improvement, metric capabilities in the system, especially the support for developing GQM plans, may be used for planning (3.3).

Provence

Provence is a system for monitoring and visualizing software development processes [KB93]. It uses advanced hooks into the file system of its host computer to do so. Provence demonstrates how a number of special-purpose tools can be integrated with a process-centered system to form a process-centered SEE. Their process-centered system is Marvel, a rule-driven system for process modeling and enactment [KBS90]. Processes are monitored and enacted based on their representation as Marvel rules, and any work activity performed on the computer can be supported.

Provence focuses on visualizing process data. These data include process status and progress as well as process and product metric values. A special aspect of this system is the component called the 3D file system, which lets it monitor all accesses on certain files or in file system directories. This capability allows the system to collect data unobtrusively as well as to collect data from all tools that use the standard file system. From this capability, the Provence system derives a genuine openness and flexibility for collecting and using data from any on-line products.

The Provence system supports the variable definition of processes. The target user is any software developer or manager (metric 1.1), any software evolution activities can be supported (1.2), and it provides no construction tools nor access to tools (1.3). Provence allows a process to be variably defined (2.1), uses the Marvel strategy language to describe and enact processes (2.2), and uses process models to facilitate data visualization (2.3). Because the Provence system offers no support for defining metrics, the definition, collection, and use of data is left to the tools that it invokes (3.1, 3.2, 3.3).

SynerVision

The SynerVision system is a commercial product that supports defining, guiding, and enforcing a process for software development activities [Hew94, Dia94]. The definition of a project leans toward a loose collection of tasks rather than a tightly coordinated process view. A basic difference between this system and others surveyed here is that developers can begin to use SynerVision with essentially no task definitions, and then populate the tool with their views of their tasks. In addition to specifying dependencies on other tasks and specifying hierarchies (i.e., refinements) of tasks, users may attach text explanations to the tasks that they define. Task definitions can also be easily modified while in execution, but the system performs no analysis of the impact of those modifications. Of course, SynerVision also supports providing developers with information about a predefined process.

The system automatically measures the time that a person spends enacting a task. All other data collection and use of data in dependencies is supported by the tools, not the system. SynerVision is controlled by process scripts and process templates. A process script is a Unix sh(1) shell script, specially extended for SynerVision, and lists the tasks that should be performed. More precisely, a script causes tasks to be instantiated in a developer's working context. Where a script specifies, for example, which developer is responsible for a task, a template provides a placeholder for the information about which developer is responsible for a task. Templates facilitate the reuse of process descriptions, and can be invoked by scripts at run time, which offers the possibility of very late binding of resources (e.g., personnel) to tasks. Dependencies in the process scripts express the order in which tasks can be performed. A dependency rule can additionally invoke a tool to check whether a requested action (e.g., marking an activity as complete) is legal. Users can also define many different views on their own collections of tasks, based on the priority, status, etc. of those tasks. For example, a task status can take on the values "completed," "abandoned," "in-progress," "new," or "ongoing."

The SynerVision system supports definition of a process and offers its users an environment in which they can work. Although part of the Hewlett Packard "SoftBench" product, SynerVision can also be used alone. The target user is any software developer or manager (metric 1.1), any software evolution activities can be supported (1.2), and although it provides no construction tools, users may access tools from SynerVision via the SoftBench Broadcast Message Server (1.3). SynerVision allows a process to be variably defined (2.1), uses its own script language to describe and enact processes (2.2), and uses process models to define, enact, and enforce a process (2.3). The system offers strong support for collecting data about the time spent on a task, but otherwise the definition, collection, and use of data is left entirely up to the tools with which it communicates via the Broadcast Message Server (3.1, 3.2, 3.3).

Ceilidh

The Ceilidh system is a quality control system for teaching students how to develop programs according to specifications and to quality standards [BBF93]. Ceilidh (pronounced "cay-lee") evaluates the result of coding activities and reports on the evaluations. Like all systems for measurement-based guidance, it is critically dependent on the models used as the baselines for comparison with the collected data. In the Ceilidh system, the baseline is based on a sample solution to the problem. One drawback is that the baseline is not explained to the system's users until after they have submitted a solution. In other words, measurement-based feedback is offered very late in the cycle.

Students use Ceilidh to write, test, and evaluate their coding assignments. The system compiles and tests assignments that are submitted to it. It then compares the submitted source code and test results against established quality standards, and reports to the student on the result. Quality standards are defined in Ceilidh in terms of correctness (determined by testing), run-time efficiency of the program, as well as structure, complexity, and style of the source code. Figure 2.6 shows an example of the feedback provided by the system to students after they submit a solved exercise. Ceilidh additionally provides students with course notes, examples, program requirements, and access to editors, compilers, and test support.

Ceilidh's automatic assessment facility is responsible for grading the student's assignments. A necessary part of the assessment facility is a test oracle that determines whether output from student programs meets the specification or not. Regular expressions play a large role in matching

```

SubmitProg
File for Unit 2 ex pay suffix c copied to coursework directory
Mark for Unit 2 ex pay copied to coursework directory
Analysis of Dynamic Correctness
      item: mark:out of: lost
      Simple test: 40: 40: 0
      Next test: 30: 30: 0
      Check text too: 30: 30: 0
Score for Dynamic correctness is:100 out of 100

Mark summary
      category: mark:out of
      Dynamic correctness: 60: 60
      Features: 5: 15
      Typographic: 23: 25
Overall mark awarded : 88: 100

Ok

```

Figure 2.6: Feedback provided by Xceilidh

the test results with the expected output, thereby lifting any requirements of specifying program output with the precision of what text must appear in what column. Test coverage tools are used to detect useless code in a student's program, static source code analyzers check complexity, and the UNIX tool *lint* is used to detect structural weaknesses.

Ceilidh, like Ginger, offers no workbench-like tools, But unlike Ginger, Ceilidh installations can define their own grading criteria. The target user is a student learning how to program (metric 1.1), only the fixed process of editing, compiling, and testing is supported (1.2), and although the system offers no construction tools, it does offer access to tools in the operating system's environment (1.3). The system assumes an implicit process (2.1), and no other work processes can be defined or used (2.2, 2.3). The metrics used to evaluate student's programs are fixed; they can only be redefined by changing the assessment engine (3.1). The data is collected by analyzing, compiling, and running code artifacts (3.2), and the data is used to evaluate work products and provide feedback to the system's users (3.3).

ProcessWeaver

ProcessWeaver is a commercial product that supports the organization and coordination of work done by groups of individuals [Cap92, Fer93, Chr94]. The Weaver system uses a collection of representations to define a comprehensive process model. The various representations that model and support the enactment of complex processes are manipulated by graphical interfaces specific to each type of representation. Although the representations (and corresponding support tools) are structured in a strict hierarchical manner, their names do not reflect the hierarchy well. At the top of the hierarchy, a collection of methods (supported by a "method editor") describes the work breakdown structure for a software evolution project. Each method is then refined into an interface (called an "activity," supported by an "activity editor") and an implementation (called a "cooperative procedure," supported by a "cooperative procedure editor"). The interface ("activ-

ity”) describes the input and output products, and the implementation (“cooperative procedure”) defines the subprocesses that perform the work of that method. Hierarchical definitions are permitted in that elements of cooperative procedures can also be cooperative procedures. At the bottom of the hierarchy, the element within a cooperative procedure that describes a concrete task (atomic process step) is called a “work context.” Work contexts bundle all the elements needed to perform an atomic process step, namely the task description, the products to be consumed or produced, and the tools needed to perform the task.

The end user views, manipulates, and delegates his or her collection of work contexts using a tool called the “agenda.” Facets of processes that cannot be captured by the graphical representations or implemented using Petri net rules are implemented in a command-interpreter (shell) language. This language supports the delegation of work contexts and the invocation of tools from the operating system environment. The use of shell scripts to implement facets of a comprehensive process model makes the ProcessWeaver system especially flexible, but it also makes the process model correspondingly more difficult to comprehend as a whole.

All communication between cooperative procedures, the agendas of different users, and the other tools of the environment is done via messages that are sent to a broadcast message server. Components foreign to ProcessWeaver such as measurement tools can be integrated into the system. On-line forms for measurement can also be integrated conveniently by adding the appropriate work contexts (i.e., a work context that requests a person invoke the on-line data collection form) to the related cooperative procedures. Both synchronous and asynchronous data collection can be implemented in ProcessWeaver simply by building the Petri nets accordingly.

Weaver offers no software evolution tools. The target user is the software developer, although significant support for developing process models is part of the system (metric 1.1). Any activities may be supported (1.2), and the system offers access to tools of the supporting computer system (1.3). Work processes may be variably defined (2.1), and the variable models of work processes are based on the use of Petri Nets to describe coordination (2.2). Because the various models are entered using a number of editors, it is difficult to generate a comprehensive model of a project’s processes for review or understanding purposes. Therefore the models are primarily used to control the process (2.3). All data definition, collection, and use depends on the tools invoked from outside the Weaver system. Weaver provides no support for defining (3.1), collecting (3.2), or using (3.3) data to influence the processes.

Table 2.3 presents a summary of the systems that were surveyed here. The columns correspond to the metrics explained in Section 2.4.1.

2.4.3 Summary of process-centered SEEs

Many of the systems surveyed here address the problems of integrating software construction, process definition, and measurement capabilities into a single system. A classification scheme defined by Rombach is expanded and used to classify the integration of these three elements in the surveyed systems [Rom89]. Figure 2.7 graphs construction versus measurement components in SEEs. Figure 2.8 shows process definition versus measurement capabilities. Finally, Figure 2.9 charts construction versus process definition facilities. Included in these figures is an estimate of where MVP-S, the system described in Chapter 4, fits into this classification scheme.

There is considerable progress towards the upper right regions in Figures 2.7 and 2.8, meaning

Table 2.3: Summary of software engineering environments

	Metric 1.1 Target user	1.2 Evolution activity	1.3 Constr. tools	2.1 Work processes	2.2 Process definition	2.3 Use of proc. def.	3.1 Data definition	3.2 Data collection	3.3 Use of data
Ginger	pgmr.	coding	access	fixed	n/a	n/a	fixed	code	feedback
Amadeus	SEE bldr.	any	none	SEE	SEE	SEE	variable	variable	variable
ES-TAME	designer	design	none	variable	kwn. rep.	guide	variable	variable	plan
Provence	s/w dev.	any	none	variable	Marvel	visualize	tools	tools	tools
SynerVision	s/w dev.	any	access	variable	scripts	guide	tools	tools	tools
Ceilidh	student	coding	access	fixed	n/a	n/a	fixed	code	feedback
Pr. Weaver	s/w dev.	any	access	variable	graph. rep.	guide	tools	tools	tools

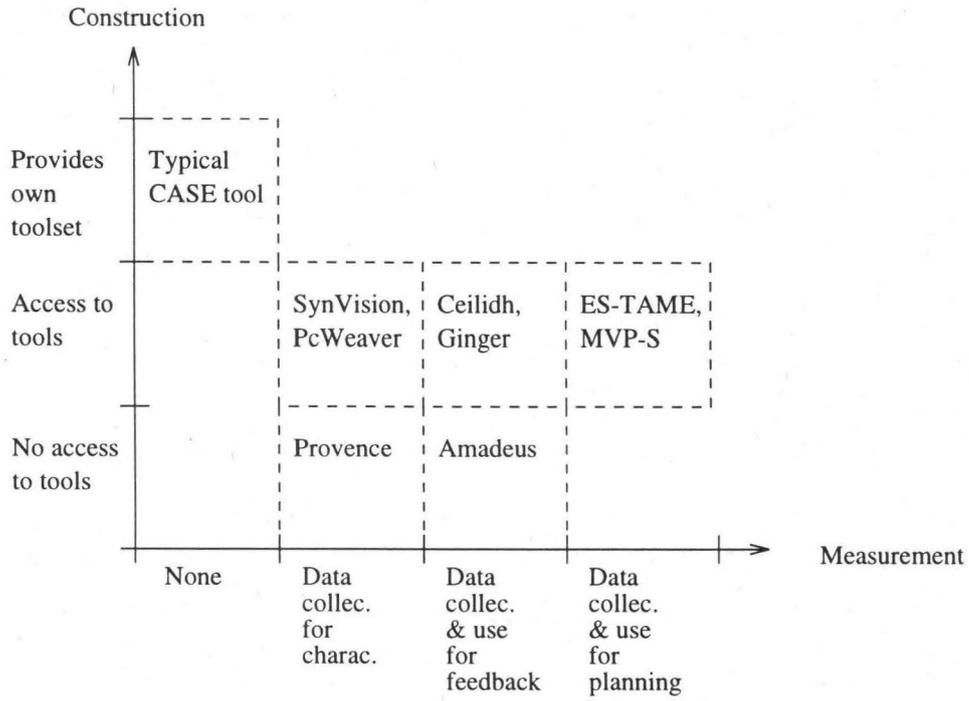


Figure 2.7: SEE components: construction versus measurement

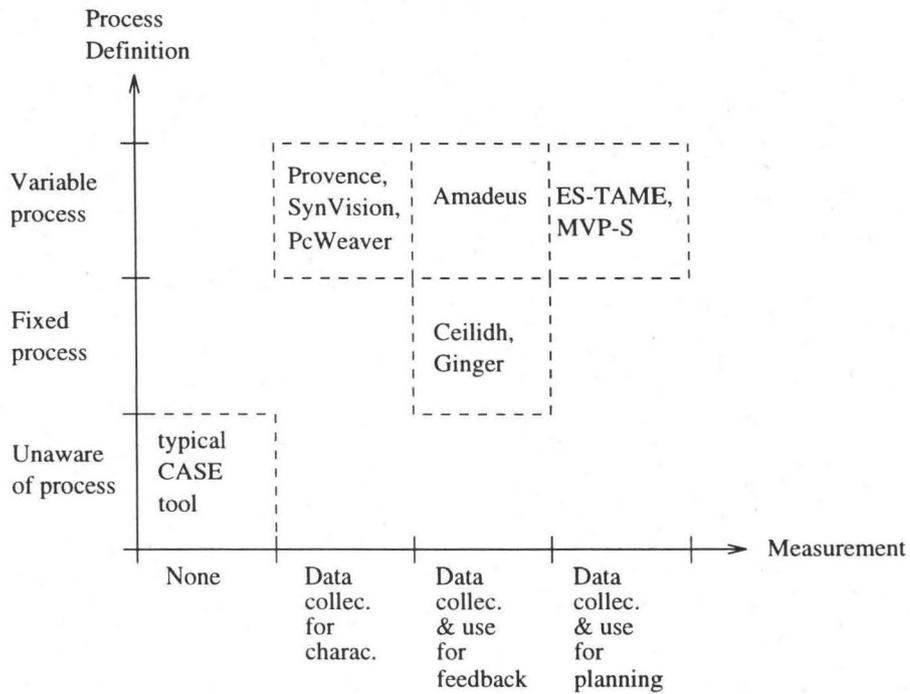


Figure 2.8: SEE components: process definition versus measurement

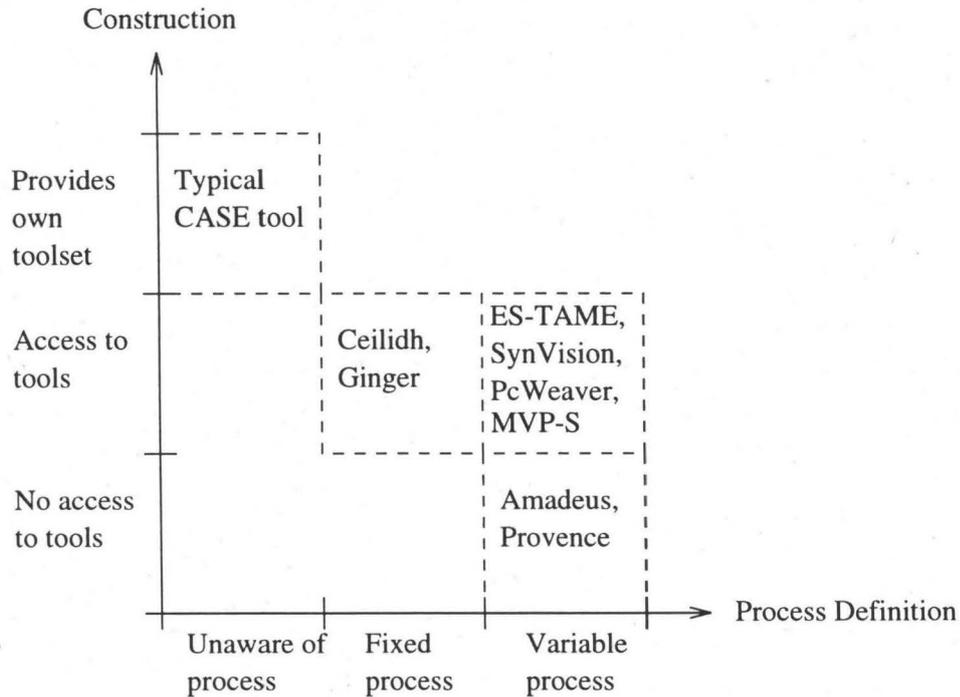


Figure 2.9: SEE components: construction versus process definition

that some systems support both construction and measurement or both process definition and measurement, respectively. In Figure 2.9, the upper right regions describe systems that provide their own tool sets and also support process definition. Efforts to reach the upper right regions of Figure 2.9 will most likely only be successful if the system designers pay close attention to interfacing with existing tools rather than attempting to replace them.

2.5 Empirical studies of software engineering environments

Empirical work on evaluating various techniques and methods in software engineering is not common [Gla94, TLP95]. Evaluations of software engineering environments are even less common. However, this relatively new technology has begun to attract considerable attention, process-centered environments are commercially available (see also [Lot94]), and some work has been done to evaluate this new technology.

Christie reports on a comparison of ProcessWeaver and SynerVision [Chr94]. The study's goals were limited to evaluating issues concerning the tool's ease of use as viewed by both the process developer and the end user (e.g., development of the process description, screen layout, response time). Issues of adoption among the end users (e.g., intrusiveness and impersonality of on-line guidance) were also evaluated. However, neither the amount of time that users required nor the quality of their results were evaluated.

The study compared and contrasted aspects of the two systems in the context of a small experiment that was performed once for each system. In the experiment, a team of three people cooperated on a simple process using a process guidance system. The process involves three people who assume two roles, a manager and a technical editor. The person in the role of the

manager supervises the work of editing and reviewing a document that is done by two people who both assume the role of a technical editor. In the scenario, the editing job is delegated, refused, and finally assigned. One editor edits the document; the other editor reviews the edited document, and finally the document is approved. All interactions between the people are supported by the system.

Some discussion is given about the difficulty of bringing about a change in the culture of a software development organization, change that is a prerequisite for the use of a guidance system. The conclusion raises important questions, including the following:

- “Will resistance to working in an automated process be a major impediment?”
- “Will process automation products be sufficiently flexible . . . to model industrial-strength projects?”

2.6 Summary

This chapter presented many measurement and process technologies, and showed that there are extremely few attempts to link the two technologies. This chapter also revealed that some process-centered software engineering environments support the collection and use of empirical data. The use of empirical data in process-centered software engineering environments offers great opportunities for providing guidance and feedback to users about their work. Finally, some empirical work on evaluating the usefulness of these environments was presented.

In the next chapter, an integration of measurement and process technologies is presented that makes extensive use of the GQM paradigm and the MVP-L process modeling notation. Chapter 3 also presents a set of requirements for an automated system that will use integrated measurement and process views to present guidance and feedback to personnel.

Chapter 3

Foundations of Measurement-Based Feedback

This chapter discusses the foundations of measurement-based guidance and feedback. Section 3.1 presents an overview. Section 3.2 discusses the information required to integrate measurement and process views, and introduces consistency rules for the integrated result. Section 3.3 gives an algorithm for integrating the two types of views, and Section 3.4 demonstrates the application of the algorithm with a comprehensive example. Section 3.5 discusses a small case study that was used to evaluate the integration approach proposed here. Finally, Section 3.6 states a series of requirements concerning automated support for providing measurement-based guidance and feedback based on the integrated views.

3.1 Overview

The integration of measurement and process views brings together two partial descriptions of a single software development project. These two views of a software project support each other strongly; they have been researched and applied separately only due to historical reasons. A measurement view describes data explicitly and references processes, products, and resources implicitly. A process view describes processes, products, and resources explicitly and references data implicitly. Integration of these views essentially makes explicit the existing, implicit overlaps and relationships between measurement activities and development processes.

The approach for integrating the two views uses the Goal Question Metric Paradigm (GQM) and the Multi-View Process Modeling Language (MVP-L). A measurement view consists of GQM plans that describe how some aspect of a software development project will be measured. A process view consists of information about the processes, products, and resources in a software development project. The idea of integrating these views began with the realization that they overlap *at least* in the representation of empirical data. Linking the representations of empirical data in a measurement view (metrics) and a process view (attributes) is shown in Figure 3.1.

The basic idea of integrating process and measurement views is embodied in the task of constructing a measurement plan. A measurement plan specifies precisely who collects what data, when it is collected, and how it is collected. However, constructing a measurement plan

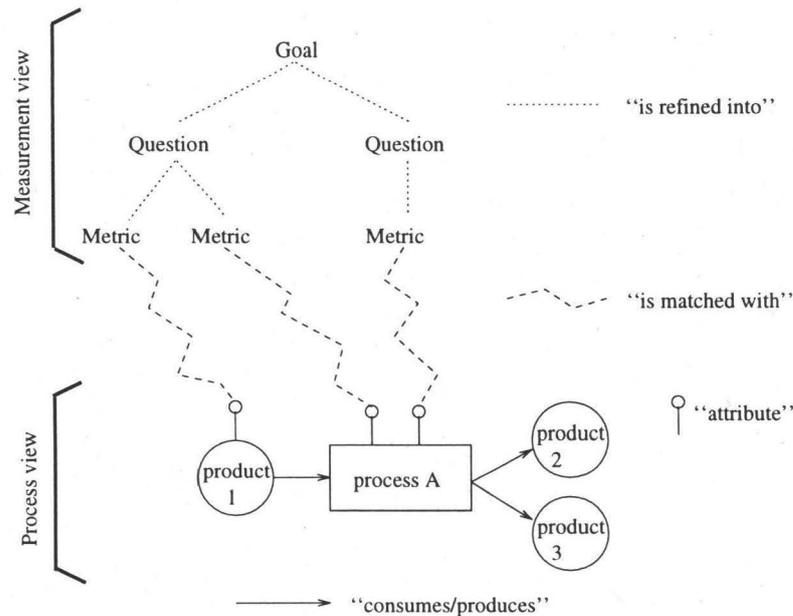


Figure 3.1: Linking metrics and attributes from measurement and process views

without explicit process models relies on implicit knowledge of the relevant processes, products, and resources. Without an explicit representation of that knowledge, the resulting measurement view cannot be reviewed for consistency problems, and is likely to suffer problems that will be detrimental to a measurement-based process improvement effort.

Integrated measurement and process views offer many benefits for planning a measurement and improvement program. First, the integrated views permit checking various consistency properties. The consistency properties define a target condition in which personnel can use an integrated set of views to collect useful data, interpret that data appropriately, and judge the suitability of such views for providing measurement-based feedback to software developers. Further, integrated views support the automation of data-collection activities, and enable the generation of measurement-based feedback using an on-line system.

This chapter also states four sets of requirements that a process-centered software engineering environment must possess in order to support measurement-based feedback. First, the use of tools to collect and use data is discussed. For example, a software engineering environment must be able to invoke on-line tools to collect data from documents and on-line products. Second, interactions with users to collect data are sketched. Users must be queried in response to various process events and milestones. Third, issues in providing measurement-based feedback to personnel who play technical roles are presented. For example, the environment must be explicitly aware of the processes and track their status in order to generate feedback. Finally, support for personnel who play management roles is discussed. These personnel require support for supervisory duties such as assigning personnel to processes.

3.2 Integration of Measurement and Process Views

This section introduces the integration of measurement and process views. Section 3.2.1 gives requirements for the information needed from measurement and process views. Next, Section 3.2.2

discusses the reuse of existing formalisms, and Section 3.2.3 states the information used from those formalisms. Finally, Section 3.2.4 addresses limitations of the integration approach and some open issues.

3.2.1 Requirements for integrating measurement and process views

This section states requirements on the information needed from the view of measurement activities, from the view of software development processes, and on the integration model that will link the views together.

Information required about measurement activities

A measurement view consists of GQM plans and additional information that coordinates measurement activities and assists with interpreting the resulting data. The entire view (and thus the measurement program) should be based on the explicit statement of goals. Information about the following items that describe “how well” various processes are performed is required from a measurement view for the purposes of integration:

- The goals of the measurement program.
- Rationale for the choice of metrics.
- A precise definition of the metrics for which data must be collected (may include the scale, units, etc.) For example, definitions of “effort” or “lines of code” must be stated.
- Procedures by which the data are collected.
- Relevant quality models (existing baselines, if any).

Information required about software development processes

A process view consists of information about a set of software development or maintenance processes. Information about the following items that describe the “what” and “how” of software development are required from a process view for the purposes of integration:

- The input, output, and intermediate products.
- The processes that transform input into output products.
- The resources available for performing and supporting the processes, including a definition of what personnel are responsible for what steps.
- Identification of milestones (events) in the processes.
- The criteria that define when a process may begin and when a process was successful.

Requirements for the integration model

Pooling the information common to measurement and process views permits analyzing the integrated result for inconsistencies that would be detrimental to a measurement-based process improvement program. Example inconsistencies are attempting to collect data from a process that doesn't really exist, or collecting data from a product at the wrong point in that product's life cycle. The additional information required to link the views is thus a statement about how the views overlap and how their common parts can be linked with each other. The integration activity may also augment a process view with a number of quantitative statements about how well various activities are expected to be performed. The requirements for the model include the following:

- Take the improvement goals (gaining understanding or providing feedback) into account.
- Identify processes, products, and resources common to both views (identify all possible connection points).
- Link data sources with data needs (outputs with inputs), thereby satisfying needs for pooled information in both views. A data source may be either a model or an object from the project. This means that information can be gathered either from a model (at any time) or from the project (only during execution).
- Link metrics (what to collect) and data-collection procedures (how to collect it) from the measurement view with events (when to collect it) and personnel (who collects it) from the process view.
- Consider the need to measure repeated instantiations of the same entity (e.g., effort from repeated instantiations of the same process, experience of multiple developers, size of code modules, etc.).

Consistency properties. The consistency properties for an integrated set of views must either prevent or enable the detection of problems so that the following conditions hold:

- All types of references in the measurement view to processes, products, and resources can be unambiguously linked with a description of the corresponding process, product, or resource in the process view.
- The measurement view makes no demands for data from nonexistent processes, products, or resources.
- No data is collected that is not driven by a measurement goal or needed to generate feedback to a process (i.e., no values are immediately consigned to a data graveyard).
- If in accordance with the goals of the modeling effort, data are used to provide feedback.

3.2.2 Reuse of existing formalisms

The requirements stated in the previous section are satisfied through the reuse of existing research. The Goal Question Metric Paradigm (GQM) is used to structure measurement views. As discussed

in Section 2.2.5, a measurement view based on the GQM paradigm consists of a goal, a supporting set of questions, and finally a set of metrics. The Multi-View Process Modeling Language (MVP-L) is used to structure process views. As discussed in Section 2.3.2, a process view based on the MVP-L formalism consists of a set of process models, product models, resource models, and attributes.

Goal Question Metric Paradigm

The survey of existing measurement frameworks presented in Section 2.2 showed that only one measurement framework was developed to take explicit consideration of process issues, specifically the GQM Paradigm. However, GQM lacks a formal notation for the definition of processes and their products and resources. The GQM paradigm's explicit support for process issues makes it an excellent candidate for reuse in later discussions of linking measurement and process. Because GQM offers much that can be used without change and thereby can minimize the amount of work necessary, GQM will be reused.

A GQM plan may be augmented with additional information that specifies the data-collection method and other information for directly collectible metrics. This information is sometimes named a "measurement plan" [Hoi94, GHW95]. For example, the GQM metric "effort" (defined as hours) might eventually be stated as "the effort that Jane spent in week 30 as collected on form F12." A measurement plan will also include the data-collection forms that personnel will complete in the course of their work. Traceability between a GQM plan and a measurement plan is an important issue, but relatively straightforward to define. Essentially, each metric in a GQM plan must have corresponding information in a measurement plan to state who collects the data, how, and when.

A comprehensive measurement view of a software project is thus comprised of a GQM plan as augmented by a measurement plan. Essentially each GQM goal leads to the definition of a single measurement view. The combination of multiple measurement views is a natural one due to the common overlap in questions and metrics among related GQM plans, but it is not addressed in this research.

The Multi-View Process modeling language

The survey of existing process modeling notations in Section 2.3 showed that only one takes explicit consideration of measurement and data collection issues, specifically MVP-L. This notation supports the execution of a comprehensive set of process models to track the status of processes and products and to generate feedback for personnel. MVP-L additionally supports representing the collection of empirical data, and using the data to influence the path taken by processes. Although MVP-L offers no support for the top-down planning of a measurement program, or for the bottom-up interpretation of data in the context of goals, it does offer formal execution semantics and a supporting execution environment. Therefore, MVP-L is the most appropriate reuse candidate for this research.

3.2.3 Information that links the measurement and process views

This section identifies the linking points in measurement and process views.

Information used from the measurement view

A measurement view of a software project consists of a GQM plan, augmented with a measurement plan, as discussed in Section 3.2.2. The following discussion is in general limited to a single measurement view (one GQM goal), and the supporting questions and metrics.

First, there is the GQM goal from the measurement view. The intent of the goal (i.e., understanding or improving) must be declared. Further, the *type* of a goal's objects of interest (i.e., processes, products, or resources) must also be declared.

A GQM goal is refined using a series of questions. These questions are divided into three sets, namely the set pertaining to the object of interest (set O), the set for the quality aspect of interest (set Q), and the set for feedback (set F). A refinement of the goal in terms of these questions must be declared. The refinement of questions in terms of other questions must similarly be declared. The refinement of questions in terms of metrics must be similarly declared. The refinement of goals into questions, then questions into other questions, and finally questions into metrics within a GQM plan forms a directed acyclic graph.

All metrics from the GQM plan must denote values that can be collected directly (i.e., none are calculated based on other metrics). This restriction is minor, because metrics that would need refinement can be expressed instead as leaf questions. Further, the units of measurement must be declared for each metric. The scale of the data yielded by each metric must also be declared, for example in terms of the set {nominal, ordinal, interval, ratio, absolute}. Each metric must be marked as to whether it is an objective (i.e., repeatable) assessment such as "the schedule time in days" or a subjective assessment such as "the ease of use of a method." Finally, the data collection procedure must be declared for each metric. A data collection procedure states whether values are derived by asking a person, by invoking a tool, or by some other method. A forms-based tool used to query an individual for some data value would still be described as "asking a person."

Internal consistency of a measurement view. The following restrictions on a measurement view are necessary to permit integration with a process view. First, it is meaningless to have a GQM plan with neither questions nor metrics. Second, we must be able to determine the intent of a goal (understanding or improvement) as well as the type of object of interest (i.e., whether the goal focuses on a process, product, or other aspect). Third, no question and no metric may be left isolated, not related to any other questions or metrics. Fourth, questions cannot be refined in such a way that a loop would result (the refinement must be a directed acyclic graph). Fifth, the definition of the metric must make clear precisely what is desired. For example, the definition of "experience of personnel" might be measured by years of experience, self estimates, or performance on a test.

Finally, traceability in the refinement of goals into questions and so on is an important consistency property. For example, a question about cost per unit time that is only refined into a metric for time does not have proper traceability because the aspect of cost is ignored. This

deep concept of traceability in the information needs of various refinement levels is noted here to serve as a guideline and to identify an open problem.

Information used from the process view

The process view includes information about processes, products, and resources.¹ The refinement of objects in terms of other objects (e.g., a process is refined in three subprocesses) must be declared. Attributes are attached to objects of the process view to capture information such as the status or availability of that process, product, or resource. The attributes on each object must be declared. These attributes are also used in constraints on processes to describe possible paths (control flow) through the set of processes.²

The type of each attribute must be declared (any legal MVP-L type is allowed). Further, the source of data for a specific attribute must also be declared, in terms of whether the source is external or internal to the process view. For example, an attribute's value may be supplied by some external procedure (e.g., a person enters the value), or may be determined solely using other attributes. To determine how a particular attribute is used in the process view, each attribute must be declared as to whether it is used in the criteria of some process or in some other way. An attribute of any object in the process view may be used in the criteria of a process, so this rule is not specific to process objects.

Personnel are the fundamental resource of any software project. The set of people involved in the project must be defined. The mapping of personnel to objects in the process view captures the responsibility relationships between those personnel and the objects. This information is important for both processes and products. However, it is nonsense to link personnel with resource objects that model the personnel anyhow. Finally, a set of milestones (events) from the process view must be declared. Example events are beginning a process, completing some process, etc. These events are important for automated support.

Internal consistency of a process view. The following restrictions on a process view are necessary to permit integration with a measurement view. First, sets of process models, product models, resource models, attribute models, events, and personnel must have been developed. Second, the mapping of attributes with objects, the type of each attribute, the source for data for each attribute, and whether each attribute is used within process criteria must have been declared. Third, the responsibility for processes and products must have been declared via a mapping with personnel. Fourth, objects cannot be refined in ways that would cause loops in the refinement graph. Finally, processes can only be refined in terms of other processes, and the same holds for products and resources.

Linking information from both views

This section explains the links that are made to integrate measurement and process views, and what each link means. The links are separated into two groups. First, the views may directly

¹This discussion considers model-world representations of processes, products, and resources that mirror their real-world counterparts. We do not discuss models themselves because a model represents a class of objects.

²Attributes will be used to represent empirical data that describes the various objects. However, no sophisticated use of attributes is expected in a process view before it has been integrated with a measurement view.

overlap in that the same artifact is explicitly referenced in both. For example, the measurement view component "GQM object of interest" will also be modeled in the process view. Alternately, the views may have no direct overlap, but support each other. Essentially, implicit and explicit aspects are linked so that everything can be made explicit. For example, it may be helpful to declare responsibility for collecting data demanded by the measurement view using the resources modeled in the process view.

Direct overlaps. The following links consider artifacts that are explicitly addressed by both measurement and process views.

1. A GQM goal's object(s) of interest is linked with MVP-L processes, products, or resources. This establishes that the objects of interest are clearly identifiable in the process view. For example, if the GQM goal mentions a design review, that review process must be represented in the process view.
2. Data needs (GQM metrics) are linked with data sources (in MVP-L). A data source may be either a part of the comprehensive process view (e.g., a product model), or it may be an attribute of some model in the process view that represents data to be collected. One of the following two cases should hold for all metrics:
 - **Case I: Model as data source**
In this case, GQM metrics for which data can be collected from MVP-L models are linked with those models. For example, metrics that demand data about the type of design review process performed could be answered by the model of that design review process. There is no need to collect data from the project for these metrics. This attempt at linking views may encourage the process view to be extended.
 - **Case II: Project as data source**
In this case, GQM metrics that require data to be collected during the project are linked with MVP-L attributes. Attributes from the process view represent data that will be collected. The set of attributes that was developed as part of the process view may need to be extended for this purpose. For example, metrics that demand data about the time spent on the design review process could be collected during project performance.

Indirect overlaps. The following links consider artifacts that are implicitly addressed by one view and explicitly addressed by the other.

1. Each GQM question is linked with MVP-L processes, products, or resources. This establishes possible sources for data that will answer the questions. For example, questions concerning defects detected in the design review could be linked with the review process.
2. Each GQM metric is linked with an MVP-L event. An event in the process view models a point in time when something happens, such as the start or completion of a process. This link establishes the point in time when data for a metric will be collected. For example, data concerning the time spent on a design review process could be collected when that process completes.

3. Each GQM metric is linked with an MVP-L person resource. This establishes the responsibility for collecting data for each metric. For example, the moderator responsible for a design review process could be made responsible for collecting data from that process.
4. Each GQM question is linked with an MVP-L attribute. If a goal includes providing feedback to the personnel, then a source of information (evaluated data) is needed. Answers to questions serve as this source, and therefore this relationship is a reversal of previous relationships. Here, the measurement view is the source and the process view is the sink for information. Answers to questions from the measurement view can be plugged into attributes in the process view, where they may be used as feedback. For example, a comparison of historical defect data could be made at the direction of the measurement view, and the result could be plugged into the process view.

3.2.4 Limitations and open issues

Some limitations and open issues surrounding the approach for integrating measurement and process views are discussed here. First the limitations stemming from the use of MVP-L are discussed, then issues in mismatches of refinement levels are addressed, and finally some open issues in integration are sketched.

Limitations

We have assumed that it is possible to write measurement views that can be fully integrated with a process view. Stated differently, all aspects of the processes, products, and resources that are somehow referenced in the measurement view must be modeled in the process view. However, this assumption may impose severe restrictions on a measurement-based process improvement effort. For example, an aspect such as “organizational structure” might yield important information for a measurement view. However, it would be difficult to incorporate this information into a MVP-L-based process view. Therefore finding a mapping from GQM goals, questions, and metrics onto objects from the process view would be impossible.

Limitations on the integration approach can be defined by identifying what aspects of a software process can be modeled in MVP-L. Essentially, the integration model is restricted to the processes, products, resources, and properties of these objects that can realistically be modeled in MVP-L. The following list of questions may help resolve the problem of a measurement view that makes references to some object that apparently cannot be found in the process view:

- Can the object be modeled in MVP-L?
If yes, then the process view should simply be extended as needed.
- If the object cannot be modeled in MVP-L, is the object implicitly modeled (i.e., do the personnel have a deep understanding of the object)?
If yes, then a modeling approach other than MVP-L might be used to augment the process view.
- If the object apparently cannot be modeled, then can the data that are supposed to be collected ever be used sensibly?

If no, the problem disappears. Otherwise, some solution must be found.

- Given that the data can be used sensibly, can the data even be collected at all?

If no, then this aspect may only help to understand limitations on the measurement view, and in this case poses no problems for the integration. Otherwise, again some solution must be found.

Based on the limitation to aspects that can be modeled in MVP-L, the percentage of the measurement view that can be integrated with the process view can be determined. If only a percentage can be integrated, then only the portions for which integration is possible should be checked for consistency.

Mismatches of refinement levels

The integration information cannot resolve problems that result from modeling reality at different levels of abstraction. For example, if a measurement view is refined further than the corresponding process view, it may demand information about a not-yet-modeled process. If the process view is refined further than the corresponding measurement view, then not all parts of the process view can be linked with the measurement view. The first case (process view is not refined sufficiently) is a genuine problem, and provides sufficient motivation for another iteration through the activity of modeling processes, products, and resources. The second case may be a problem. It may indicate that the scope of the modeling effort was chosen inappropriately, or that the developers of the measurement view neglected to measure important activities.

The issue of multiple instantiations of a single process or product is related to the issue of different refinement levels. A case that must be considered is one in which a single metric will lead to collection of data from multiple instantiations of a process or product. For example, a metric may require the collection of data for the sizes of all developed code modules, for the amount of effort expended on repeated iterations through the same process, or for the years of experience for all the software personnel. The measurement plan is generally expected to specify the additional details that resolve this issue.

Open issues in integration

Some open issues in the integration of measurement views and process views are briefly sketched. The resolution of these issues would significantly extend the approach for integrating measurement and process views.

The first issue is the possibility of mapping portions of a measurement view with the *models* in a process view. Because a model represents a class of objects, mapping any measurement-view entity onto a model could mean that all instantiations of that model would inherit the mapping. For example, mapping a question onto a process model would mean that all processes instantiated from that process model are members of the mapping from the question. Both the power and the consequences of such a mapping deserve further investigation.

The second issue is the inclusion of roles and their viewpoints into the integration model. A role is understood here to be a projection (in the database sense) of a process view. For example, the view seen by a person who plays the role of a tester might include only those products,

processes, and resources relevant to the activities required of that role. A role from the process view appears to be strongly related with the viewpoint as stated in a GQM goal.

Finally, the case in which either of the views can be further refined and then re-integrated with the other *during* project performance is not considered in this research. It is impossible to discuss mapping metrics onto attributes of parts of the process view that are unknown at the time of performing the integration. Similarly it is impossible to describe data collection within a process view until concrete metrics are defined by the measurement view.

3.2.5 Definition of consistency properties

The consistency properties presented here extend ideas first mentioned by Lott and Rombach in [LR93]. They are intended to satisfy the requirements set out in Section 3.2.1. The consistency properties make statements about the integrated measurement and process views. These statements define a target condition in which personnel can use the integrated set of views to collect useful data, interpret that data appropriately, and judge the suitability of such views for providing measurement-based feedback to software developers. Further, the consistency properties supports checking that measurement is not based on false assumptions about the development activities, and that development activities can take advantage of quantitative data. An example of an inconsistency might be that some items from the measurement view do not dovetail with the process view (i.e., questions cannot be answered because they are meaningless in the given context). Another inconsistency might be that the measurement view references some development or maintenance activities that are not represented in the process view. The consistency properties are expressed as two models, one that reflects the purpose of understanding and one that reflects the purpose of providing feedback.

Consistency model U: Understanding

This model of consistency addresses views that were integrated for the purpose of understanding software development or maintenance activities. Information from the GQM question sets that concern the object of interest and the quality aspects of interest must be considered here. This model considers one-way data flow: data are gathered from development processes and plugged into the measurement view for interpretation. The resulting information will not be used to generate feedback in a SEE. The following discussion is limited to a single measurement view (i.e., a single GQM goal and the questions and metrics that directly support that goal).

Rule U1: Every goal's object of interest must be mapped onto some nonempty set of process-view objects. However, the set of process-view objects need not be covered (i.e., a goal doesn't have to mapped onto each process-view object).

Rule U2: The type of a goal's object of interest (process, product, or resource) must be the same as the type of object with which it is mapped. For example, it makes no sense to map a goal concerning a product onto process object.

Rule U3: Every GQM question must be mapped onto some nonempty set of objects from the process view. Similar to rule U1, the set of process-view objects need not be covered by a question mapping.

Rule U4: All metrics must be mapped onto at least one object or attribute to link data needs (metrics) with data sources (object models or attributes). This is defined using the following two conditions:

1. If a given metric is not mapped onto any process-view object, then that metric must be mapped onto some nonempty set of attributes.
2. If a given metric is not mapped onto some nonempty set of attributes, then that metric must be mapped onto some process-view object.

It makes sense to map a metric onto a set of attributes for metrics such as “the data bindings for each module changed” that must be collected from many products. This is the case of a repeated instantiation of a single process, product, or resource.

Rule U5: Metrics and attributes that are linked (mapped) with each other must be compatible in terms of type. The measurement view captures information about a metric’s unit, scale, and whether it is a subjective or objective measure. An attribute only includes information about its type (in the sense of the MVP-L modeling language). Type promotion among compatible types must be allowed. Table 3.1 gives rudimentary rules that express compatibility between metrics and attributes based on the metric’s scale and the attribute’s type.

Metric scale	Attribute Type
nominal	enumerated
ordinal	enumerated
interval	integer or enum
ratio	float, integer, or enum

Table 3.1: Compatibility of a metric and an attribute

Rule U6: Metrics can only be mapped to attributes that actually collect data. These attributes are probably (but not necessarily) attached to leaf processes, products, or resources. Although GQM metrics are restricted to those that are directly collectible, this condition cannot be imposed on all attributes because of the use of attribute mappings. An attribute mapping specifies how to compute the value of an attribute based on the value of other attributes. Therefore, metrics cannot be linked with attributes that derive their values via attribute mappings.

Rule U7: Leaf processes and products generally mirror actual processes and viewable products. In contrast, abstractions are often artifacts of the modeling process and do not represent real-world artifacts. Therefore, we recommend only mapping metrics onto attributes used in leaf elements.

Rule U8: All attributes of the process view must be used in some way. If there is no motivation (goal) or need behind an attribute, then there's no point in modeling it. An attribute may be motivated by the need for data in the measurement view or the need to express a constraint on control flow in the process view. Therefore, each attribute must be mapped to exactly one metric, or that attribute must be used in the criteria section of some process. This excludes attributes that are not capable of collecting data such as the "process status" attribute [BLRV95].

Rule U9: Because improvement through better understanding is the goal here, no use of attributes in entry/exit criteria that influence the control flow among processes is permitted.

Consistency model F: Feedback

The model of consistency for providing feedback is a superset of the model of consistency for the purpose of understanding. The extended model assumes that the views were integrated for the purpose of improving software development or maintenance activities by providing feedback. In addition to information from GQM question sets concerning the object of interest and the quality aspect of interest, the questions concerning feedback must also be considered here. In this scenario, data are gathered from the process and fed into the measurement view with the purpose of evaluating the data to provide feedback to the personnel – a two-way flow of data between the two views. The following discussion is restricted to a single measurement view (i.e., a single GQM plan).

Rules F1–F7: Rules U1–U7 are reused.

Rule F8: All attributes of the process view must be used in some way. This rule has the same motivation as the one in the model of consistency for understanding. The rule must be extended for the feedback scenario because of the possibility of mapping questions from the measurement view onto attributes from the process view (two-way data flow). If an attribute is not mapped onto a metric, nor is that attribute used in the criteria of some process, then that attribute must have been mapped with a question from the measurement view.

Rule F9: Because improvement through providing feedback is the goal, some use of attributes in process entry or exit criteria is expected. We require that at least one attribute be used in the criteria of some process. No rule is stated for the other direction. In other words, there is no requirement for mapping all attributes used in some process criteria onto a metric. The reason is that attributes may be used to express control flow based on simple status or availabilities as discussed previously.

Rule F10: Information from the measurement view must be fed back into the process view. At least one question must be mapped onto an attribute of some object in the process view.

3.3 Algorithm for Integrating Measurement and Process Views

The algorithm for integrating measurement and process views takes as input the goal of understanding or improving a set of business practices for developing or maintaining software. The output of the algorithm is a set of integrated views that support the automation of measurement-based feedback. The intended user of this algorithm is any person who sets goals and determines processes for a project, sometimes called a project planner.

The algorithm is presented in the context of the Quality Improvement Paradigm (QIP) [Bas85]. The Quality Improvement Paradigm specifies six steps towards attaining the goal of quality improvement in a software development or maintenance project. These steps are: (1) characterize the current environment, (2) define goals for the project and for process improvement, (3) model and choose the processes to track those goals, (4) execute the project and give feedback to personnel, (5) analyze the results, and (6) package the lessons learned for future use. The algorithm presented here focuses on activities that are performed within QIP steps 2 and 3. The resulting integrated view of software development activities can be used in QIP step 4 to support data collection and the generation of measurement-based feedback.

Sections 3.3.1 through 3.3.6 explain how integration tasks are accomplished within the six steps of QIP.

3.3.1 QIP step 1

The first step in QIP entails characterizing the environment (application domain, personnel, etc.), deciding what type of project is under study (a first-of-its-kind, the nineteenth refinement of a well-known system, etc.), and trying to find data in the organization about similar projects.

3.3.2 QIP step 2, integration task 1

The second step in QIP involves the definition of project goals and improvement goals. A project goal may be to offer a new "killer application" before any competitor. A supporting improvement (i.e., measurement) goal may be to shorten the cycle time involved in reviewing subsystem designs. The integration activity focuses only on the improvement goals and the need to evaluate them quantitatively, as supported by the measurement view.

Integration task 1

Integration task 1 involves developing the measurement goal, which is the first part of the measurement view.

Inputs: From QIP step 1, the best available knowledge of the type of project, the project's implicit goals, and any previously existing implicit processes.

Outputs: Statement of the measurement goal. Of primary importance is that this goal statement identifies the object(s) of interest to be analyzed, as well as the context. For example, a

measurement goal may target a design process or a requirements document within a specific application domain.

Algorithm: We use the GQM Paradigm to frame and structure measurement (and by implication, improvement) goals. To develop one or more such goals, the appropriate individuals will be interviewed, and relevant information from other sources will also be used. Ultimately, each goal will be refined in a tractable way into questions, and finally metrics, but not in this step. The definition of both metrics and data-collection procedures will be postponed until a comprehensive process view is available.

3.3.3 QIP step 3, integration tasks 2–6

The third step in QIP addresses choosing the processes to be used in order to attain the goals defined in the previous step. For example, a process may be optimized to bring a product to market extremely quickly, or to maximize the stability of a product to be delivered. Most of the integration work is accomplished within this QIP step, as discussed next.

Integration task 2

Integration task 2 involves developing the process view.

Inputs: The GQM goal statement from the previous step, which identifies the object(s) of interest as well as the context.

Outputs: A comprehensive process view, but one that still lacks quantitative criteria for the successful start and completion of processes.

Algorithm: The scope of the modeling effort is set by the GQM goal, specifically the statements concerning the object(s) of interest and the context. Models of the processes, products, and resources within this scope are created using the MVP–L process modeling notation.

It may be difficult to capture an unbiased picture of the processes after setting explicit measurement goals, because stating such goals may influence the modeling activities. Further, iteration with the previous and following integration tasks (stating the measurement goal and refining that goal, respectively) will almost certainly be required.

Integration task 3

Integration task 3 involves developing the second part of the measurement view, namely the questions and metrics that refine the GQM goal.

Inputs: The GQM goal and the comprehensive process view.

Outputs: The completed GQM plan.

Algorithm: In this task, the original measurement goal is refined using questions and metrics to yield a complete GQM plan. Iteration with the previous integration task of developing a process view will almost certainly be required. Note that there is still no measurement plan; i.e., no definition of the who, when, and how for collecting data.

Integration task 4

Integration task 4 involves integrating the measurement and process views.

Inputs: Measurement and process views from the previous steps.

Outputs: Integrated measurement and process views.

Algorithm: This task involves the steps that will integrate the measurement and process views according to certain predefined relationships. Parts of the measurement view (structured using the GQM paradigm) are linked with the appropriate parts of the process view (structured using the MVP-L formalism). Links are made as discussed in Section 3.2.3:

- Link the GQM goal's object(s) of interest with MVP-L object(s).
- Link data needs with data sources (at least metrics and attributes).
- Link the GQM questions with MVP-L objects.
- Link GQM metrics with MVP-L events.
- Link GQM metrics with MVP-L person resources.
- Link GQM questions with MVP-L attributes.

The activity of linking the two views will probably require both views to be extended. A difficult question is how much the process view can realistically be changed to accommodate the necessary measurement activities. The primary concern is that only data will be collected that is absolutely necessary and well motivated by the measurement view. Finally, multiple measurement views (that support multiple measurement goals) may be integrated with each other by repeatedly integrating a measurement view with an existing process view.

Integration task 5:

Integration task 5 involves checking the consistency of the measurement and process views.

Inputs: The integrated set of measurement and process views.

Outputs: A judgement about consistency problems in that set of views.

Algorithm: The consistency properties defined in Section 3.2.5 are checked in this task. Essentially, checking the consistency properties ensures that the integrated measurement and process views will help personnel collect useful data and interpret that data appropriately.

Integration task 6:

Integration task 6 involves developing the last part of the measurement view, namely the measurement plan.

Inputs: The integrated (and consistent) measurement and process views, especially the mapping of metrics with events and persons.

Outputs: The measurement plan.

Algorithm: The measurement plan specifies the who, when, and how of data collection. All information that must be included in the measurement plan is available in the integrated views. The linking of metrics with models, attributes, events, and personnel resources determines the who, when, and how of all data collection. Writing a comprehensive measurement plan is simply a matter of extracting the data from the integrated views. The most difficult part is developing the data-collection forms.

Data-collection forms are also considered to be part of the measurement plan. An additional design criteria for the forms is the need for gaining valid data [BW84]. The main benefit of developing the required forms as part of the integration activity is that one can identify the points in time (events) when various data points should be collected [Ham94a]. Identifying these events (and obviously the persons responsible for providing the data) groups the data to be collected at any one point in time. These groupings are a natural unit for the data that can be collected using a single data collection form.

3.3.4 QIP step 4

The fourth step in QIP involves performing the project as specified by the processes chosen in the previous step. The integrated views are finally used during this step to support data collection and to generate guidance and feedback for the software developers. The views can also be used to validate and interpret the collected data.

3.3.5 QIP step 5

The fifth step in QIP involves analyzing and interpreting the collected data. The integrated views will assist in the analysis because they document the planner's original expectations.

3.3.6 QIP step 6

Finally, Step 6 in QIP involves packaging the lessons learned from data analysis for use in the future. Under “packaging” the reader should understand adding a description of the background and context information necessary to understand and use the lessons learned. Packaging may involve improving the set of integrated measurement and process views. The process view might be streamlined to eliminate steps that were found unnecessary, or the measurement view might be augmented to collect additional data about validation steps.

3.4 Example of Developing and Integrating Views

This section demonstrates the application of the algorithm presented in Section 3.3. A measurement view and a process view are developed, integrated with each other, and the result is checked for consistency. The DCT1 example first presented in Section 1.4.1 is reused.

3.4.1 Task 1: Start the measurement view

For this example, assume the existence of a development organization that would like to gain an understanding of the effort spent on development activities. The following GQM goal concerning the quality aspect “effort” is used in the example:

Analyze the **processes of project DCT1**
in order to **understand them**
with respect to **effort**
from the viewpoint of the **development team leader**
in the (fictional) **development organization GBR**.

3.4.2 Task 2: Develop the process view

After the goal is stated and the objects of interest are thereby identified, the next task is to represent those objects explicitly by developing the process view. In this case, the objects of interest are processes. We assume that the DCT1 project follows a life cycle based on the idea of iterative enhancement [Bas75]. In that life cycle model, the requirements document is divided into chunks, and the project repeatedly iterates through design, coding, and testing processes until the entire set of requirements has been implemented.

Figure 3.2 presents the MVP-L “project_plan” that comprises the top level of a comprehensive process view. Figure 2.4 (on page 29) presented the MVP-L “process_model” that describes the design process. The models for the other processes are highly similar to the design process model. The criteria in the process models are limited to asserting facts about the availability and completeness of the input and output products. Figure 3.3 presents the MVP-L “product_model” that describes the requirements document. The models for the other products are highly similar to the requirements document model. Figure 3.4 shows the MVP-L “resource_model” that describes a developer. The models for the other resources are highly similar to the developer resource model. These additional models will help identify possibilities for data collection.

```

project_plan Testplan2 is

imports
  process_model Write_design, Write_code, Test_code;
  product_model Requirements_document, Design_document;
  product_model Code_document, Test_document, Feedback;
  resource_model Developer, Tester, Manager;

objects
  write_des   : Write_design;
  write_code  : Write_code;
  test_code   : Test_code;
  req_doc     : Requirements_document;
  des_doc     : Design_document;
  code_doc    : Code_document;
  test_doc    : Test_document;
  df : Questions;
  cf : Problems;
  tf : Failures;
  Doe : Developer("J. Doe");
  Roe : Developer("N. Roe");
  James : Tester("S. James");
  Bert : Manager("D. Bert");

object_relations
  write_des ( in1 => req_doc, out1 => des_doc, out2 => df);
  write_code( in1 => des_doc, out1 => code_doc, out2 => cf);
  test_code ( in1 => code_doc, out1 => test_doc, out2 => tf);

end project_plan Testplan2

```

Figure 3.2: MVP-L project plan for the DCT1 example

The process model for the “Write_Design” process indicates that both developers “Doe” and “Roe” are responsible. The responsibility for the “Write_Code” process is similarly assigned to the two developers, and responsibility for the “Test_Code” process is assigned to “James.” A manager is modeled as a resource in the process view, but is not assigned as a person responsible for the technical activities of the processes. The manager, “Bert,” plays an observational role.

Not shown in these models are events (points in time) that transpire during project execution. MVP-L specifies that the events “project.start” and “project.complete” are defined for all objects, the events “start” and “complete” are defined for all processes, and the events “consume.start”, “consume.complete”, “produce.start”, and “produce.complete” are defined for all products. The models as shown do not take advantage of these events, but the events will later be extended and used as part of the integration step.

Table 3.2 summarizes the information required from a process view. Next, the rules for internal consistency of a process view can be checked. Sets of process, product, and resource models have been developed, mappings have been declared for attributes, and responsibility for product and processes has been determined via mapping with personnel.

```

product_model Requirements_document(status_0 : Product_status) is

  product_interface
    imports
      product_attribute_model Product_status;
    exports
      status : Product_status := status_0;
    end product_interface

  product_body
    implementation
    end product_body

end product_model Requirements_document

```

Figure 3.3: MVP-L product model for “Requirements_document,” before integration

```

resource_model Developer(Name_0 : Person_name) is

  resource_interface
    imports
      resource_attribute_model Person_name;
    exports
      Name : Person_name := Name_0;
    end resource_interface

  resource_body
    implementation
    end resource_body

end resource_model Developer

```

Figure 3.4: MVP-L resource model for “Developer,” before integration

Aspect	Examples
Processes	“Write_design”, “Write_code”, “Test_code”
Products	“Requirements_document”, “Design_document”, etc.
Resources	“Developer”, “Tester”, “Manager”
Object refinements	none
Attributes	“Product_status”, “Name”, etc.
Link attributes and objects	“Product_status” with “Requirements_document”, etc.
State attribute types	“Product_status” is “enumerated”, etc.
State source of data	All data sources are “external”
Personnel	“Doe”, “Roe”, and “James”
Link personnel and processes	“Doe” and “Roe” with “Write_design”, etc.
State process events	“project_start”, “start”, “produce.complete”, etc.
State use of attributes	“Product_status” is used as “criteria”, etc.

Table 3.2: Summary of the example process view

Nr.	Question
1	What is the average experience of the developers?
2	How many parallel tasks compete for the developer's time?
3	What process steps are involved?
3.1	What steps are planned?
3.2	What steps are actually performed?
4	How severe is the time pressure (deadlines)?

Table 3.3: Questions to describe the objects of interest (set O)

Nr.	Question
5	How is time spent on the project?
5.1	What are the estimated time requirements for the project?
5.2	What is effort and calendar time per requirements chunk?
5.3	What is the distribution of effort/time across the process steps?
5.4	What is effort and calendar time per increment?
6	What is the effort for activities not described by the process model?

Table 3.4: Questions to describe the quality aspect of interest (set Q)

3.4.3 Task 3: Complete the GQM plan

Next the GQM plan is completed using the process view. Completion of the measurement view is postponed until the integration step. Because the goal will ultimately be used to build a baseline (none exists already), no baseline is presented as part of the measurement view.

Supporting questions and metrics. The questions that are used to refine the goal provide a rationale for the choice of metrics. However, the refinement of a GQM goal into questions and ultimately into metrics is a demanding, intuitive task. The integration framework offers no systematic help for this task.

Table 3.3 lists the questions that describe the objects of interest, which in this case are the processes. Table 3.4 lists the questions that describe the quality aspect of interest, which in this case is effort. The refinement of questions into other questions is also shown in the two tables. Because the intent of this goal is to gain understanding, no feedback is required, and therefore question set F (feedback) remains empty. Finally, Table 3.5 lists the metrics for which data must be collected to answer the questions. The table shows refinement relationships between the questions and metrics, assigns each metric a number, describes the metric, gives the units for the metric, states the scale, and finally describes the metric as either a subjective or objective measurement. The procedure for collecting data for all of these metrics will be to query the people involved.

3.4.4 Task 4: Integrate the views

The measurement view and the process view are integrated with each other in this task. This is done by specifying the integration information that links portions of both views, and by extending

Q. nr.	M. nr.	Metric	Units	Scale	Sub/obj.
1	1	Number of developers	count	abs.	obj.
1	2	Estimated experience	scale 1..5	ord.	sub.
2	3	Number of parallel tasks	average	ratio	sub.
3.1	4	Planned steps	model	nom.	obj.
3.2	5	Steps actually performed	prose	nom.	obj.
4	6	Time pressure	high, avg., low	ord.	sub.
5.1	7	Estimated effort for the project	person hours	ratio	obj.
5.1	8	Estimated effort for the project	calendar days	ratio	obj.
5.2	9	Number of requirements chunks	count	ratio	obj.
5.2-5.4	10	Effort per iteration in design	person hours	ratio	obj.
5.2-5.4	11	Effort per iteration in design	calendar days	ratio	obj.
5.2-5.4	12	Effort per iteration in code	person hours	ratio	obj.
5.2-5.4	13	Effort per iteration in code	calendar days	ratio	obj.
5.2-5.4	14	Effort per iteration in test	person hours	ratio	obj.
5.2-5.4	15	Effort per iteration in test	calendar days	ratio	obj.
6	16	Effort for non-project activities	percentage	ratio	sub.

Table 3.5: Metrics that help answer the questions

Set/Fn.	Content
Goal	"To analyze..."
Intent of goal	Intent value "understand"
Object of interest type	Type value "process"
Questions	"What is the average experience...", etc.
Metrics	"Number of developers", etc.
Metric units	State unit of "count" for metric 1, etc.
Metric scale	State scale "absolute" for metric 1, etc.
Metric sub./obj.	State "objective" for metric 1, etc.
Metric collection	State "person" for all metrics

Table 3.6: Summary of the example measurement view

some information in the process view as needed. The measurement plan is similarly developed using the integration information. First the direct overlaps are specified, and then the other links are explained.

Map goal and objects

The goal is linked with the three process objects mentioned in the process view. This establishes that the objects of interest are clearly identifiable in the process view.

Link data sources and data needs

Data demanded by a metric can sometimes be gathered simply by examining the model for some object in the process view, especially for metrics that refine questions from GQM set O (describe the objects of interest). However, data must be collected in most cases during project performance. Table 3.7 shows the correspondence of the metrics with both objects and the newly defined attributes; both mappings are discussed in more detail next. The type (in the sense of the MVP-L modeling language) is also shown for all the mappings; either an object model or an attribute type is given. In the cases where a model is used to satisfy the needs of a metric, the scale of the metric is understood to be "nominal." Both cases are discussed next.

Case I: Link metrics and objects. For this example, data demanded by metrics 1 and 4 can be gathered from the process view directly. Metric 1 ("number of developers") leads to the data value 3 based on the three instances of personnel resources declared in the process view. The needs of metric 4 ("planned steps") are well served with the portion of the process view that describes the processes (i.e., the process models).

Case II: Link metrics and attributes. Based on the mapping of questions to process-view objects, the process view's set of attributes is extended as shown in Table 3.7. These extensions permit mapping metrics onto the new attributes that model data sources explicitly. Note that the issue of gathering data from multiple instantiations for a single metric must be handled in the mapping from metrics onto attributes. The issue of multiple instantiations can take the form of a number of products, each of which is represented in the process view, or the form of a single process that is performed multiple times. In both cases, the data demanded by a single metric will comprise a set of values.

Map questions and objects

The questions are similarly linked with the objects, which establishes possible sources for data that will answer the questions. Table 3.8 summarizes this information.

Table 3.8 lists no mapping for question numbers 3.2 and 6. This indicates that no appropriate entity could be found in the process view from which it would be natural to collect data to answer that question. Questions 3.2 and 6 demand information about the actual tasks performed (as opposed to the planned tasks) and about the effort spent on activities not considered part of

Metric	Description	Process-view object	Attribute	MVP-L type
1	Nr. devel.	Project 'DCT1'	n/a	proj. plan
2	Est. exp.	Personnel 'Doe', etc.	Experience	enum.
3	Par. tasks	Ditto	Parallel_tasks	real
4	Planned steps	All processes	n/a	proc. mod.
5	Actual steps	-no mapping-		
6	Time pressure	All processes	Pressure	enum.
7	Est. eff. hrs.	Project 'DCT1'	Est_hours	real
8	Est. eff. days	Ditto	Est_days	real
9	Req. chunks	Product 'req_doc'	Chunks	integer
10	Des. eff. hrs.	Process 'write_des'	Effort_hours	real
11	Des. eff. days	Process 'write_des'	Effort_cdays	real
12	Code eff. hrs.	Process 'write_code'	Effort_hours	real
13	Code eff. days	Process 'write_code'	Effort_cdays	real
14	Test eff. hrs.	Process 'write_test'	Effort_hours	real
15	Test eff. days	Process 'write_test'	Effort_cdays	real
16	Other effort	-no mapping-		

Table 3.7: Mapping of metrics and objects/attributes for example DCT1

Question	Objects
Q1, average experience	Personnel (Doe, Roe, and James)
Q2, parallel tasks	Personnel and processes "write_des", "write_code", "write_test"
Q3.1, planned tasks	Processes "write_des", "write_code", "write_test"
Q3.2, actual tasks	-no mapping-
Q4, time pressure	Processes "write_des", "write_code", "write_test"
Q5.1, estimated time	Processes "write_des", "write_code", "write_test"
Q5.2, effort / requirement	Product "req_doc", processes "write_des", "write_code", "write_test"
Q5.3, effort distribution	Processes "write_des", "write_code", "write_test"
Q5.4, effort / increment	Processes "write_des", "write_code", "write_test"
Q6, other effort	-no mapping-

Table 3.8: Mapping of questions and objects for example DCT1

the project, respectively. Because this information is beyond the scope of the process view, it lies outside the applicability of the integration model, as discussed in Section 3.2.4. Thus these questions will be ignored when analyzing the integration for consistency. The default resolution of this problem is to query the personnel for these data.

Link metrics and events

Events model the points in time when data should be collected. Experience has shown that the points in time when processes begin and end are well suited for collecting data [BDT96]. A complete set of "start" and "complete" events are defined for all processes in the process view by default, so the set of events does not need to be extended. Because the processes of project DCT1 are performed iteratively, the estimated effort for each iteration of each process must be collected when the process begins, and the actual effort must be collected from each process when it completes. This is an example of collecting data for a single metric from multiple instantiations

Metric	Description	Event	Person(s) responsible
1	Nr. devel.	n/a	n/a
2	Est. exp.	project_start	Doe, Roe, James
3	Par. tasks	project_complete	Doe, Roe, James
4	Planned steps	n/a	n/a
5	Actual steps	project_complete	Doe, Roe, James
6	Time pressure	process_complete	Doe, Roe, James
7	Est. eff. hrs.	project_start	Bert
8	Est. eff. days	project_start	Bert
9	Req. chunks	project_start	Roe
10	Des. eff. hrs.	process_complete	Roe
11	Des. eff. days	process_complete	Roe
12	Code eff. hrs.	process_complete	Roe
13	Code eff. days	process_complete	Roe
14	Test eff. hrs.	process_complete	James
15	Test eff. days	process_complete	James
16	Other effort	project_complete	Doe, Roe, James

Table 3.9: Mapping of metrics, events, and personnel for example DCT1

of some object, yielding a set of values. The mappings for other metrics are also straightforward. The mapping of metrics onto events is summarized in the first three columns of Table 3.9.

Link metrics and personnel

In order to capture the responsibility for gathering data for a metric, metrics must be linked with personnel resources. The MVP-L models show that three developers are involved in the project. Developers “Doe” and “Roe” cooperate on the design and code processes, while developer “James” is solely responsible for the testing process. As mentioned earlier, manager “Bert” plays only an observational role, but can be queried for data. Developers “Doe” and “Roe” are therefore equally good choices for querying about data pertaining to the products and processes of designing and coding. Only one is needed, and “Roe” is chosen (ideally based on an agreement between the two members). Developer “James” will be queried for data pertaining to the products and processes of testing. All three developers will be queried for data that concerns each of them individually such as experience. Using this information, the mapping of metrics onto personnel is summarized in the first, second, and last columns of Table 3.9.

Link questions and attributes

Questions are linked with sets of attributes in order to use answers to generate feedback. However, the goal in this example concerned understanding (not feedback), so this function is not needed, and is therefore left undefined.

The foregoing discussion defined all information required to integrate the measurement and process views with each other. Table 3.10 summarizes that information.

Mapping	Example
Link goal and objects	Map goal onto processes "write_des", etc.
Link questions and objects	Map question 1 onto resource instances "d1", "d2", etc.
Link data needs and sources	Map metric 1 onto the project_plan, etc.
Link metrics and attributes	Map metric 2 onto attribute "Experience", etc.
Link metrics and events	Map metric 2 with event "project start", etc.
Link metrics and personnel	Map metric 1 with all three personnel, etc.
Link questions and attributes	(not applicable here)

Table 3.10: Summary of the example integration information

3.4.5 Task 5: Check consistency of views

The example is based on the goal of gaining an *understanding* of effort in project DCT1. Therefore, the rules from consistency model 'U' defined in Section 3.2.5 are used to analyze consistency properties of the integrated views.

Rule U1 is satisfied because the GQM goal's object of interest is mapped onto a nonempty set of process-view objects. Rule U2 is satisfied because the GQM object of interest is a process and is only linked with processes from the process view. Strictly speaking, rule U3 is not satisfied because it does not link every question with some set of objects from the process view. However, as was stated earlier, two of the questions in the example measurement view reference aspects that are outside the scope of the process view, and the integration model does not apply to such questions. Therefore, rule U3 is satisfied for all questions within the applicability of the integration model. Rule U4 is satisfied because all metrics are used, either via a mapping with a metric or as part of a process constraint. Rule U5 is satisfied because the metrics and attributes that are linked with other are compatible in their types. Rule U6 is satisfied because all metrics are linked with attributes that are used to collect data directly from people; none of the attributes mapped with metrics are used in attribute mappings. Rule U7 is satisfied simply because there are no refinements of processes, products, or resources in the simple example; for that reason, all attributes appear in leaf elements. Rule U8 is satisfied because all attributes are used in some way, either to denote data collection for a metric or to constrain the control flow among the processes. Finally, rule U9 is satisfied because the empirical data collected via attributes is never used in a criteria of a process.

3.4.6 Task 6: Develop the measurement plan

Finally, the integrated and consistent views are used to develop the measurement plan, including data-collection procedures. The procedures are based on the events identified when developing the integration information, in conjunction with the responsibilities for collecting data. Earlier it was stated that all data are collected by querying personnel. The integration information helps design the data collection forms. Using the events, all data that must be provided at a certain time by a single person are grouped together onto a single data-collection form. In addition to supporting the development of data collection forms, this approach also helps minimize the effort and intrusiveness of the data collection procedures [BDT96]. The completion of the measurement plan thus completes the measurement view, as follows.

- Event “project_start”
 - Doe, Roe, and James: estimated experience.
 - Bert: estimated hours and days for the project.
 - Roe: count of requirements chunks
- Event “complete” of process “Write_design”
 - Roe: effort in hours and days.
- Event “complete” of process “Write_code”
 - Roe: effort in hours and days.
- Event “complete” of process “Test_code”
 - James: effort in hours and days.
- Event “project_complete”
 - Doe, Roe, James: parallel tasks, actual steps, other effort

3.4.7 Summary of the example

This example demonstrated that an enormous amount of detailed information must be developed to build the foundation for providing measurement-based feedback. That information can be developed systematically by applying the algorithm from Section 3.3 and by analyzing the result using the consistency properties defined in Section 3.2.5. Next we present a study to evaluate the integration approach.

3.5 Evaluation of the Integration Approach

This section presents a preliminary evaluation of the integration approach based on a cooperative effort between academia and industry. The goal of this evaluation is to extract some lessons learned from applying the model of consistency, and to identify both strengths and weaknesses in the integration approach.

3.5.1 Planning an industrial measurement program

The cooperative effort involved researchers and industrial personnel who implemented a measurement program in a software development department of the Robert Bosch GmbH [Hoi94, BD94, Ham94a, Ham94b, Fe195, BDT96]. In the course of that effort, integrated measurement and process views were developed based on an early version of the integration approach. No consistency properties had yet been defined for the views.

The approach taken by the researchers who worked with Bosch is sketched in [Ham94b]. They began by modeling the products, processes, and resources (in that order) to develop a process view. Then they developed GQM plans using the technique of GQM abstraction sheets coupled

with interviews. Their next step was to integrate the process view and the partial measurement view primarily by linking metrics and attributes. Finally, they used their integrated views to choose points in time to measure and to develop automated support for data collection, storage, and evaluation.

Their work confirmed the value of using process models in conjunction with planning a measurement program, a fundamental assumption of the integration approach [BDT96]. Further, the combined use of measurement and process views made the planning of data-collection procedures straightforward. Clustering items for individual data-collection sheets was easily done based on events (timing) as well as processes and products.

3.5.2 The case study

The case study examined the measurement and process views that were developed for Bosch.

Hypothesis

The hypothesis investigated in the case study is that inconsistencies remain in the integrated set of views, and that these inconsistencies will be detected by the model of consistency presented in this research.

Procedure

The measurement and process views developed for Bosch were examined first for internal consistency properties specific to each view, and then for the consistency properties specified for the integrated result. Interviewing the researchers who developed those views also helped gain an understanding of their work processes and of the results.

Results

The GQM plans developed for Bosch had significant traceability problems. A traceable refinement from goals to questions and finally to metrics is extremely important to accomplish the integration with a process view. Traceability means that all aspects mentioned in the goals and especially questions are ultimately treated by the metrics. Further, no metrics are defined for aspects not mentioned in the questions. The case study exposed numerous traceability problems in the Bosch measurement views that could have been avoided if the internal consistency rules for a measurement view had been checked. This suggests that a systematic approach would have been superior to the *ad hoc* approach that was actually taken.

Lessons learned

Multiple instantiations are common. The Bosch measurement views commonly included GQM metrics that refer to multiple instances of some artifact; for example, "all the functions of module Y." The repeated use of metrics that refer to multiple instantiations of real-world artifacts

implies that it is important to support mapping a metric onto a set of attributes. Such a mapping supports the goal of collecting a set of values, not just a single value.

Both models and instances can answer questions. The measurement view demands answers for a set of questions that are intended to define the object of interest. The models in the process view serve as one means for answering all types of questions from the measurement view (specifically, the questions to describe the objects of interest). The instances (e.g., processes, products, and resources in the project) are the other source of data for answering these questions, as expressed by the attributes declared for those instances. Although the instances could answer all possible GQM questions, it is inexpensive and helpful to derive data from the models in the process view. Conversely, the process view alone cannot answer all questions. It's impossible to model sufficiently well to test any hypothesis (i.e., a GQM goal) that might come along later.

3.5.3 Summary of the evaluation

The results indicate that the integration approach proposed in this work is extremely helpful for planning a measurement program, but that the consistency properties set a standard that is difficult to achieve. Therefore, we can support the hypothesis that checking the rules for internal consistency of measurement and process views reveals inconsistencies. As mentioned previously the measurement view suffered from problems of traceability. It appears that process and measurement planners need more explicit instructions for modeling and planning. However, we can make no statements about the hypothesis concerning inconsistencies in the integration. It was unrealistic to check the consistency properties based on the integration of internally inconsistent measurement and process views.

The case study served as an initial evaluation on the integration approach proposed in this work. It showed that the approach leads to concrete benefits when planning a measurement program, but that achieving the consistency properties necessary for integrated measurement and process views is a significant challenge.

The next section presents a set of requirements for a system that will provide automated support for measurement-based guidance and feedback.

3.6 Requirements for Automated Support

This section states requirements for an on-line system that will provide guidance and feedback to software developers about their work.³ Figure 3.5 illustrates the scope of activities which a process-centered SEE might support. Models of processes, products, resources, and measurement activities are constructed using various editors and combined, thereby forming an integrated set of views. Empirical quality models for the environment's products, processes, and resources are included in the integrated views to define process constraints and to set goals. The integrated set of views may be checked for various properties using an analyzer, and is finally enacted by a process engine to provide measurement-based feedback to teams of developers. During enactment

³Portions of Section 3.6 are copyright 1994 IEEE and are reprinted with permission from *Proceedings of the Ninth International Software Process Workshop*, Airlie, Virginia, 5-7 October 1994, pp. 47-49.

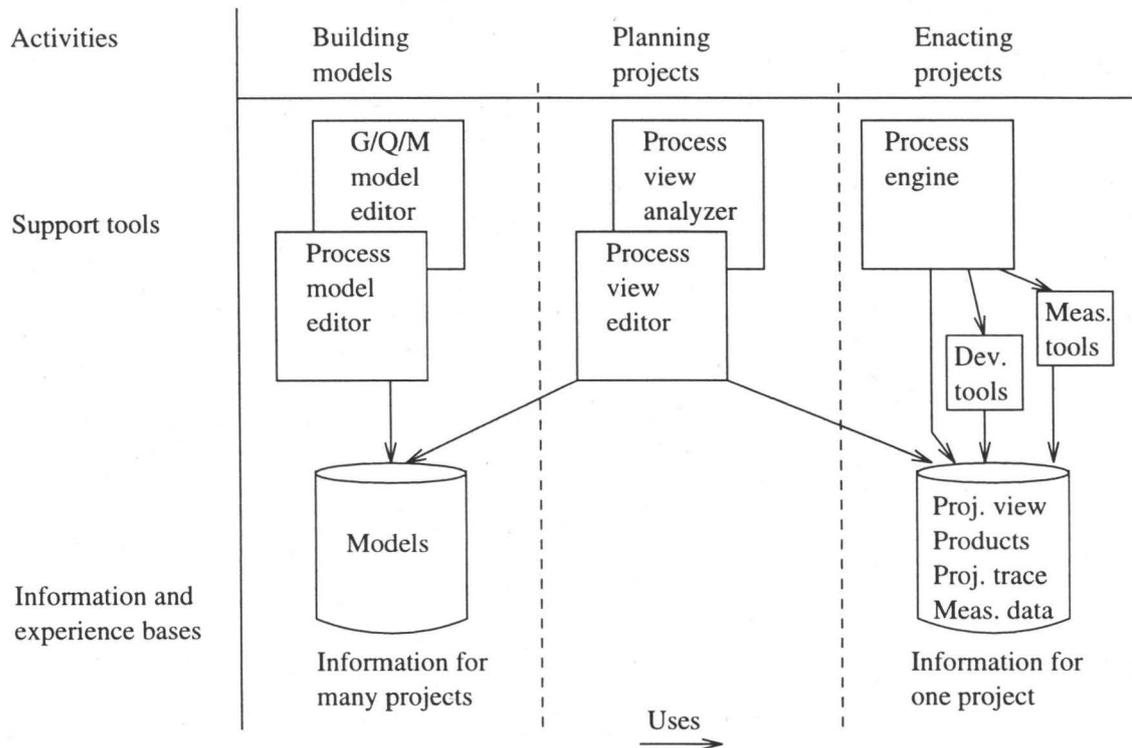


Figure 3.5: Scope of activities for a process-centered SEE

of the process view, various development and measurement tools are used to accomplish the work and to collect data.

Next, a number of requirements are stated that a process-centered SEE must satisfy for the purposes of this research. Sets of requirements are presented for interacting with users to collect empirical data, for supporting technical roles, and for supporting management roles. The fundamental concept to be supported is the systematic definition, collection, use, and evaluation of integrated process and measurement information to provide measurement-based feedback.

3.6.1 User interactions for collecting data

This section addresses the problem of gathering empirical data from ongoing software development activities using a SEE, sometimes called placing “hooks into the real world” [Huf93]. But unlike programmatic hooks, by which some software packages may be customized and which run automatically, data collection hooks often depend on human interaction. Examples of data that must be supplied by humans are the effort spent tracing a system failure to a software fault and the classification of the fault’s type and probable cause.

The difficulty of using a SEE to collect data by interacting with people encouraged the refinement of the high-level requirements that Rombach presented in [Rom89] to focus solely on data collection. Requirements are stated that a process-centered software engineering environment must satisfy in order to collect empirical data from personnel. The requirements, which were first sketched in [GHLR94], are divided into basic technical issues, technical issues due to human involvement, and nontechnical issues. Since this is still an open question that requires further research, meeting these requirements is not claimed to be sufficient for solving all problems in

data collection.

Basic Technical Issues. The process of data collection is composed of the following steps:

- Step 1: Decide upon a data item (metric) and define it unambiguously.
- Step 2: Specify the condition that should trigger data collection when it occurs.
- Step 3: Recognize when the trigger has occurred.
- Step 4: Collect a value for the data item.

Deciding what data to collect and defining the metric unambiguously is a highly creative activity. The task of step 2, specifying a triggering condition for the computer, ranges from trivial to impossible; the recognition of the trigger's occurrence (step 3) is equally easy or difficult. Only people understand trigger specifications such as "detailed design is complete" and can recognize their occurrence. However, it is simple to specify such triggers as saving a file from an editor or compiling a source-code file, and it is equally simple for a machine to recognize their occurrences. For example, in the Provence system [KB93], a recognizable trigger consists of moving a document into a designated directory. In the case of on-line work, where the work products are available to on-line data collection tools, data collection (step 4) from those products can be automated. This discussion of data collection motivates the first three requirements:

- R1: Accept machine-understandable specification of triggers.
- R2: Recognize occurrences of triggers.
- R3: Invoke tools to collect data.

An SEE must be coupled with on-line work to justify the preceding discussion of invoking tools to collect data. Fernström introduces four coupling levels in [Fer93], namely loosely coupled (a euphemism for uncoupled; the SEE only knows what people tell it), active support (access to work products is partially automated), process enforcement, (access to work products is totally controlled), and process automation (no human intervention is required). Requirements R1–R3 motivate the next requirement:

- R4: The SEE is coupled ("hooked") to on-line work products.

Next a few examples of automated data collection from on-line work products are given. One involves attaching data collection to a version control system, where the action of checking a document into the system is defined as the trigger. Another example is presented by the tools for conducting inspections of documents on-line, which could be used to gather data about the complex inspection process at low cost [MDTR93]. Finally, computer-aided software engineering (CASE) tools for constructing software work products offer a great (yet mostly neglected) possibility for collecting data about those products. For example, the CASE tool "JoYCASE" automatically collects values for the function point and function bang metrics from structured analysis diagrams drawn using that system [RLL93].

Data collection activities can be expected to yield an enormous store of data for which persistence and querying capabilities will be required. Storage, retrieval, and management of empirical data should be delegated to a database management system.⁴ Query support systems

⁴This issue should not be confused with the problem of storing all types of software-engineering work products in a database.

can then be used to test hypotheses about the data, a function arguably outside the domain of a SEE. This issue motivates the next two requirements:

R5: Store empirical data in a database.

R6: Retrieve empirical data from a database as needed.

An SEE may obtain data from data-collection tools via two communication paths, either via a direct coupling with a tool or via a database. In the first, the SEE invokes a tool and accepts the new data directly from that tool. In the second, the SEE invokes a data-collection tool that stores the new data in the database. The SEE subsequently uses a unique key (probably obtained from the tool) to query the database for the newly collected data, or may fetch previously collected data from the database without invoking a collection tool. This discussion leads to the next requirement:

R7: Accept data from tools via multiple paths.

Technical Issues Caused by Human Involvement. Data cannot be demanded from a person in the way data can be collected from an on-line product. The person may not be available at the precise moment when the SEE needs the data, or may need a long time to collect and submit the data. Therefore, the SEE must not block or otherwise delay all process-tracking and feedback activities while waiting for a reply from a person. One solution is that the SEE sends an asynchronous request to the person about the data that it needs. The SEE does not block on the data supplier until the data is provided, but instead accepts the data at any time after sending the request. However, data will arrive at unpredictable times, so the SEE must deal with incomplete data on a regular basis. This discussion motivates the following requirements:

R8: Notify people of the need for data from them.

R9: Accept data from people asynchronously.

R10, Function with incomplete data.

In order to collect data about off-line work such as think time, brainstorming sessions, meetings, and inspections, the people involved must be queried. On-line forms offer a reasonable method for querying people without requiring a human interviewer [Rom89]. Figure 3.6 gives an example of an on-line forms tool that was derived from [Nat90], a possible implementation of this requirement:

R11: Query people using on-line forms.

Nontechnical Issues. These issues are primarily concerned with gaining the trust and acceptance of the humans who supply data. Technical personnel must be well informed about the improvement goals that led to measurement, as well as the intended use of the data, so that they are willing and able to provide valid data. Providing this information on-line will help people understand the goals and allow them to refresh their memories as needed. These issues, also discussed in [BW84], motivate the following requirements:

R12: Explain data definitions on-line.

R13: Explain goals and intended use of the collected data on-line.

```

prf
Press control-g to enter menu. Press control-h for help.
File Quit

PERSONNEL RESOURCES FORM
  Name: C. M. Lott
  Project: MVP-S
  Date: 26 Feb 94

Section A: Total Hours spent on Project for the Week: 14

Section B: Hours by Activity
  Predesign
  Create design      10
  Read/Review Design
  Write Code         4
  Read/Review Code
  Test Code Units
  Debugging
  Integration Test
  Acceptance Test
  Other
  Go on to Section C y

Create Design: Development of the system, subsystem, or components design.
Includes development of PDL, design diagrams, etc.

```

Figure 3.6: Forms-based data collection tool

Next there is the issue of data which is linked to individuals. First, storage of such data is restricted by law in many countries, Germany among them. Second, if management is going to see the numbers and their correspondence with individuals, there is the possibility that future performance reviews will be based on the data, for example on a developer's speed in isolating faults or effectiveness in testing code. In that case, people will do everything in their power to make themselves look good; this is absolutely understandable but it ruins data validity. The ultimate solution is to mask out identities when the data is stored. These issues are discussed at length elsewhere (see [Pff93]) and motivate this requirement:

R14: Allow the identities of all data submitters to be masked.

The third and last nontechnical issue is the extra work, intrusion, and annoyance associated with providing data. To address this issue, both management and technical people must be educated about the costs associated with collecting and validating data, as well as the benefits that stem from analyzing the data (for cost-benefit analyses, see [MP90, Dio93]). Second, all attempts must be made to minimize the effort of providing data and the intrusiveness on people's work. The computer can reduce the effort of collecting data by automating some collection activities and decreasing the effort required for other activities. The following requirement is also addressed in [KB93]:

R15: Minimize intrusiveness and overhead.

3.6.2 Generating feedback for technical roles

The following set of requirements addresses the interactions between users who play technical (i.e., non-managerial) roles and a system that provides measurement-based feedback. Interactions dedicated to collecting empirical data are excluded. These requirements primarily concern the minimum capabilities for generating the feedback that is presented to users about their work.

A system that will generate feedback for personnel must have a comprehensive knowledge of processes, products and resources for a given project. Therefore, the system must accept a formal (machine-interpretable) description of activities and documents (a process view, as previously discussed).

R16: Accept a formal definition of processes, products, and resources.

Based on the comprehensive information about a project, a user must be informed about the processes for which he or she is responsible, and also about the products and resources involved in those processes. This information may be presented textually or graphically. For example, a person responsible for a testing process should be informed about the recommended technique for developing test cases, as well as the criteria for successful completion of the testing process. Furthermore, the current state of all processes must be presented explicitly. That information might be displayed in conjunction with the basic information about the processes, products, and resources. For example, presenting the state of the process could mean noting whether the process is blocked (waiting for required inputs), or whether the process can be performed at any time. If the basic information about processes is shown graphically, then the state of these objects might be shown advantageously with different colors. Finally, the user must be able to query a support system for additional information about the current state at any time, such as evaluating measurement goals on demand. These needs motivate the following requirements:

R17: Present information about processes, products, and resources.

R18: Present the current state of all objects explicitly.

R19: Allow querying the state of all objects.

Users must be able to communicate changes in the status of various objects (processes, etc.) to the support system. These changes include starting or completing processes, arrival of new versions of documents, etc. For example, a computer system cannot automatically know the outcome of a review meeting, so it must accept notifications about changes in the status of objects from users. Obviously the system can only trust what its users tell it. This motivates the next requirement:

R20: Accept notifications about changes in the state of objects.

The generation of feedback concerning the success of a process depends wholly on a model of the processes in the support system. The model states expectations on processes, and these expectations are communicated to users. All baselines and goals defined in the model must be continuously evaluated to detect deviations from that model. For cases in which the system cannot automatically track the state of the processes and products, it must trust personnel to meet those expectations. However, if those expectations cannot be met, then replanning is generally needed. This discussion motivates the next requirements:

R21: Maintain the consistency of the state of all objects with actual project performance.

R22: Track the state of all objects to detect deviations from goals.

As mentioned above, replanning may be necessary in case of deviation from expectations. The term "replanning" is understood to involve changes to a process view that must be done by

people. Following such a replanning process, the altered process view must be used to restart the feedback system as close as possible to the situation that was current when replanning activities were initiated. Certain types of changes in the process view, such as deleting subprocesses, will require substantial work to fit the new process view together with the old situation and still allow feedback to be provided. This motivates the next requirement:

R23: Support recovery from replanning during project performance.

3.6.3 Capabilities to support managerial roles

The following set of requirements addresses the interactions between users who play managerial roles and a system that provides measurement-based feedback.

The issue of personnel resource assignments is addressed first. A process view must list the resources available for performing the various processes. For maximum flexibility, it should be possible to assign resources to individual processes during the project, not only during the planning phase. We assume that these assignments will be made by users who play managerial roles. An assignment may be completely fixed in that it states that only one particular individual may perform some process. Alternately, an assignment may be constrained but not completely fixed. One possible constraint would be to group resources into teams, and then make assignments based on teams. Assignment via teams must distinguish between cases in which one particular individual from a team, any individual from a team, or the entire team is assigned to a process. Another possible resource assignment constraint for a given process might involve a statement about qualifications. For example, a person could only be assigned if he or she had experience with some particular test technique, or had more than five years experience with the implementation language. Any personnel who meet the qualifications could be assigned (or could assign themselves) to that process. Regardless of how the assignments are made, they must be communicated to the personnel. Finally, to cope with vacations, illness, and other absences, any substitutions in resources must be communicated to the guidance and feedback system. These issues in resource assignment motivate the following requirements:

R24: Support the assignment (binding) of personnel resources to processes during the project.

R25: Support the grouping of personnel into teams, and use teams to assign personnel resources.

R26: Use specifications of minimal qualifications for personnel that can be assigned to a given process.

R27: Inform personnel of any assignments to processes made on their behalf.

R28: Permit substitutions to be made in personnel assignments.

Processes views may be structured hierarchically to give managerial personnel an overview and some control over the execution of those processes. The idea is that child processes refine some parent process into increasingly fine-grained tasks. Under normal project execution, it is expected that a parent process must be explicitly started before any child process may be started. Similarly, all child processes must explicitly completed before the parent process can be completed. This scheme allows a person in a managerial role to exercise some control over when processes may be performed. Deviations from this scheme are considered exceptions, and should be handled to provide personnel with maximum flexibility. In the case of a child being

started before its parent, the parent should be notified or possibly forced into an “active” state. In the case of a parent being completed before some child, the child should be notified or possibly forced to stop. The use of hierarchies motivates the following requirements:

R29: Support the execution of hierarchically structured process views.

R30: Handle exceptions in which a child must be started or parent must be completed contrary to normal conditions.

3.7 Summary

This chapter introduced the integration of measurement and process views based on the GQM Paradigm and the MVP-L process modeling notation, defined a set of consistency properties for an integrated set of views, presented an algorithm for accomplishing the integration, and offered a detailed example of building an integrated set of views. Requirements concerning automated support for measurement-based feedback were also stated.

These contributions were made possible by building upon existing research, especially the work on the Goal Question Metric paradigm. The author contributed to the design and development of the MVP-L process modeling language, developed the integration model and consistency properties for the integrated views, refined the integration model in cooperation with a working group at the University of Kaiserslautern, and defined the requirements for measurement-based feedback.

The next chapter discusses a process-centered software engineering environment that uses integrated measurement and process views, and satisfies the requirements stated in this chapter.

Chapter 4

Automated Support for Measurement-Based Feedback

This chapter discusses automated support for providing measurement-based feedback to personnel who participate in a software development or maintenance project.¹ Section 4.1 presents an overview. Section 4.2 describes MVP-S, a prototype system for providing measurement-based feedback, and gives an example of using that system. Finally, Section 4.3 states a series of assumptions about how users will interact with an automated system that offers measurement-based feedback.

4.1 Overview

We implemented a software system that supports our approach for providing software developers with measurement-based feedback using integrated measurement and process views [LHR95]. The system is named the “Multi-View Process System” (MVP-S). MVP-S is an existence proof of a process-centered software engineering environment that supports measurement-based feedback. Fundamentally, the system only provides information. MVP-S does not proscribe any actions on the computer, it does not provide any construction tools, nor does it store any of the documents that personnel may develop. The system informs users about the processes that they are expected to perform, displays information about the products and resources associated with each process, collects data about various aspects, and performs limited evaluation on those data. The MVP-S system architecture uses the client-server model, and supports distributed work across a local-area network. A process engine acts as the server for all information, based on the information in the integrated measurement and process views. The only client currently implemented is a user interface that lets technical personnel view and make changes to the information maintained by the server.

MVP-S has much in common with the systems Ginger and Amadeus (see Section 2.4.1). However, the three systems cover different portions of the solution space involved in providing measurement-based feedback. This solution space is defined to be the integrated, on-line use of

¹Portions of this chapter are copyright 1995 Springer-Verlag and are reprinted with permission from the *Proceedings of the Fourth European Workshop on Software Process Technology*, C. M. Lott, B. Hoisl and H. Dieter Rombach, “The use of roles and measurement to enact project plans in MVP-S,” pp. 30–48.

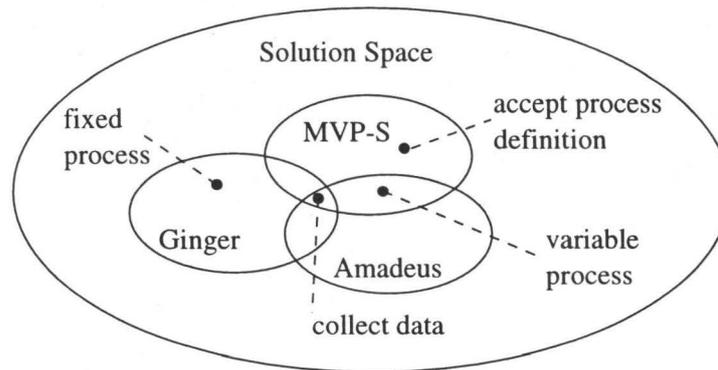


Figure 4.1: Coverage of the solution space for measurement-based feedback

measurement and process views (i.e., automation level 4 as discussed in Section 1.5). Figure 4.1 helps illustrate some specific differences. The Ginger system has extensive facilities for collecting and evaluating data automatically, can provide feedback to users based on that data, but is limited to a single, fixed process. In contrast, MVP-S has limited facilities for collecting and evaluating data, but treats process as a variable. The Amadeus system has comprehensive support for collecting data from code and other on-line documents, as well as for storing and displaying the data. However, Amadeus cannot provide feedback to personnel who perform some processes; it was designed to be used in conjunction with some process-centered environment. In contrast, MVP-S accepts a process definition and can provide feedback to personnel, but it offers no code analysis tools and has limited capabilities for storing and displaying data. In summary, the three systems solve different portions of a large problem.

A number of assumptions are stated concerning the interactions between users and a process-centered software engineering environment. For example, this work assumes that software developers and maintainers will *accept* measurement-based feedback. The operational definition of acceptance used here is that the developers will view the feedback as helpful, believe the feedback that they are given, and modify their behavior accordingly. We also make a number of assumptions about providing timely feedback and the usability of various user interfaces. These assumptions are stated to permit testing them in an empirical study that is described in the next chapter.

4.2 MVP-S, A Prototype Process-Centered SEE

We have implemented MVP-S, a process-centered software engineering environment (SEE), that supports measurement-based feedback. Section 4.2.1 presents existing functionality of the system, and Section 4.2.2 discusses possible extensions. Section 4.2.3 presents a detailed example of providing automated feedback to personnel using integrated measurement and process views.

4.2.1 Capabilities of the prototype

The MVP-S system provides role-specific feedback to software developers during their projects based on explicit process views, role definitions, quality models, and collected measurement data. This section focuses on the process engine and user interface.

Classification according to levels of automation

MVP-S is primarily a tool for collecting data from personnel and giving the personnel guidance and feedback. The MVP-S system is not a perfect match with any of the levels defined in Section 1.5. Automated support for the process view is clearly provided (matches levels 3a and 4), aspects of the measurement view are automated (matches level 4), but access to products and tools is not mediated by the system (matches level 2).

Basis for automated support

All automated support for providing measurement-based feedback in MVP-S is based on an extended process view. As explained in Chapter 3, the primary output of the integration of measurement and process views is an extended process view encoded in MVP-L. Choices of data to collect and ways to use the data were thus expressed in that process view using capabilities of MVP-L.

The process engine takes an extended MVP-L process view as input. A process view consists of a “project_plan” construct and a collection of supporting models. After reading the MVP-L process view, the process engine creates an internal representation that is the basis for all later queries and changes.

The use of role information

Role definitions are used to assign personnel to activities. We can identify at least four role groups. Each role group requires its own view of a software project. Personnel who play the *planning* roles, of which an example might be the project planner, construct the project plan before the project is performed. Personnel in the *technical* roles, such as the software developer,² are guided by the process view to accomplish the necessary work. The *management* roles, for example a project’s technical lead, require their own views to monitor information during enactment of the process view. Finally, those who play *replanning* roles, for example the project manager, require a view that will let them adjust the process view for unforeseen circumstances.

The prototype offers limited support for all activities involving these roles, but concentrates on giving guidance, collecting data, and offering feedback for the personnel who play the technical roles. Our prototype supports primarily the technical roles owing to our interest in collecting data from development activities. For example, a person playing the role of a design engineer shares responsibility for all processes that represent design activities. This person must be able to ask for information about the objects that are related to the design processes. These objects include the process model itself, the requirements document (consumed by a design process), the design document (produced by a design process), and the resources provided to enact the processes.

²Includes designers, programmers, testers, quality assurance engineers, maintainers, etc.

MVP-S Developer Interface		
action info plan help quit		
USER NAME: lott	PROJECT PLAN: ISPW_with_measurement	ROLE: Design_Engineer
Process ID	Process model	Process status
step_major_design_review	Major_Design_Review	disabled
step_minor_design_review	Minor_Design_Review	disabled
step_modify_code	Modify_Code	enabled
step_modify_design	Modify_Design	enabled
step_test_unit	Test_Unit	disabled

Figure 4.2: Viewing the project state in a role-specific work context window

User interface for technical roles

Personnel who play technical roles see a view of their processes named the role-specific work context.³ A role-specific work context is the interface between the developer and the set of processes that involve the developer in a single, technical role. An example of a MVP-S role-specific work context window appears in Figure 4.2, which shows the processes for user “lott” who works in project “ISPW with measurement” and plays the role of a “Design Engineer.” Column one in the figure shows the names of the process instances, column two the names of the corresponding process models, and column three shows the current status of each process instance.

A work context window shows only the processes for one developer in one particular role, so MVP-S supports multiple work context windows to let developers view information about all of their processes in all of their roles. The user interface also supports a detailed view of each process called a “process context.” An example of a process context view is shown in Figure 4.3. The status values of processes that appear in the role-specific work context window are kept consistent with the project state managed by the process engine. The status value for a process instance is one of disabled (can’t be performed), enabled (can be performed but no one is doing so), or active (is being performed). Upon receiving a request to start or complete a process, the process engine checks the entry and exit criteria specified for that process. The system informs the person who placed the request accordingly whether the request could be approved according to the current state of the project. This constitutes a simple model for coordinating teams of personnel by interacting with the SEE. Finally, the user may also request more information about a given process or product object. Figure 4.4 illustrates how the object model for the activity “Major Design Review” (process “step_major_design_review”) would be displayed to the user.

³The terms “work context” and “working context” are used by various SEEs (see MERLIN [PSW92] and Process Weaver [Fer93]) with dramatically different meanings.

Process-specific work context for process step_major_design_review

Process `step_major_design_review` in project `ISPW_with_measurement`
is of type `Major_Design_Review`

Comment for this process:

-- A major design review is conducted on a design product

Current process state:

Attribute `status` of type `process_status` has value `disabled`
Attribute `effort` of type `Process_effort` has value `0`

Products:

Product `p_fd` of type `Feedback_Document`
Attribute `status` of type `Product_status` has value `non_existent`

Product `c_dd` of type `Design_Document`
Attribute `status` of type `Product_status` has value `non_existent`
Attribute `size` of type `Product_size` has value `0`
Attribute `complexity` of type `..._complexity` has value `0`

Product `c_rd` of type `Requirements_Document`
Attribute `status` of type `Product_status` has value `complete`

Entry criteria:

(c_dd.status = `complete`) and
(c_dd.size > 10) and
(p_fd.status = `non_existent`)

Update

Help

Close

Figure 4.3: Context of activity "Major Design Review"

Object model for process
step_major_design_review:

```
process_model Major_Design_Review(effort_0: Process_effort) is
  -- A major design review is conducted on a design product

  process_interface
    imports
      product_model Requirements_Document, Design_Document,
        Feedback_Document;
      process_attribute_model Process_effort;
    exports
      effort: Process_effort := effort_0;
    product_flow
      consume
        c_rd: Requirements_Document;
        c_dd: Design_Document;
      produce
        p_fd: Feedback_Document;
      consume_produce
    context
      entry_exit_criteria
        local_entry_criteria
          (c_dd.status = 'complete') and
          (c_dd.size > 10) and
          (p_fd.status = 'non_existent');
        global_entry_criteria
        local_exit_criteria
          p_fd.status = 'complete';
        global_exit_criteria
      end process_interface
end process_model
```

Close

Figure 4.4: Viewing the object model for process “step_major_design_review”

```
xterm
Date: Sun, 14 Aug 94 9:52:48 MET DST
From: mvps@informatik.uni-kl.de
Subject: Request for data
To: lott@informatik.uni-kl.de
Status: R

MVP-S REQUEST
-----

To: lott
Project: ISPW_with_measurement
Role: Design_Engineer
Authorization: 19940408a3cc4

Message: amount of effort in hours
-----

Please fill in the value for the requested attribute, restricting your
changes to the field below, and return this form to the address
mvps@informatik.uni-kl.de

Please enter value for attribute 'effort' of attribute type 'Process_Effort'
HERE->_____

Thank you!
Mail> █
```

Figure 4.5: E-mail request generated by the process engine

Process engine

The MVP-S process engine interprets a textual MVP-L process view, creates an internal representation, handles requests to query and manipulate the current project state, and maintains the project state across shutdowns of the host computer. To manage the project state, the process engine stores the values of all process, product, and resource attributes. The process engine waits for events generated by the developers, reacts to the events appropriately, and sends back the result of the events via the information channel provided by the user interface. An event may be a process completion, a change of an attribute value, or a request for information. In the prototype, events are processed in serial, not in parallel.

Automated support for collecting and using data

The MVP-S system offers various forms of support for collecting and using data. The data may be collected either from an on-line product or from a person. Data from an on-line product is collected synchronously by invoking a measurement tool. The task of invoking data-collection tools is delegated to the user interface because the process engine does not necessarily have access to the work products on the user's machine. This is an example of synchronous data collection: in response to a request from the process engine, the interface invokes the data-collection tool and waits for a response. The data is directly accepted by the SEE. A forms-based interactive tool that queries a person for data may also be called via this mechanism, as can a tool that collects measurement data from artifacts within the computer system.

However, personnel are not software products. Data is collected from personnel by requesting it, asynchronously, using electronic mail. When the process engine discovers that it needs data from an individual, it generates a mail message and gives it to the machine's operating system to

be delivered, thus bypassing the MVP-S user interface. An example e-mail request for data is shown in Figure 4.5. This lets us reuse both a well-known interface as well as existing spooling facilities. The individual is expected to respond to the data request and eventually to send it back to the process engine. The process engine receives the mail, parses the message, and extracts the new data.

Empirical data are primarily used in MVP-S in logical expressions that are written into the process view, as expressed in MVP-L. These empirical data serve as target values for processes. The system can provide guidance by explaining these target values at the outset of a process and offer feedback by comparing collected values with target values that are encoded in a process's entry and exit criteria. For example, limits on code complexity or total effort may be entered as exit criteria of a coding process. The user can obtain information about these targets at any time, ideally at the outset of the process. Then based on comparisons of collected data with the target values, the system can offer the user additional feedback about how well the goals of the process were met.

Satisfaction of measurement-related requirements

The capabilities of the system are summarized here based on the requirements stated in Section 3.6. Those requirements express desirable capabilities for the collection and use of empirical data in a process-centered software engineering environment. This section explains to what extent MVP-S satisfies those requirements. One caveat here is that judging whether a requirement is satisfied is sometimes subjective.

- R1: Specification of triggers: Satisfied; simple triggers such as a process completion or time of day can be specified.
- R2: Recognize occurrences of triggers: Satisfied; all specifiable triggers can be recognized.
- R3: Invoke tools to collect data: Satisfied; any tool can be invoked, but the generation of appropriate arguments for that tool (e.g., the file name of the appropriate document) is not supported well.
- R4: Coupled to on-line products: Weakly satisfied; MVP-S may be coupled to on-line products, but it involves hard-coding a file name in a model.
- R5: Store data in a database: Not satisfied; data are stored in simple files.
- R6: Retrieve data from database: Not satisfied.
- R7: Accept data via multiple paths: Satisfied; data are accepted from tool invocations and from people.
- R8: Notify people of data need: Satisfied through use of e-mail as a notification mechanism.
- R9: Accept data asynchronously: Satisfied through use of e-mail as a delivery mechanism.
- R10: Function with incomplete data: Trivially satisfied; the system does not recognize that its data are incomplete and makes no allowances for this.
- R11: Query people using forms: Satisfied through the capability of invoking any measurement tool.

- R12: Explain data definitions on-line: Satisfied; data are explained either by a forms-based tool or in a mail message requesting data.
- R13: Explain goals behind data: Not satisfied.
- R14: Allow identities to be masked: Not satisfied.
- R15: Minimize intrusiveness: Partially satisfied through the use of e-mail as a notification mechanism.
- R16: Formal definition of processes: Satisfied by the use of MVP-L models.
- R17: Present information about objects: Satisfied through the use of process models, product models, and resource models encoded in MVP-L.
- R18: Present current state of objects: Satisfied; all attribute values from all objects are reported.
- R19: Query project state: Satisfied; the user can ask about all modeled aspects of the project, and all attributes of all objects.
- R20: Accept notifications about changes in project state: Satisfied by the capability to accept notifications of process starts, process completions, and empirical data.
- R21: Consistency of project state with reality: Satisfied, but relies wholly on users.
- R22: Track reality to detect deviations: Satisfied, but limited to information supplied by users.
- R23: Support recovery during replanning: Not satisfied.
- R24: Assignment of personnel resources to processes: Not satisfied; current work will extend the system to satisfy this requirement.
- R25: Support grouping of personnel into teams: Not satisfied.
- R26: Use specifications of personnel qualifications: Not satisfied.
- R27: Inform personnel of assignments: Not satisfied; current work will extend the system to satisfy this requirement.
- R28: Permit substitutions: Not satisfied; current work will extend the system to satisfy this requirement.
- R29: Support the execution of hierarchically structured views: Not satisfied; current work will extend the system to satisfy this requirement.
- R30: Handle exceptions: Not satisfied.

4.2.2 Possible extensions to the prototype

The MVP-L formalism lacks a flexible mechanism for specifying the correspondence between model-world representations of objects and real-world objects. Instead, file names must be hard-coded into process models. Because of this deficiency, the MVP-S prototype is poorly coupled with on-line work, and automatic data collection from work products is not supported elegantly by the prototype. Ideally a user could point the system to some product to be measured at any time during the project.

We would like to have graphical representations of processes, products, and resources similar to those in Mi and Scacchi's work in order to improve comprehension of this information [MS92]. The current status of each activity could be indicated graphically, for example by using different colors. The products that are produced by some activity, but do not yet satisfy the exit criteria of that activity, could be highlighted to show the developer which products still require work.

Future work will investigate support for managerial roles. The system must allow a person playing a managerial role to constrain resource assignments or to perform one-to-one matches with personnel and process instances. In the current implementation, personnel who play technical roles may bind themselves to any process for which a match exists between the person's role and the role demanded by the process. Also, support is needed for handling exceptions involving the forced start of a child process (refinement of some process) or the forced halt of a parent process (abstraction of a collection of processes).

4.2.3 Example of providing measurement-based feedback

We demonstrate the advantages brought by the use of role definitions and measurement by presenting an example of how a developer uses MVP-S. For our process we reuse the "Software Process Modeling Example Problem" that was developed for the Sixth International Software Process Workshop [KFF⁺90]. In this scenario, a team of maintainers cooperates to design, code, and test a maintenance change to a software module. Figure 4.6 presents a view of the process showing the major control flow relationships among the individual process steps. We assume that the organization is interested in learning about faults, their causes, and their repair costs. Empirical models were already developed using historical data to describe characteristics of the software system and their relationship to faults, namely design complexity, expected fault rates in changed code, and effort to implement different types of changes. All of these goals can only be tracked if empirical data is collected from the work activities.

We present two variations on this problem. The first involves enacting the processes without measurement data. The second variations shows how the processes may be enacted with the support of measurement data. Both variations are restricted to the two activities "Modify Design" and "Review Design;" they differ in the information provided for choosing a specific type of review. If the change was small, the review is accomplished without calling a meeting and is named "Minor Design Review." If the changes were large,⁴ the reviewers will need to spend extra preparation time before the review and will meet to review the design; this is named "Major Design Review." We refine the original process step "Review Design" into the two steps mentioned above. Figure 4.7 shows a view of the data flow in the refinement. In the first variation, no support is given for selecting a review process. In the second variation, the design engineers

⁴"Small" and "large" will be quantified shortly.

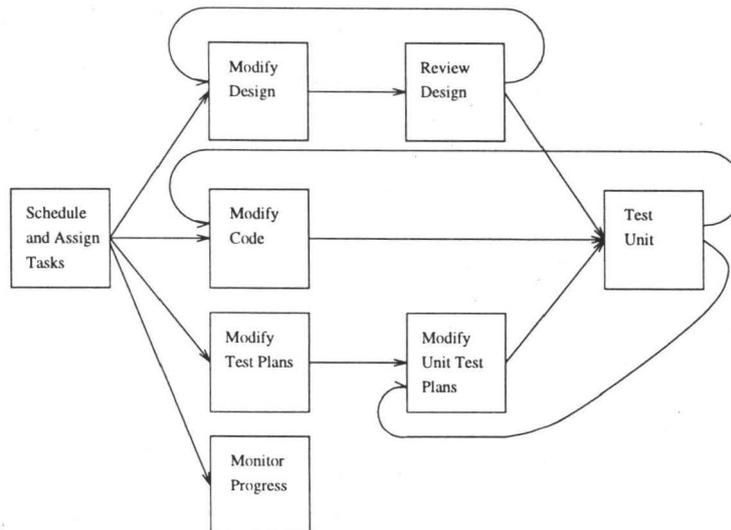


Figure 4.6: Control-flow view of the ISPW activities (from [CML⁺90])

use measurement data to select the most suitable review process.

Enaction Without Measurement Data

The first variation uses a process view with simple goals defined for activity “Modify Design” (process “step_modify_design”), namely that its outputs must be provided. Entry constraints for both the major and minor review processes are also simple, namely that the inputs must be available, so both have the same enabling condition. The criteria that allow personnel to distinguish objectively between small and large changes cannot be represented explicitly without the use of measurement.

Activities begin with the initial state shown in Figure 4.6 after the steps have been assigned. One of the engineers notifies MVP-S that the activity “Modify Design” (process “step_modify_design”) has begun by sending the event “start.” A design or quality assurance engineer may request additional information from MVP-S about that activity. MVP-S answers with an activity-specific work context showing the products consumed and produced, the current state, and the goal of the activity as shown in Figure 4.8.

Eventually the design engineer decides that the activity “Modify Design” (process “step_modify_design”) is complete and notifies MVP-S by sending the event “complete” to that process. In the resulting project state, both review activities are marked by the system as enabled because their inputs are available. Figure 4.9 shows the role-specific work context for a quality assurance (QA) engineer, who sees that both design review steps are enabled. The assumption is that the QA engineers will know that only one of the review steps should be selected and enacted. The choice of step to enact is left to the experience and intuition of the engineers who participate in that process step. We do not wish to argue against individual control. However, this juncture is a great opportunity for the SEE to supplement the individual’s experience and intuition with concrete information about their obligations and which process should be enacted next according to the process view. This information can only be provided to personnel if the organization’s implicit knowledge can be made explicit in the process view.

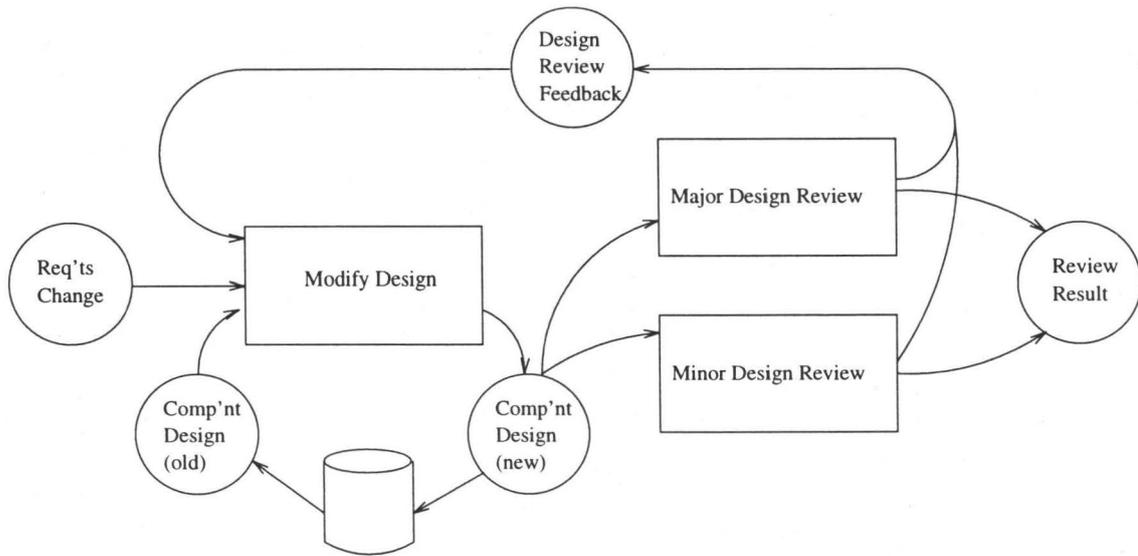


Figure 4.7: Data-flow view of the detailed example

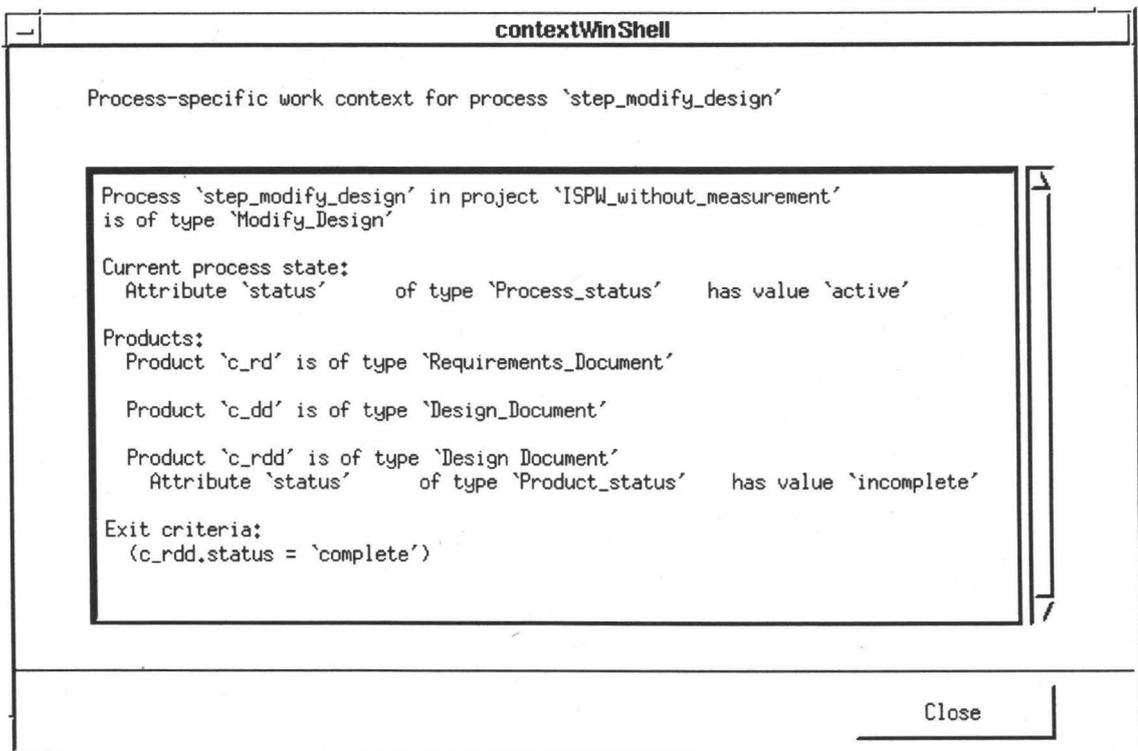


Figure 4.8: Work context for activity "Modify Design," no quantitative feedback

MVP-S Developer Interface		
action info plan help quit		
USER NAME: agse	PROJECT PLAN: ISPW_without_measurement	ROLE: QA_Engineer
Process ID	Process model	Process status
step_major_design_review	Major_Design_Review	enabled
step_minor_design_review	Minor_Design_Review	enabled
step_modify_test_plans	Modify_Test_Plans	enabled
step_modify_unit_test_plans	Modify_Unit_Test_Plans	disabled
step_test_unit	Test_Unit	disabled

Figure 4.9: Role-specific work context for a QA engineer, no quantitative feedback

Enaction With Measurement Data

The second variation augments the first with objective, measurable goals for the activity “Modify Design” (process “step_modify_design”) and establishes quantitative entry constraints for the two design review steps. Using measurement data to enact this example results in at least three advantages both for the developers and for the organization. First, the state and current situation of the activities can be described more precisely. Second, specific goals of each activity can be presented to the engineers during the time that they perform them. Third, the SEE can use data to provide feedback about which of the two design review steps should be performed after the design-modification activity is completed.

Our choice of constraints is based on the following goals and assumptions. First, data should be collected about design quality aspects to help predict code complexity and maintainability. Second, the organization wants to collect data about the effort (i.e., cost) required to make the change. Third and finally, a review process that is appropriate for detecting defects in that module’s design should be chosen; we assume that a minor review is expected to be sufficient when fewer than 30% of the lines were changed, and that a major design must be done otherwise.

Based on these goals and assumptions, we choose to measure the engineer’s effort as well as the design quality aspects of module coupling, module cohesion, information hiding, number of lines changed, and the total size; all of these depend on a formal design representation. We specify that effort data must be collected from the person performing the design change upon completion of that activity. We assume that the SEE is coupled to the work environment and can collect data from the design document using measurement tools. Data that can be collected automatically are the size, the number of changed lines, and the values for the coupling, cohesion, and information hiding metrics. Data requested directly from the design engineers is the effort spent in changing the design.

This variation also starts from the initial project state sketched in Figure 4.6. The addition of measurement data makes it possible to provide the design engineer with highly specific information about the design-modification activity after it has been started. Figure 4.10 shows how the information about the current state and specific goals of the activity “Modify Design”

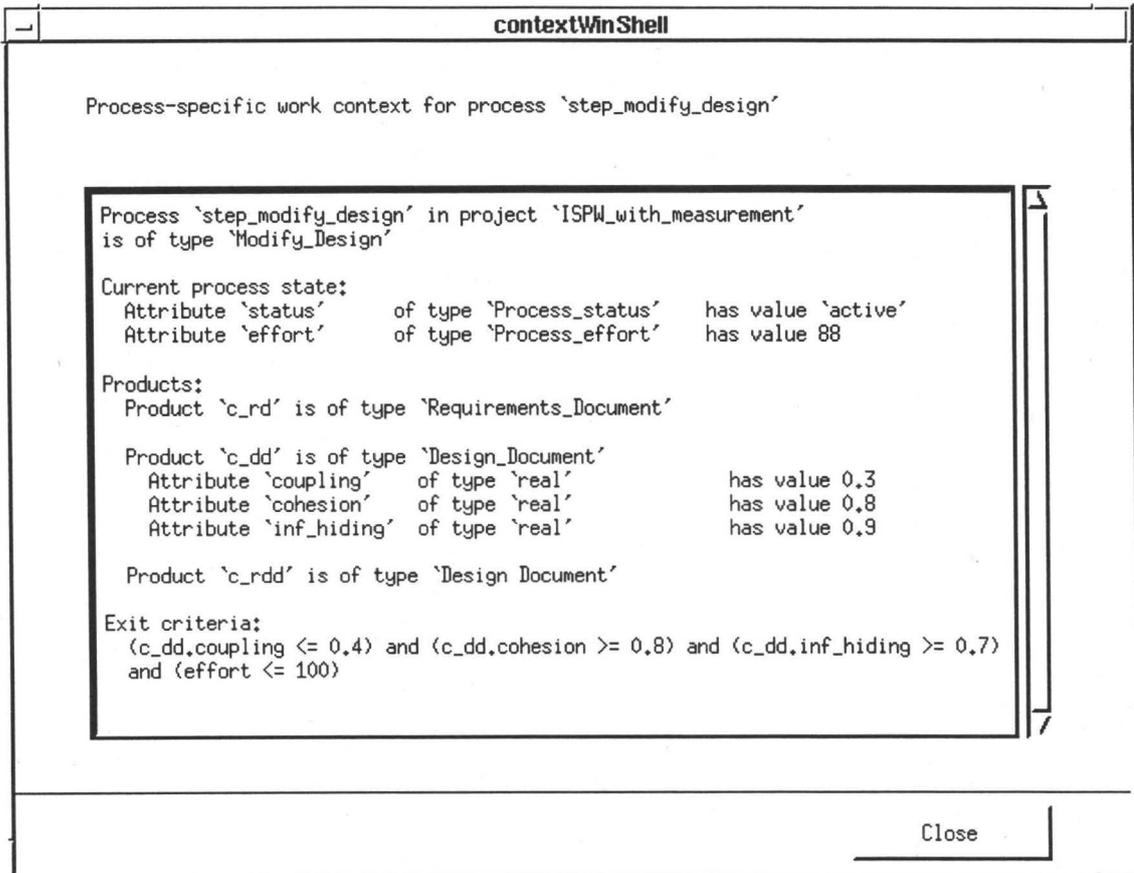


Figure 4.10: Work context for activity “Modify Design,” with quantitative feedback

(process “step_modify_design”) are presented to a person playing the role of the “designer.” The differences with respect to Figure 4.8 include showing values for the design-quality attributes “coupling,” “cohesion,” and “inf_hiding” as well as the definition of the goals of the activity (the exit criteria) using empirical data.

The design engineer eventually decides that the activity “Modify Design” (process “step_modify_design”) is complete, and correspondingly notifies MVP-S of the completion of the process step. In response to this trigger, the SEE gathers data about the design product (coupling, etc.) by invoking measurement tools and gathers data about the effort required by sending a mail request. Figure 4.5 illustrates an e-mail request for effort data that the process engine sends to the designer.

We assume that data gathered from the design document about coupling, cohesion, and information hiding was within the expected bounds, and that the effort data collected via the e-mail request was also within the preestablished limit. The data for the number of lines changed show that the previously chosen 30% threshold was exceeded. Therefore, the SEE informs the developer that the activity “Major Design Review” (process “step_major_design_review”) is the appropriate next step according to the process view. Figure 4.11 shows the MVP-S window with this feedback. Figure 4.3 (page 85) shows our conception of the information that the SEE should provide to quality assurance engineers who request an explanation about why that particular process step is enabled. In the figure, attribute values from products consumed by the process step “Major Design Review” are displayed. This information can improve coordination among

MVP-S Developer Interface		
action info plan help quit		
USER NAME: agse	PROJECT PLAN: ISPW_with_measurement	ROLE: QA_Engineer
Process ID	Process model	Process status
step_major_design_review	Major_Design_Review	disabled
step_minor_design_review	Minor_Design_Review	enabled
step_modify_test_plans	Modify_Test_Plans	enabled
step_modify_unit_test_plans	Modify_Unit_Test_Plans	disabled
step_test_unit	Test_Unit	disabled

Figure 4.11: Role-specific work context for a QA engineer, with quantitative feedback

the participants in this process step because all affected personnel understand that, according to the process view, the more detailed review process is most appropriate for finding defects in the newly modified design document. By using quantitative criteria, implicit information was made explicit for all personnel.

4.3 Assumptions About Automated Support

This section states a series of assumptions about the interactions between a user and an automated system that offers process-centered feedback. These assumptions refine the goal of providing guidance and feedback to teams who cooperate on software development projects. The purpose of stating these assumptions is to permit them to be tested in the context of an empirical study. The acceptance or rejection of these assumptions will identify future research directions for measurement-based guidance and feedback in particular, and for process-centered software engineering environments in general.

In the course of working with a process-centered system, a user interacts with a specific collection of text and shapes on a computer screen called a user interface. We distinguish between problems specific to one particular implementation of a process-centered software engineering environment and issues fundamental to the approach. To help in separating the two issues, aspects of the MVP-S user interface are summarized first. Then a set of assumptions are stated about the interaction between a software developer and a process-centered software engineering environment, with some dependencies on MVP-S.

4.3.1 Summary of the MVP-S user interface

This section summarizes the MVP-S user interface. This summary is provided because it may help explain the results of a study that evaluates MVP-S.

All text, no graphics. MVP-S uses a window, icon, menu, pointer (WIMP) interface. However, it is limited to showing textual information about processes, products, and resources. It cannot display any graphical representation of that information. These textual representations are expected to be adequate for giving a user a comprehensive understanding of the relevant objects.

Passive role. The role of the system is limited to providing information to personnel when they request it. MVP-S does not actively open windows or take any action on its own to tell personnel what to do. However, it does have the facility to request data from personnel via E-mail, as discussed earlier.

No control of tools or products. MVP-S makes no restrictions on access to on-line tools or to the products manipulated by those tools. In other words, developers have direct access to design documents, source code, test cases, etc.

Single level of refinement. The version of MVP-S described here does not support the execution of hierarchically structured process views. Only the top-level process model (the project plan) can be refined into a collection of submodels. This means that hypotheses concerning hierarchies of processes cannot be tested.

4.3.2 Assumptions about the use of MVP-S

This section states assumptions about the use of a process-centered software engineering environment, with special attention to MVP-S. A number of these assumptions will be tested in the empirical study discussed in Chapter 5.

Assumption 1: Sufficient information in models. We assume that sufficient detail about a process can be captured in models so that feedback can be provided. In other words, information sufficient for the software developer to understand and perform a given process can be captured in a set of explicit models. In the work discussed here, the models are built using the MVP-L process modeling language.

This assumption might be tested by questioning users of such a system during and after they have completed some process. The users could be asked whether they felt they had sufficient information, whether they were forced to infer significant amounts of information to accomplish the task, and what information they felt was missing.

Assumption 2: Deviations. The assumption here is that deviations from the prescribed way of performing a process are rare. Stated differently, developers can usually work in the context of a previously prepared set of models without replanning. The activity of replanning means reworking the plan to match unforeseen circumstances of the current project. This assumption is expressed in MVP-S in that the system does not support replanning, nor delegation of tasks to others.

This assumption concerns processes in-the-large, and cannot be tested credibly in a small process of short duration. This assumption could only be tested realistically in the context of

a revenue-producing project by examining the number of times a project must deviate from a predefined process. Some existing work on "process waivers" indicates that deviations from a standard process are a regular part of any complex process [VZ95]. That evidence suggests that this assumption is unrealistic.

Assumption 3: Acceptance of on-line feedback. The assumption is that software development personnel are willing to accept an on-line representation of their expected tasks as feedback about what they should do. The operational definition of acceptance used here states that personnel like being provided with information, believe the information that they are given, and adjust their behavior accordingly.

We believe that the issue of acceptance is strongly influenced by whether personnel feel that their creativity is restricted or otherwise limited by such a system. This assumption may be refined into acceptance of guidance and feedback. Guidance might take the form of a statement about what tasks are expected, and feedback might take the form of a statement that a certain task was (or was not) performed in accordance with some plan. The acceptance of these two types of information should be treated separately.

The assumption of acceptance might be tested by having people work with the system and then asking them questions. Acceptance in the short run can be evaluated by asking whether people found the information helpful and by observing whether people adjusted their behavior according to the information provided. Long-term acceptance might be evaluated using the number of interactions between users and the system to detect patterns such as increasing or decreasing use with time.

Assumption 4: Quantitative criteria. The assumption here refines the assumption concerning acceptance of feedback, and states that people believe and like receiving quantitative feedback, especially for starting and terminating processes. The basis for comparison here is simple criteria such as the availability of products. For example, a developer finds it more helpful to know that a series of test cases should cause 85% statement coverage during the test, rather than just the test cases should provide "thorough" coverage.

This assumption might be tested by supplying subjects with information that is extremely relevant to some process. For example, subjects might be given a defect model that states the number of expected defects per hundred lines of code. This defect model would estimate very closely the number of defects actually present in some object under review. If subjects keep reviewing until they find that many, and stop reviewing after they have found them, this is an indication that they believed the information and found it useful.

Assumption 5: Timely interaction with feedback system. The assumption here is that people will report their actions to a process-centered system in a timely manner. First, the users will report when they start and complete processes. Second, they will supply data within a reasonable time after a request. This is especially important in a system like MVP-S that relies completely on personnel to report their actions.

This assumption might be tested by observing how much time a user allows to elapse after finishing a task and notifying the system of that task's completion. Similarly, data could be requested from subjects, and the amount of time elapsed before the subjects supply that data

could be measured. An empirical study could begin to collect data about the variance in response times, which would help build an operational definition of "timely."

Assumption 6: Awareness of the process and its context. The assumption here is that the use of a feedback system will make the users highly aware of the context (i.e., input products, output products, dependencies on other personnel, etc.) in which they work, and encourage a deep understanding of the process.

This assumption might be tested by having subjects use the system to achieve some goal and asking them questions during and following completion of the task. The questions might ask the subjects to describe a process and its context. Evaluating the tenability of this assumption would be based on the thoroughness with which the subjects could recall and describe the processes that they had so recently performed.

Assumption 7: Speed of comprehension. The assumption here is that developers can comprehend the process information offered by an on-line feedback system as quickly as directions written on paper. Process information consists of the inputs, outputs, goals (criteria), and other information about a process.

This assumption might be tested by evaluating the amount of time users require to accomplish some process using a feedback system and using paper directions. If the users require less or the same amount of time to accomplish the process when using the feedback system, this would suggest that speed of comprehension is not a problem. However, this issue is also intimately tied up with a particular user interface. A poor user interface in an on-line system could easily skew the results.

Assumption 8: On-line data collection. The assumption here is that on-line data collection improves the validity of the data that people supply. The idea is that people supply data soon enough that they can remember the values for the data. When paper data-collection forms are used, the assumption is that people may wait until the memory of the activity from which the data should be collected has faded.

This assumption might be tested by requesting data from subjects in the course of performing some process, and observing them in that process to collect the same data independently. The observation is necessary to evaluate the validity of the data.

Assumption 9: Unknown processes. We assume that the effects of using a process-centered system are most pronounced when it supports personnel who perform a process that is unfamiliar or entirely unknown to them. The idea is that the personnel are most dependent on the feedback from such as system in this situation. In a situation with a well-known process, we assume that personnel would only use the process-centered system to remind themselves of forgotten details.

This assumption might be tested by having a subject perform a number of similar but initially unknown tasks, all with support from a process-centered system. Data could be collected about the amount of time the subject spends using or querying the system to obtain information about the process. This assumption would be supported if the subject were to spend far less time on the last replication of the task than the first.

Assumption 10: Effectiveness. Finally, we assume that the use of measurement-based feedback yields measurable benefits in how personnel accomplish their work. We assume that the feedback improves the person's clarity of purpose (process improvement: less effort, less rework, less computer resources) and the person's understanding of quality goals (product improvement: fewer faults, more understandable result, quality goals more closely achieved) as compared to a person who uses a off-line (paper) set of directions.

This assumption could be tested by comparing the work done by personnel who use a process-centered system with personnel who use more conventional methods. Data could be collected about the amount of effort, the defects found, the personnel's subjective evaluations of satisfaction with the project, and related issues. An experiment to test this assumption would develop initial data concerning the cost/benefit ratio of developing and using these environments.

4.4 Summary

This chapter introduced MVP-S and demonstrated the benefits of using integrated measurement and process views to provide measurement-based guidance and feedback. MVP-S is the first implementation of our requirements for collection and use of empirical data in SEEs. Finally, a series of assumptions were stated about how users interact with a process-centered software engineering environment. Explicit statements of such assumptions will permit evaluating them in an empirical study.

Design, development, and refinement of the MVP-S system was a joint effort on the part of the author and his master's degree students at the University of Kaiserslautern. The assumptions about user interactions were defined based on that experience.

An empirical study that evaluated the acceptance and effectiveness of MVP-S is presented in the next chapter.

Chapter 5

Empirical Evaluation of Measurement-Based Feedback

Much work has been done on developing process-centered software engineering environments, but few studies have demonstrated measurable, objective benefits of their use. This chapter discusses a controlled experiment that compared the use of measurement-based process guidance and feedback as implemented off-line (on paper) versus on-line (using MVP-S). Because relatively little empirical work has been done on process-centered systems, the research presented here is best classified as exploratory, and can be expected to uncover more questions than it answers.

Section 5.1 gives an overview. A simple characterization scheme for experiments in software engineering is presented in Section 5.2 to set the context for the study. Section 5.3 states five hypotheses based on the assumptions from Section 4.3. Section 5.4 states the plan for the experiment including the design, Section 5.5 discusses procedures used while conducting the experiment, and Section 5.6 presents results and interpretations. The instruments (instruction sheets, code objects, forms, etc.) used in the experiment are shown in Appendix A.

5.1 Overview

The fundamental hypothesis concerning process-centered software engineering environments is that they improve communication and reduce problems among teams who cooperate to achieve some task. This hypothesis might be evaluated both in uncontrolled and controlled experiments that complement each other's strengths and weaknesses. To attain maximum external validity, a field study (uncontrolled experiment) might follow at least two teams through large tasks. To attain maximum internal validity, a controlled experiment could follow many individuals through small tasks. A field study would be quite expensive, and controlling the threats to internal validity would be difficult. A smaller, more controlled experiment could help the community understand how to manage the threats involved in a larger field study. Therefore, the study described here focused on understanding how a single user interacts with a process-centered system.

A controlled experiment was conducted that compared off-line and on-line implementations of measurement-based process guidance and feedback. Each subject used an off-line guidance

technique and an on-line guidance technique as assistance while performing an unfamiliar exercise and while working alone. The off-line version of each exercise was supported by paper instruction sheets and paper question sheets. The on-line version of each exercise was supported by MVP-S, our prototype process-centered software engineering environment, and on-line question sheets. Each exercise involved testing a small program according to a set technique, namely functional testing or structural testing. The primary differences between the two exercises performed by one subject were the source of the guidance (off-line versus on-line), the means of data collection (again off-line versus on-line), and of course different programs.

The testing exercises consisted of generating test cases, executing them, and evaluating the output to find failures. In one exercise, principles of functional testing were applied to a program named "cmdline." In the other exercise, principles of structural testing were applied to a program named "tokens." The exercises were chosen to require a similar amount of time to complete, with a minimum of about one hour. In both cases the subjects used the computer to test the program with their test cases. Subjects were recruited from the regular and student employees of the Research Group for Software Engineering at the University of Kaiserslautern. The 20 subjects who completed both exercises were not professional testers, and had never applied these particular test techniques in a systematic way.

Many of the assumptions presented in Section 4.3 about how users interact with a process-centered software engineering environment were reformulated as hypotheses to be tested in the experiment. The main hypotheses and corresponding results are summarized next:

- Hypothesis: Subjects accept an on-line system for guidance and feedback.

Result: The data did not support this hypothesis. Acceptance of the on-line system among the subjects was measured at approximately 50%. The two groups stated remarkably consistent reasons about their work styles that lead to their preferences. Members of the group that preferred the off-line version stated that they preferred having an overview of the exercise to be done, and that the overview was easy to achieve with paper. Members of the group that preferred the on-line version stated that they enjoyed the "filtering" aspect of MVP-S because they did not have to search through paper sheets for information. These results suggest that automated support for measurement-based feedback is feasible, but that much work remains to be done in satisfying the demands of all work styles in order to improve acceptance of the approach.

- Hypothesis: Subjects adjust their behavior based on quantitative quality models.

Result: The data did not support this hypothesis. Responses ranged from "very helpful" to "irritating and harmful." Again there were two easily identifiable groups that corresponded to these results. Members of the group that found the quality models helpful attained results that were very comparable with the information that was supplied in advance as guidance. Members of the group that found the quantitative quality models harmful attained results that deviated sharply from the information. Only the group that found the quality models helpful adjusted their behavior.

- Hypothesis: Subjects accomplish their work faster and better when using on-line guidance and feedback as compared to off-line guidance and feedback.

Result: The data do not support this hypothesis. Analysis of the data revealed no significant differences in the duration or quality of the exercise performed by subjects who used the off-line and on-line guidance and feedback techniques. However, subjects who work alone,

uncoupled from the work of others, need no notification of critical changes or other outside influences. We expect that the real benefits from using these systems will be derived from using them in the context of a team. Therefore, we feel encouraged that the system was not a hindrance to subjects who worked alone.

We draw the conclusion that measurement-based process feedback can realistically be used with automated support. However, much work remains to develop a highly usable interface and to overcome problems that cause people to reject the use of an on-line guidance and feedback technique.

5.2 Characterization of Experiments

Many factors can be explored in an experiment that evaluates how a person can be assisted while performing a software engineering task. Table 5.1 characterizes some of these factors [Bas94]. Collections of factors are considered to be an approach. The approaches listed here include an *ad hoc* approach (i.e., no supportive methods or other information are used), methods (e.g., structured programming), empirical models (e.g., defect profiles), goals (e.g., "write fast code"), process definitions (e.g., a process view based on MVP-L), tracking and evaluation (e.g., collecting and evaluating data from process), and replanning (e.g., information about how to recover from some situation). Some of the individual factors can benefit from automated support.

The levels in the table constitute a simple hierarchy. Each level forms a new approach by adding a single factor to the preceding approach. An experiment can be designed to compare two vertically adjacent approaches because only one factor changes. However, it would be more difficult to design an experiment that compares two horizontally adjacent approaches because they differ in more than one factor.

The study presented in this chapter compared the use of off-line and on-line feedback (without versus with automation). It can be characterized using the schema of experiments presented in Table 5.1 as a comparison between level 5a and level 6a.

5.3 Hypotheses and GQM Plan

In the following discussion, hypotheses are stated that are based on the assumptions from Section 4.3, and a GQM plan is developed for the study. In the GQM plan, a study goal is stated, and it is then refined operationally into questions and metrics according to the principles articulated in the GQM Paradigm.

5.3.1 Hypotheses

The study presented in this chapter investigated a set of hypotheses that must be tested before the use of SEEs in teams can be evaluated. The focus is on understanding how a single user reacts to measurement-based process guidance as supplied by an on-line system. The following hypotheses incorporate ideas from the assumptions listed in Section 4.3. Due to the limited scope of this study, not all of the assumptions stated there can be evaluated. For example, assumption

(1a)	Ad hoc		
(2a)	Methods	(2b)	Empirical models
(3a)	Methods Empirical models		
(4a)	Methods Empirical models Goals	(4b)	Methods Empirical models Process definition
(5a)	Methods Empirical models Goals Process definitions		
(6a)	Methods Empirical models Goals Process definition/automated	(6b)	Methods Empirical models Goals Process definition Tracking and evaluation
(7a)	Methods Empirical models Goals Process definition/automated Tracking and evaluation		
(8a)	Methods Empirical models Goals Process definition/automated Tracking and evaluation Replanning/automated		

Table 5.1: Hierarchical characterization schema for software engineering experiments

A2 concerning deviations is not tested because absolutely no deviations are expected during the course of a carefully planned exercise of short duration.

Hypothesis 1: Subjects accept guidance and feedback from an on-line system (MVP-S) as willingly as conventional instructions on paper (assumption 3).

Hypothesis 2: Subjects comprehend on-line versions of guidance and feedback better than they comprehend conventional, off-line instructions (assumptions 1 and 6).

Hypothesis 3: Subjects take quantitative quality models such as a suggested maximum amount of time for a step seriously, and adjust their behavior accordingly (assumption 4).

Hypothesis 4: Subjects supply data to a process-centered software engineering environment about starting and ending processes, as well as other data about those processes, on a timely basis (assumptions 5 and 8).

Hypothesis 5: Subjects who use an on-line guidance system accomplish their work more quickly and do a better job than subjects who use off-line instructions (assumptions 7 and 10). Hypotheses 1 is a vital prerequisite for testing this hypothesis. In other words, if subjects reject the use of on-line guidance, it is not meaningful to analyze their speed when using such as system.

5.3.2 Goal

Next a goal is stated for the study. The structure of the study goal follows the template for GQM goals:

Analyze **off-line and on-line measurement-based process guidance**
for the purpose of **comparison**
with respect to **their acceptance and effectiveness**
from the point of view of the **researcher (system builder)**
in the context of **small exercises performed by subjects in a controlled experiment.**

5.3.3 Questions

The following series of questions provide an operational refinement of the goal into a series of metrics. The questions were derived using the previous list of hypotheses.

- Q1. What is the subject's preference with respect to the guidance technique? (from hypothesis 1)
- Q2. Do subjects feel that the creative portions of the work were restricted by use of the guidance technique? (from hypothesis 1)

- Q3. To what extent is a textual representation of a process adequate for understanding quickly what is expected in that process? (from hypothesis 2)
- Q4. How difficult is it for subjects to comprehend the directions offered to them? (from hypothesis 2)
- Q5. How aware are subjects of the context of a process (inputs, outputs, termination criteria, etc.)? (from hypothesis 2)
- Q6. How do subjects rate the helpfulness of the quantitative quality models that are supplied as guidelines (start and termination criteria) for processes? (from hypothesis 3)
- Q7. Do subjects adjust their behavior according to the quantitative guidelines? (from hypothesis 3)
- Q8. What delays are encountered when subjects are expected to enter data about processes that they have started or completed? (from hypothesis 4)
- Q9. How well and how quickly do subjects perform the exercises? (from hypothesis 5)

5.3.4 Metrics

Some of the data needed to answer the questions listed above were gathered by asking the subject to complete question sheets, other data were gathered by observing the subject performing the exercises, and some data were collected by the computer. The following concrete data items were collected.

- M1. Subject's opinion about working with an on-line process-centered guidance and feedback system, as compared to off-line instructions (from question 1).
- M2. Subject's opinion about entering data on paper versus on screen (from question 1).
- M3. Subject's opinion about restrictiveness of the instructions (from question 2).
- M4. Subject's opinion of whether the instructions offered to them were made more or less comprehensible by being on-line (from question 3).
- M5. Subject's opinion about the usefulness of a textual representation (from question 3).
- M6. Subject's opinion about the sufficiency of supplied information (from question 4).
- M7. Observer's opinion of a subject's understanding of a process, formed by asking subject to list the inputs, outputs, and termination criteria of a given process while performing that process (from question 5).
- M8. Subject's opinion of the usefulness of the quantitative guidelines used as start and termination criteria (from question 6).
- M9. Observer's opinion of whether subjects adjust their behavior to conform to the quantitative quality models (from question 7).
- M10. Observer's opinion of how long subjects wait to report results to the guidance system or record on paper (from question 8).

- M11. Subject's subjective assessment of motivation (from question 9).
- M12. Observer's opinion of process conformance; i.e., whether a subject followed the process faithfully (from question 9).
- M13. Amount of time the subjects need to complete the exercise, excluding pauses (from question 9).
- M14. Number of failures that were visible in the test output and detected by the subject (from question 9).

The challenge of this experiment is not the design. A simple design is used, as shown in Section 5.4. The *procedures* are the difficult part of this experiment. Procedures must be chosen that lead each subject to use the on-line process guidance system in a credible, natural way. This consideration heavily influences the decisions to be made about the information supplied to the subjects as well as any training and pretesting activities. Procedures are discussed in Section 5.5.

5.4 Experiment Plan

This section presents the plan for an experiment that tests the hypotheses stated in the previous section. The experiment reused work done by Kamsties and Lott on comparing defect-detection techniques [KL95a, KL95b]. Most interesting was the reuse of data that could be expressed as empirical models (quantitative quality models) about the exercises.

5.4.1 Experimental design

A randomized, within-subjects design was used in the experiment. The within-subjects design (i.e., blocked on subjects) permits the detection of differences between treatments despite wide variations in subject ability.

Independent variables

The following three independent variables are manipulated in this $2 \times 2 \times 2$ design.

- Exercise (two levels: FT/cmdline and ST/tokens). All subjects performed two exercises. Both involved testing small programs, once using principles of functional testing (FT) and once using structural testing (ST). The program to be tested (cmdline or tokens) is considered part of the exercise, and was chosen to minimize differences in exercise duration.
- Guidance technique (two levels: off-line and on-line). This was the primary variable of interest. Subjects were assisted with their testing exercises once by the off-line guidance technique and once by the on-line guidance technique.
- Order of trials with respect to guidance (two levels: off/on and on/off). The order in which subjects see the exercises is shuffled at the same time.

Exercise	Guidance technique		Ordering	
	Off-line	On-line	Off-line/On-line	On-line/Off-line
FT of 'cmdline'	S ₁	S ₂	S _A	S _B
ST of 'tokens'	S ₂	S ₁		

Figure 5.1: Experimental design

Uncontrolled independent variables include the motivation of the subject for the exercise as well as a subject's prior experience with test techniques. We attempt to measure these uncontrolled independent variables by having subjects complete question sheets.

For each subject, the design randomized the match between exercise and guidance technique, as well as the order of using the guidance techniques. To do so, subjects were randomly assigned to one of the four groups S_{1A} , S_{1B} , S_{2A} and S_{2B} . For example, a subject in group S_{1A} would perform functional testing (FT) of program 'cmdline' using the off-line guidance technique, then subsequently that subject would perform structural testing (ST) of program 'tokens' using the on-line guidance technique. The experimental design is summarized in Figure 5.1.

Dependent variables

The metrics in Section 5.3.4 serve as the dependent variables. The subjects supply most of the data for these dependent variables by completing question sheets.

Threats to validity

The primary threat to external validity is the use of an exercise of short duration. However, the testing techniques that were applied reflect state-of-the-practice techniques.

Threats to internal validity include selection and maturation effects [CS66]. A selection effect could be caused by using subjects who are predisposed towards (or against) the use of a process-centered system. However, no preliminary test was administered to the subjects that might have assessed this bias in order to avoid problems of reactivity and sensitization. Selection effects can be controlled because all subjects are observed twice (a within-subjects design). However, the drawback of multiple observations is a maturation effect that might result from a subject learning from the first exercise. Although the two exercises are structured similarly, they involve very different test techniques. The differences were expected to mitigate maturation effects.

Use of quantitative quality models

The sets of instructions for the exercises used quantitative quality models (empirical models) as additional information that was expected to be useful in the exercises. This was intended to approximate the situation in which an organization maintains a database about prior test efforts.

For example, a quality model about a test process could predict the number of failures based on the component size or complexity. All subjects received quantitative quality models, regardless of guidance technique used.

In a preset experiment using testing exercises, of course it is possible to supply perfect information about the number of failures. However, this information is not available in the typical project situation, so using that information would constitute a threat to external validity. Instead, when perfect information was available (e.g., failures), a value was supplied that was close. Other data (e.g., amount of time estimated) were derived from previous experimental trials using the same techniques and instruments. The instructions offered the subjects quantitative quality models based on “similar” components for the expected number of failures, the amount of time budgeted for the process, the number of equivalence classes (black-box testing), and the target coverage values (clear-box testing). Average values were supplied in all cases, not ranges.

5.4.2 Exercises

The testing exercises consist of generating test cases, executing them, and evaluating the output to find failures. Principles of functional and structural testing were used. A different program was used in each of the two exercises. As mentioned earlier, the exercises and programs were chosen to keep the exercise duration as similar as possible, with a target of about 60–70 minutes.

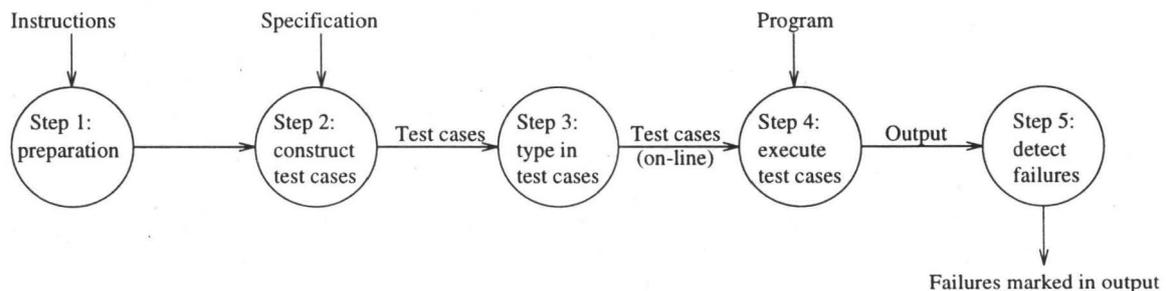


Figure 5.2: Steps in the functional-test exercise

Functional testing

Figure 5.2 illustrates the functional-testing exercise. The subjects never see the source code. In step 1 the subjects receive an instruction sheet and prepare for the exercise by fetching the needed computer files. In step 2 the subjects receive the specification. They identify equivalence classes in the input data and construct test cases using the equivalence classes, paying special attention to boundary values. In step 3 the subjects type in their test cases using a format required by a simple test harness. In step 4 the subjects use the test harness to execute their test cases, and fix any typographical or similar mistakes in their test cases. They are instructed *not* to construct additional test cases. In step 5, the subjects use the specification to detect failures that were revealed in their output. Instructions and question sheets for these steps were both on paper for the off-line version, and on the computer screen for the on-line version. The program specification and an explanation of equivalence classes were given on paper in both versions of the exercise.

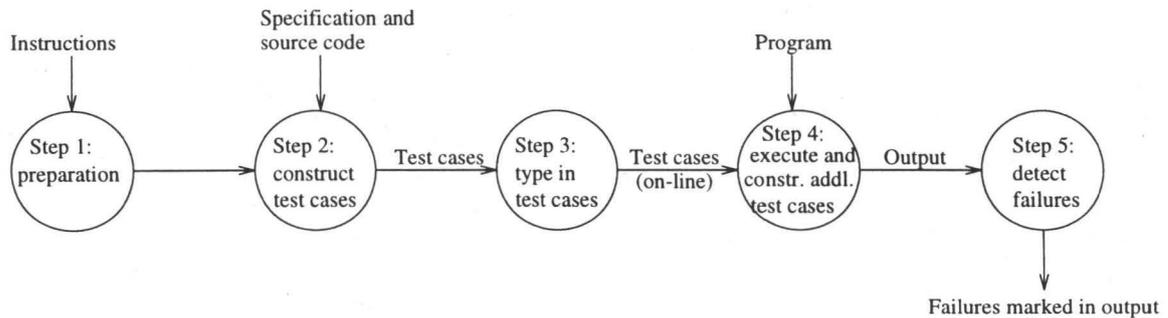


Figure 5.3: Steps in the structural-test exercise

Structural testing

Figure 5.3 illustrates the structural-testing exercise. In step 1 the subjects receive an instruction sheet and prepare for the exercise by fetching the needed computer files. In step 2 the subjects use the specification and the source code to construct test cases that will achieve 100% coverage of all branches, multiple conditions, loops, and relational operators. For example, 100% coverage of a multiple condition using a single “logical and” operator means that all four combinations of true and false must be tested, and 100% coverage of a loop means that it must be executed zero, one, and many time(s). In step 3 the subjects type in their test cases using a format required by a simple test harness. In step 4 the subjects use an instrumented version of the program to execute their test cases and view reports of attained coverage values.¹ The subjects construct additional test cases as part of step 4 until they reach 100% coverage, or believe that they cannot achieve better coverage due to various pathological cases. In step 5 the subjects use the specification to detect failures in their output. Instructions and question sheets for these steps were both on paper for the off-line version, and on the computer screen for the on-line version. The program specification, source code, and an explanation of coverage criteria were given on paper in both versions of the exercise.

5.4.3 Instruments

Many instruments were required to support the testing exercises. All are presented in Appendix A. A primary instrument was the instructions for the exercises. These instructions were offered both on-line (measurement-based feedback with automation) and off-line (measurement-based feedback without automation). The same text was offered both off-line and on-line, although formatted differently. Other required instruments are the question sheets that were used and the code objects that were tested.

The code object for the functional testing exercise, “cmdline,” checks a set of command line arguments for syntactical and semantical correctness. The code object for the structural testing exercise, “tokens,” sorts and counts words in its input. Measures of the objects are summarized in Table 5.2. The fault counts were chosen to attain a similar fault density in both programs. All faults caused unique, visible failures given suitable input data. As mentioned earlier, the instruments were chosen to keep the exercise duration as similar as possible. Normalization for

¹Instrumentation and reporting was supported by GCT, the Generic Coverage Tool, software developed by Brian Marick and available by anonymous ftp from host cs.uiuc.edu in directory /pub/testing/gct.files.

Code object	Total lines	Blank lines	Lines w/ comments	NB, NC lines	Semi-colons	Fault count
cmdline (FT)	299	34	4	261	122	10
tokens (ST)	127	9	1	117	72	5

Table 5.2: Size measures and fault counts for the code objects

Functional testing of "cmdline"			
Total time (min.)	Obs: 60, 80, 86, 55, 71, 50	Value supplied: 60	
Valid equiv. classes (count)	Obs: 7, 9, 17, 3, 20, 26		16
Invalid equiv. classes (count)	Obs: 2, 3, 6, 0, 5, 8		4
Test cases (count)	Obs: 22, 22, 15, 10, 26, 16		20
Unique failures (count)	True: 10		8
Structural testing of "tokens"			
Total time (min.)	Obs: 83, 80	Value supplied: 70	
Overall coverage (percent)	Obs: 87, 86		85
Test cases (count)	Obs: 5, 9		6
Unique failures (count)	True: 5		6

Table 5.3: Basis for quantitative quality models supplied to subjects

exercise duration resulted in a 2:1 difference in code length (the structural testers received the shorter program).

Data used to derive empirical models

As discussed in the design, the instructions provided measurement-based guidance based on prior observations and true values. This information was intended to serve as termination criteria for the successful completion of the exercises. Table 5.3 summarizes the guidance that was supplied to subjects. Information such as total time is based on observations from previous trials. Information such as the number of unique failures is based on the known, true value. The large differences in observed values for aspects such as "valid equivalence classes" shows that the previous subjects did not apply the technique very carefully. All information when supplied to subjects is explained as an approximation that is based on previous experience with similar components.

5.4.4 Subjects

A previous controlled experiment suggested that using wholly inexperienced students as subjects is fraught with difficulties [KL95b]. Therefore, the initial subjects were recruited from the full- and part-time staff of the Research Group on Software Engineering at the University of Kaiserslautern. This population consisted of people who had some experience with programming and testing. One threat to validity was that the subjects were in part self-selected. The full-time employees volunteered to participate, and the part-time employees were strongly encouraged to

participate. All subjects were experienced in window, icon, menu, pointer (WIMP) interfaces. Only volunteers who had a working knowledge of the C programming language were chosen as subjects. No knowledge of testing techniques was required.

An additional threat to validity was possible bias on the part of the subjects. All subjects had been previously exposed to principles and ideas in empirical software engineering, either through course work or research projects. They may have been biased towards accepting the use of explicit process models and quantitative quality models.

5.4.5 Data collection and validation procedures

Both off-line and on-line data methods were used to gather data from the subjects. The required dependent variables include effort recorded on forms, failures visible in test output, and subjective evaluations of the exercise. An observer also completed a form with a number of questions for each trial. After both exercises were completed, subjects were asked to complete a paper form that asked them about their personal evaluations of the exercises. The subjective evaluations include asking whether the subjects were comfortable with using the on-line system, whether they felt they understood the process, what guidance and data-collection technique method they preferred, etc. Subjects were also invited to participate in a short, individual debriefing session. This gave the experimenter an additional chance to gain explanations for a subject's preferences.

5.4.6 Data analysis procedures

The design yields two observations of each subject. A subject's performance on two exercises is more likely to be comparable than the performance of two different subjects. Therefore, we view the design as a randomized paired comparison [BHH78, p. 97-101]. To apply inferential statistical methods to quantitative data such as time and failure data, the difference between each subject's pair of data points is computed, the average difference over all subjects is determined, and a t test is applied to evaluate the probability that the sample mean is zero. The difference is computed based on the independent variables, namely the difference between functional and structural testing, between on-line and off-line guidance, or between the first and second trials. In accordance with generally accepted practice, we state that a significant result was detected if the probability value for the t-test is less than 0.05. The data for the experienced subjects was not separated from the data for the inexperienced subjects due to the small number of observations.

5.5 Experiment Procedures

This section presents the procedures that were used for conducting the experiment.

5.5.1 Training activities

No training activities with the test techniques or the guidance techniques were done, for two reasons. First, the exercise itself should be as unfamiliar as possible to the subjects. This forced the subjects to rely heavily on the information supplied by the guidance technique. Second, no

Group	Ordering of the exercises	Subjects
S _{1A}	FT/cmdline offline, then ST/tokens online	5
S _{1B}	ST/tokens online, then FT/cmdline offline	5
S _{2A}	ST/tokens offline, then FT/cmdline online	5
S _{2B}	FT/cmdline online, then ST/tokens offline	5

Table 5.4: Groups used to partition subjects

training was offered with MVP-S to avoid issues of reactivity or sensitization of the subjects to the true goals behind the experiment. It was possible for subjects to use the system with very little introduction because they only interacted with MVP-S in two simple ways. The first interaction required using a menu and a pop-up window to notify MVP-S of the start or completion of a process step. The second interaction involved using a menu to call up a window with information about a process step. No other interactions were necessary.

5.5.2 Conducting the experiment

Subjects were assigned an identification number to preserve their anonymity. A random drawing was used to assign each subject to one of the four groups shown in Table 5.4. Several subjects withdrew before completing the experiment, so recruitment continued until 20 subjects had completed both exercises. The number of subjects assigned to each group was balanced. These groups determined the match of a subject with an exercise (FT/ST + program) and a guidance technique (off- or on-line), as well as the order in which the guidance techniques were used.

All subjects were given an explicit goal of detecting as many failures (run-time behavior that deviates from the specification) in as little time as possible. This goal was written on the instruction sheets and repeated verbally. The true goal behind the experiment, namely the comparison of off-line and on-line guidance, was explained to the subjects in the final questionnaire and discussed in the individual debriefing sessions. To prevent subjects from contaminating others with knowledge about faults and such, all materials were collected after each trial, and subjects were asked not to discuss the experiment.

Experiment trials were conducted over approximately one month. Most subjects performed two exercises in a single day, and a maximum of two subjects worked in parallel on a given day. This approach permitted close contact with each subject, as well as iterative enhancement of the study over the course of the trials. However, no fundamental problems were found in the instruments or other procedures. Minor enhancements were made to the instructions and program specifications, primarily fixing typographical mistakes and other superficial problems.

Subjects who worked with off-line feedback received a package of materials that included a detailed instruction sheet, a question sheet, an explanation of the approach for constructing test cases, and the specification for the program. Structural testers additionally received a hard-copy of the program's source code. Subjects who worked with on-line feedback received a different package of materials. Instead of a detailed instruction sheet and question sheet, they received a brief explanation of how to invoke the necessary on-line systems (MVP-S for the instructions and a forms-based tool to collect data). The other materials (explanation of the approach for constructing test cases, and specification for the program and possibly the source code) were

identical to those used in the off-line technique.

To overcome effects caused by the use of an unfamiliar system, all subjects were given a 10-minute introduction to the MVP-S before they worked with it. This introduction followed a script so that each subject received approximately the same information. The introduction explained all aspects of the interface that the subjects would use. Part of the introduction involved having the subject try the two necessary interactions for themselves under the direction of the instructor.

Data was gathered during the on-line exercise in three ways. First, the subject filled out the on-line question sheet. Second, the MVP-S system sent a mail message to the principal investigator after each user interaction. Time stamps on the messages were evaluated to judge whether the system was used to record completed steps on an ongoing basis, or whether the subject waited until all steps were complete before notifying the system. Finally, the principal investigator was either in the same room with the subject(s) or close by for the entire duration of each trial. This approach was chosen to gain as much information as possible from each trial and has several benefits. First, the subject could immediately ask an expert for help. Second, the experimenter could ask the subject questions during the trial. Subjects were interrupted rather informally at least once during each trial and asked to explain, as if to an uninformed person, the current process that they were performing. The experimenter asked about input products, output products, and termination criteria in order to form an opinion about how well the subject understood that process. Third, indirect observation of the subject allowed the experimenter to collect limited data about interactions between the user and the guidance technique.

5.6 Results

The data are interpreted in terms of the hypotheses that were stated in Section 5.3.1. Based on those results, some possible improvements to MVP-S are proposed, a critique of the experiment is given, and questions raised by the experiment are discussed.

5.6.1 Data and interpretations

Only the data obtained from the 20 subjects who completed both exercises and the debriefing questionnaire are used in the following analyses. Data from subjects who failed to complete all portions of the experiment were discarded.

Hypothesis 1: Acceptance of on-line guidance

Hypothesis 1 involves the acceptance that an on-line system finds among subjects. Table 5.5 summarizes the data collected from subjects to evaluate their acceptance of on-line guidance.

The subject's preferences favored the off-line guidance technique slightly, favored the on-line data-collection technique somewhat, and the majority of subjects felt that the instructions as offered were not restrictive. There were some indications of strong feelings of acceptance for the on-line system, as evidence by several subject's disappointment when they learned that they would not be using the on-line system in the second exercise.

Metric	Data		
M1: Guidance tech.	Prefer on-line: 8	Prefer off-line: 10	Don't care: 2
M2: Data coll. tech.	Prefer on-line: 7	Prefer off-line: 3	Don't care: 10
M3: Restrictiveness	Not restrictive: 17	Too restrictive: 3	

Table 5.5: Subjective data concerning acceptance (hypothesis 1)

The acceptance of the on-line system was heavily influenced by the subject's preferred work style. Two different work styles were articulated by the subjects with extremely high consistency, as follows:

- The subjects who preferred on-line over off-line process information (i.e., accepted on-line guidance) indicated that they liked seeing precisely that piece of the instructions that was relevant to a single step (a focused view of the entire set of instructions). They also enjoyed being rid of a pile of paper through which they had to search frequently to find needed information.
- The subjects who preferred off-line over on-line process information (i.e., did not accept on-line guidance) indicated that they liked seeing all the necessary information at once. In other words, they preferred an overview, and didn't like the filtered view of information that the on-line system offered to them. Some also mentioned that they needed to write on the instruction sheets to work effectively, which was not possible when using on-line guidance.

Overall, no subject complained of being treated as a "subroutine" or otherwise felt subjugated to the control of a computer. Still, acceptance of an on-line version of process information was somewhat equivocal. These results might be explained by an even distribution of the two work styles mentioned above in our sample.

Hypothesis 2: Comprehension of on-line guidance

Hypothesis 2 involves the subject's comprehension of off-line versus on-line instructions. The results are summarized in Table 5.6.

Metric	Data	
M4: On/off-line comprehensibility	No difference: 18	Some difference: 2
M5: Textual version	Prefer text: 20	Prefer graphics: 0
M6: Sufficiency of instructions	Sufficient: 18	Insufficient: 2
M7: Obs. opinion of comprehension:	Good: 16	Fair: 4

Table 5.6: Subjective data concerning comprehension (hypothesis 2)

The vast majority of subjects felt that the comprehensibility of the instructions was neither improved nor worsened by offering them on paper or with an on-line system. Therefore the

hypothesis that on-line instructions are more comprehensible is not confirmed. The guidance provided off-line and on-line was highly similar, in that similar blocks of text were offered either on-line or off-line. A possible explanation is that the delivery mechanism was neutral.

All subjects felt that the textual version of the instructions was required, and none felt that a graphical version of the information would have been better. However, some mentioned that a graphical overview of the entire set of steps might have been helpful. Most felt that the instructions and other instruments offered sufficient information to accomplish the exercises. Those who felt that they had insufficient information mainly complained about needing more help in developing equivalence classes in the functional testing exercise. Finally, most subjects responded very well when interrupted and queried about the step that they were currently performing.

Hypothesis 3: Use of quantitative quality models

Hypothesis 3 involves a subject's adjustment of his or her behavior in response to quantitative quality models, a fundamental point in measurement-based feedback. The results are summarized in Table 5.7.

Metric	Data
M8: Usefulness of quant. models	Helpful: 9 Harmful: 4 Both:7
M9: Adjusted behavior	Those who found them helpful

Table 5.7: Subjective data concerning use of quantitative data (hypothesis 3)

Subjects exhibited dramatically different reactions to the use of quantitative quality models such as target values for total time. Nearly all subjects believed the quality models, took them seriously, and attached some significant value to them. Most viewed them as goals to be attained, and several subjects reported stopping work after having reached the goal. Some subjects stated that these quality models helped, others stated that they were harmful and distracting, while others felt that the models were at times both helpful and harmful. The difference seemed to lie in the subject's own performance, and had a strong influence on whether the subject adjusted his or her behavior:

- If a subject's own performance matched the quantitative quality model fairly well, the subject believed that the model confirmed his or her own work, felt favorably towards the model, and viewed the information as a goal that was attained. After attaining the goal, the subjects were highly likely to end that step and move on.
- If a subject's own performance deviated from the quantitative quality models sharply, the subject was likely to report feeling pressured and unpleasantly influenced. They did not abandon the exercise and move on simply because they had exceeded the estimated time or generated more than the estimated number of test cases.

It was possible for both situations described above to arise during a single exercise. The conclusion is that the quantitative quality models had a significant influence, but it was not always positive.

Source		Time in min.	Test cases	Number of failures
[KL95b], $n = 6$	Mean value	77.3	19.8	4.7
	Std. deviation	20.6	6.1	1.2
Current work, $n = 20$	Average values	65.9	19.3	3.4
	Std. deviation	30.3	13.5	1.4

Table 5.8: Comparison of data from two experiments on exercise "FT/cmdline"

Usefulness of quantitative quality models. The hypothesis concerning quantitative quality models stated that subjects adjust their behavior. Because all subjects in the experiment received this information, it was not possible to analyze whether subjects with guidance did better than subjects without guidance. For example, if people with guidance are more effective on the whole, that would suggest that the investment in collecting data, building models, and providing guidance was justified. Previous work offers data that can be used as a rough basis for comparison in order to investigate this question [KL95b]. That data includes observations of subjects who applied the functional testing technique to the program "cmdline." Due to different procedures, no data are available from that experiment that are comparable with the exercise that used structural testing and program "tokens."

To compare the two sets of results, average values are computed for some aspects for which the subjects in the current experiment received quantitative guidance. These are the total time (supplied value: 60 minutes), the number of test cases (supplied value: 20), and the number of failures (supplied value: 8). If the guidance was effective, the subjects who received the guidance should have come closer to the stated empirical guidelines (average values are close, standard deviations are low) than the subjects who didn't receive the guidance. However, a significant problem in comparing these data is that the current work included no training exercise, in contrast to the previous experiment. Further, not very many data points are available from the previous experiment. The data are summarized in Table 5.8.

Based on these data, it appears that the guidance was partially effective. Subjects required on average less time, but the standard deviation on time was higher. Their performance on test cases was similar in the two experiments, but their performance at detecting failures appeared to be worse when they received guidance. However, these differences are probably misleading. Subjects in the previous work were trained in using the test technique, which presumably helped them apply the test technique more effectively.

Hypothesis 4: Timeliness of supplying data

Hypothesis 4 involves the timeliness of a subject's reports on the status of his or her tasks. Most subjects reported changes in the process state without delay. Only in a very few cases did a subject delay significantly. A possible explanation is that they took the directions about reporting data very seriously. The subject's explicit knowledge that they were taking part in an experiment was probably an influence.

Metric	Data
M10: Delay before reporting	None: 12 Short: 7 Long: 1

Table 5.9: Data concerning reporting data in a timely way (hypothesis 4)

Aspect	n	Average	Std. dev.
M13: Total time FT (min.)	20	65.85	30.32
M13: Total time ST (min.)	20	97.55	32.84
M14: Percent failures FT	20	34.00	14.29
M14: Percent failures ST	20	29.00	16.51
Rate FT (%/hr)	20	33.98	15.39
Rate ST (%/hr)	20	19.81	12.29

Table 5.10: Descriptive statistics for time and failure data (hypothesis 5)

Hypothesis 5: Rate of work

Hypothesis 5 states that subjects are more effective when they receive on-line guidance as compared to off-line guidance. Data concerning the subject's motivation (metric M11), the observer's assessment of process conformance (metric M12), total time (metric M13), and percentage of possible failures detected (metric M14) were used to evaluate this hypothesis. The data for metrics M13 and M14 can be combined to obtain a rate with the unit 'percentage of possible failures per hour.' The normalization that results in this somewhat odd unit is necessary because the programs differed in size but had very similar fault densities, making it unrealistic to compare raw data on failures detected per hour. Other than this normalization, the data are not transformed in any other way to perform the following analyses.

Table 5.10 shows some descriptive statistics for time, failures, and rate of detecting failures. These data suggest that the quantitative quality models that were supplied for total time were 10–20 minutes too low. Figure 5.4 plots the raw data for the total time required by each subject, separated by test technique and guidance technique. Figure 5.5 plots the raw data for the percentage of total failures detected, also separated by the test and guidance technique.

All subjects reported that they were well motivated for the exercises, which is consistent with a mostly self-selected sample. The observer detected no significant deviations from the prescribed processes, so process conformance was judged to be good. Next the results are reported in terms of the three independent variables exercise, guidance technique, and order.

Independent variable 1: Exercise. Figure 5.6 plots the difference in times between the functional and structural testing exercises. At a first glance we see that all but three subjects needed more time for the structural testing exercise. Figure 5.7 plots the differences in percentage of failures detected. Here, the data suggest weakly that more failures were detected in the functional testing exercise. Finally, Figure 5.8 plots the differences in the rate at which failures were detected. Failures were clearly detected more rapidly in the functional testing exercise.

Table 5.11 summarizes the results of the statistical analysis. The data indicate that the

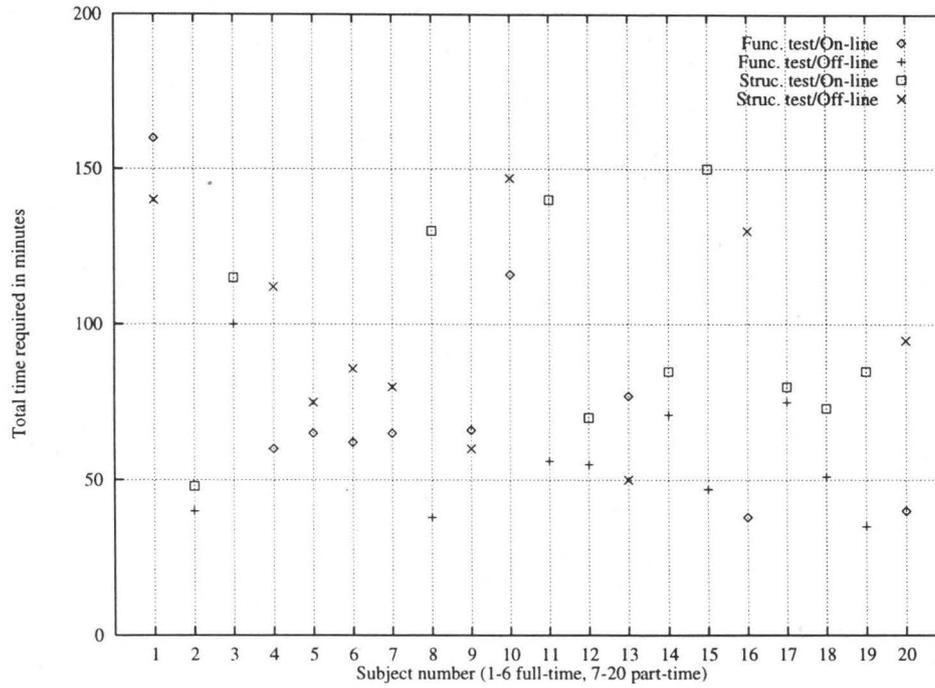


Figure 5.4: Raw data for total time required

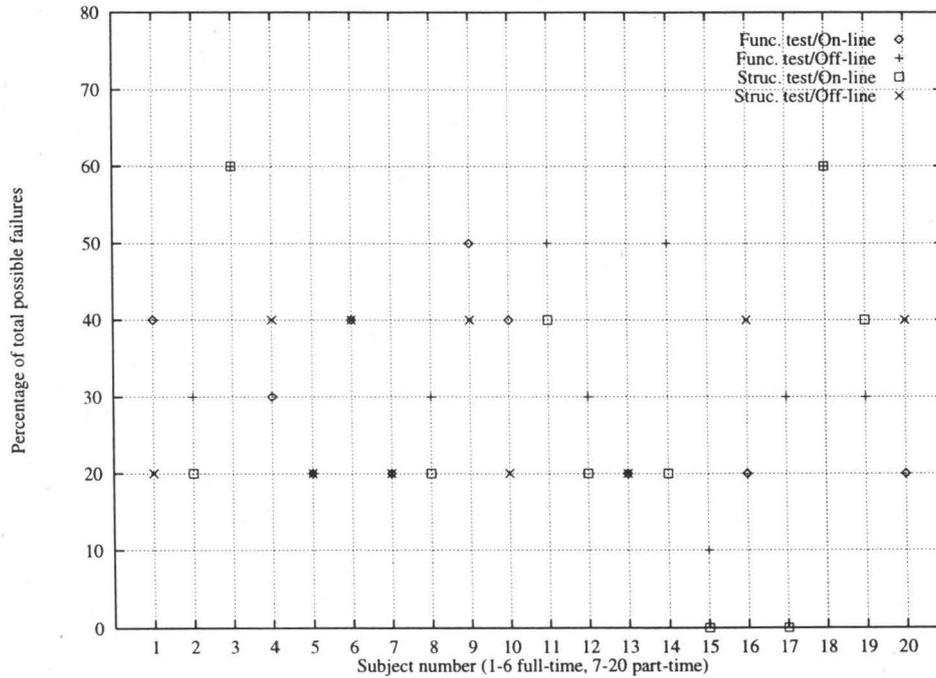


Figure 5.5: Raw data for percentage of failures detected

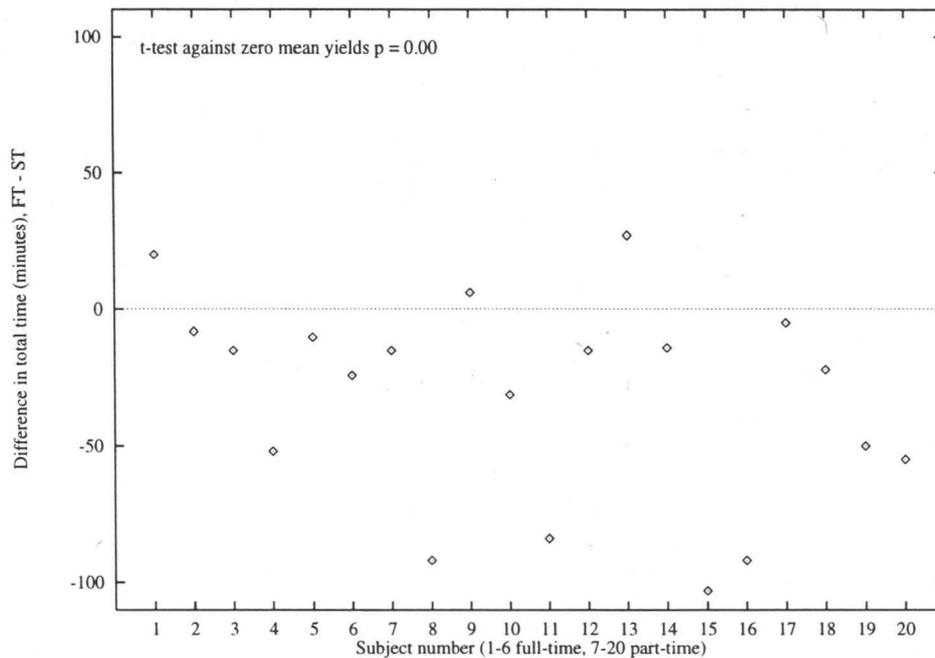


Figure 5.6: Difference in times, FT – ST

subject's performance on the two exercises differed significantly in the amount of time and the rate. However, they did not differ significantly in the percentage of possible failures detected. The most likely explanation for the differences is the different degree of exercise difficulty, as determined by the program size and complexity.

Dependent variable	n	Mean	Std. dev.	t test	<i>p</i>
Difference in time	20	-31.70	37.68	-3.76	0.00
Difference in % failures	20	5.00	13.96	1.60	0.13
Difference in rate	20	14.17	11.82	5.36	0.00

Table 5.11: Results for effect of exercise (functional – structural)

Independent variable 2: Guidance technique. Figure 5.9 plots the difference in times between the exercises performed with off-line and on-line guidance. At first glance we see that most of the subjects needed more time when using the on-line guidance technique as compared to the off-line technique. Figure 5.10 plots the differences in percentage of failures detected. No strong differences are suggested. Finally, Figure 5.11 plots the differences in the rate at which failures were detected. The figure suggests a significant difference in the rate. Some subjects were slightly faster when using on-line guidance, but others were much faster when using off-line guidance. It appears that the more experienced people worked faster with on-line guidance, and the less experienced subjects worked faster with off-line guidance.

Table 5.12 summarizes the results of the statistical analysis. The differences in time and percentage of failures are not judged to be significant. However, the difference in rate is judged to be significant. Subjects were on average more efficient when they used off-line guidance.

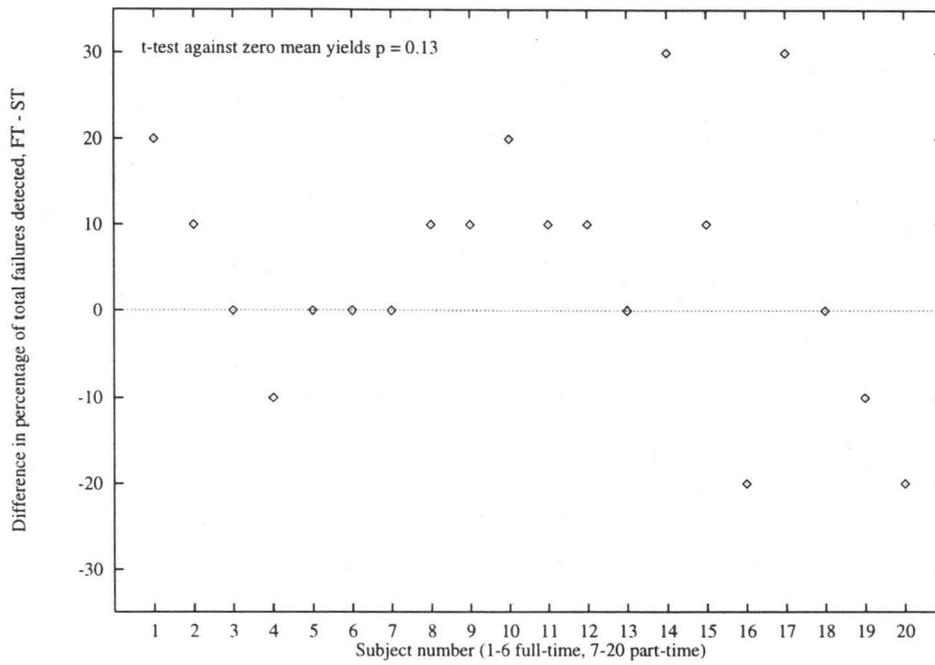


Figure 5.7: Difference in percentage of failures, FT – ST

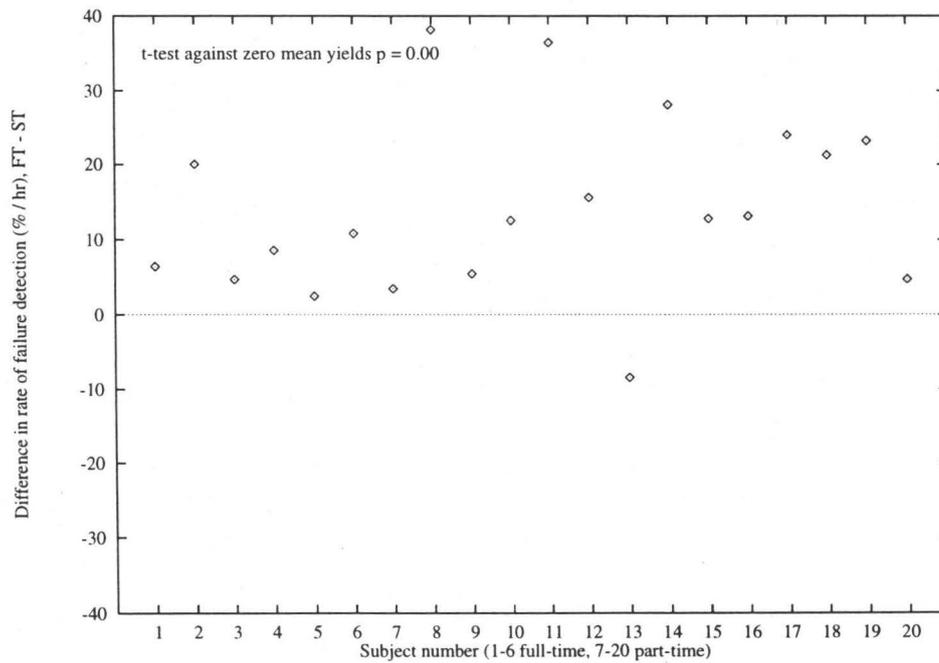


Figure 5.8: Difference in rate of failure detection, FT – ST

Dependent variable	n	Mean	Std. dev.	t test	p
Difference in time	20	-9.10	48.89	-0.83	0.42
Difference in % failures	20	5.00	13.96	1.60	0.13
Difference in rate	20	8.25	16.71	2.21	0.04

Table 5.12: Results for effect of guidance technique (off-line – on-line)

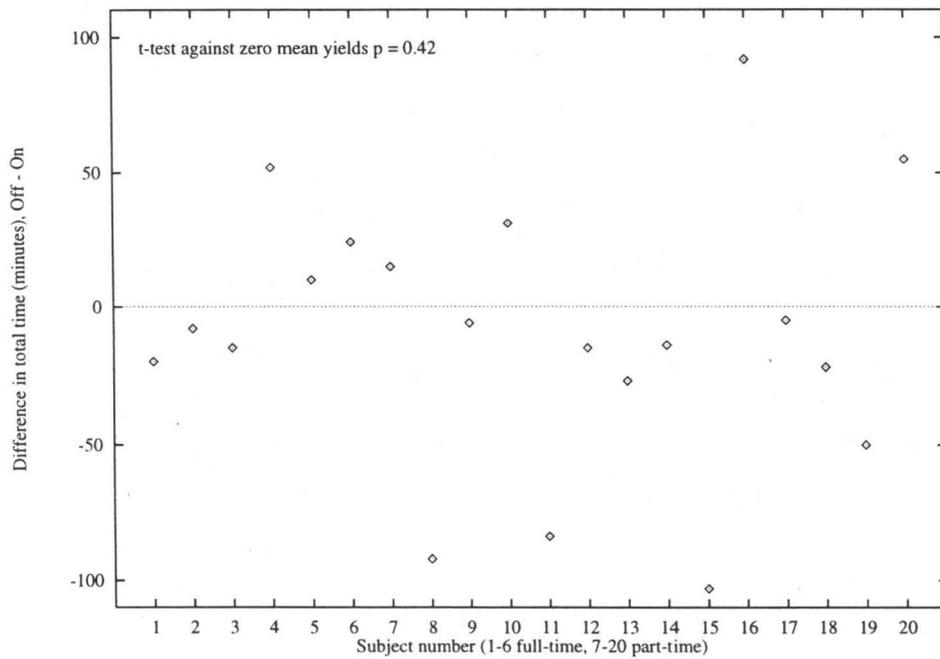


Figure 5.9: Difference in times, off – on

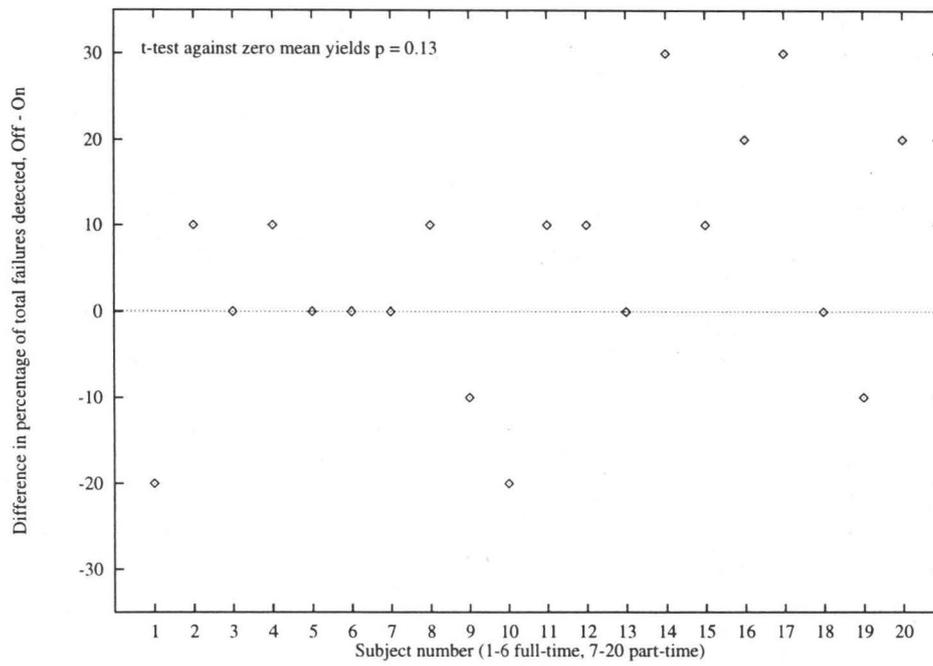


Figure 5.10: Difference in percentage of failures, off – on

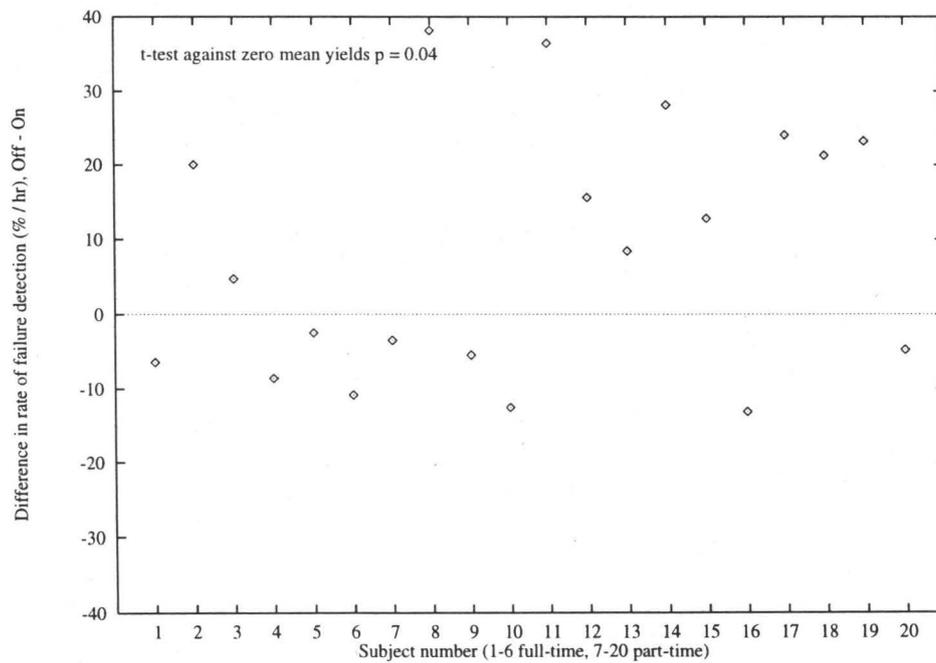


Figure 5.11: Difference in rate of failure detection, off – on

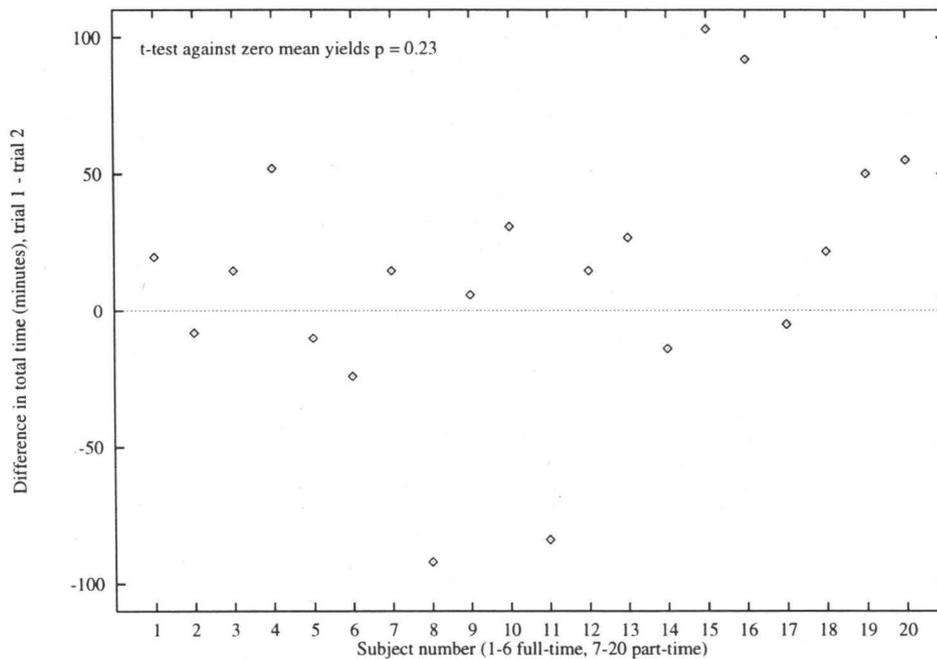


Figure 5.12: Difference in times, trial 1 – trial 2

Independent variable 3: Order. Figure 5.12 plots the difference in times between trial 1 and trial 2. It appears that subjects spent more time on their first exercises. Figure 5.13 plots the differences in percentage of failures detected. A tendency towards a lower percentage in the second trial is suggested. Finally, Figure 5.14 plots the differences in the rate at which failures were detected. No strong difference is suggested.

Table 5.13 summarizes the results of the statistical analysis. A systematic change in the data such as second trials that are invariably shorter than first trials would indicate a maturation effect. The data do not indicate a maturation effect in the amount of time required, but suggest that subjects detected fewer failures during the second exercise. A possible explanation for the difference in the failure data is a fatigue effect.

Dependent variable	n	Mean	Std. dev.	t test	<i>p</i>
Difference in time	20	13.30	47.87	1.24	0.23
Difference in % failures	20	7.00	13.02	2.40	0.03
Difference in rate	20	2.17	18.60	0.52	0.61

Table 5.13: Results for effect of order (trial 1 – trial 2)

5.6.2 Discussion

The debriefing sessions revealed that differences in people's acceptance of on-line process guidance may have depended heavily on their understanding of the goals of the exercises. For example, the goal of the functional testing exercise was to test the program thoroughly against the specification. We assume that inexperienced people lacked an understanding of the over-

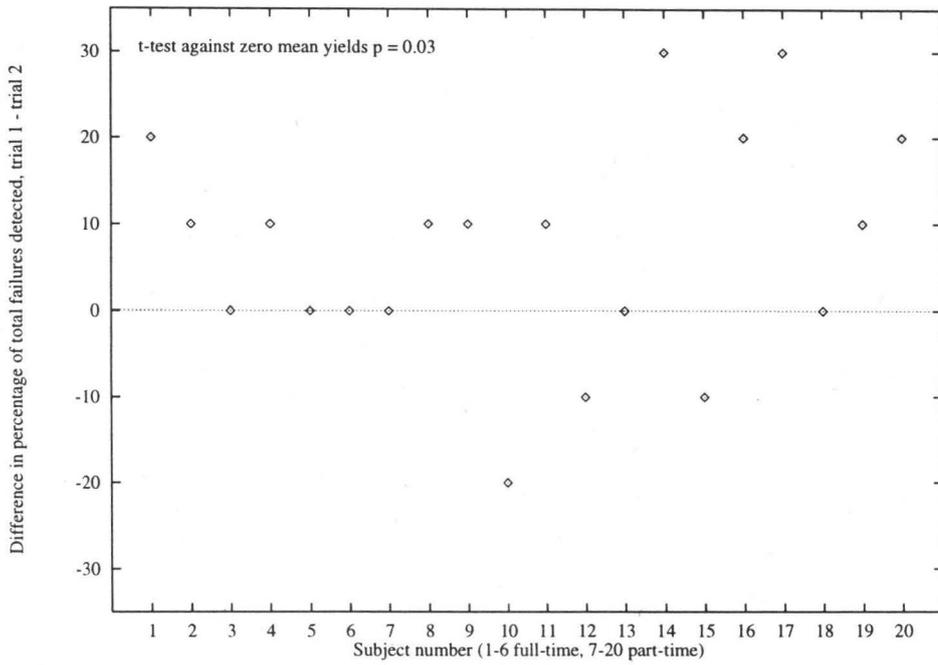


Figure 5.13: Difference in percentage of failures, trial 1 – trial 2

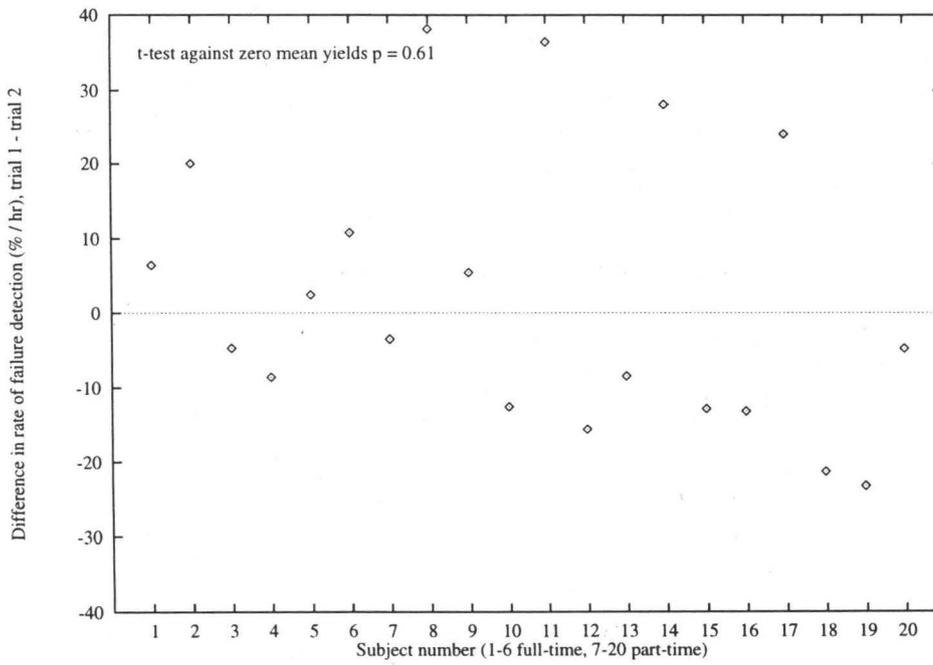


Figure 5.14: Difference in rate of failure detection, trial 1 – trial 2

all goals behind functional and structural testing. Therefore, they preferred a comprehensive overview that helped them understand the goal. In contrast, the experienced people probably understood the goals, and therefore preferred receiving exactly what they needed at each step. These results suggest providing inexperienced people with books or other comprehensive overviews, and providing experienced people with filtered, directed views of needed process information.

5.7 Critique and Future Work

This section critiques the experiment, suggests improvements to MVP-S, and identifies future research directions.

5.7.1 Improving the experiment

The experiment involved many compromises between internal and external validity issues. These compromises and some opportunities for improvement are discussed next.

Task size. The most interesting possibility for improvement is the length of the exercise, because this has a dramatic effect on external validity. Very short exercises were used to make it possible to recruit volunteers. The use of longer exercises could expose new variation factors.

Task difficulty. The instruments were chosen to attain two tasks of approximately one hour in duration. However, there is some indication from the data that the structural testing task was much more difficult. Future work might use tasks that are more closely matched in difficulty and duration.

Cross objects and test technique. If code objects could be used that are highly similar in size, then the test technique could be crossed with the objects (apply both techniques to both objects). This would eliminate the confounding of the two variables seen in this experiment, and determine which of the object and test technique had a larger influence on the results.

Nature of task. Testing exercises were used because of their availability. The use of design or coding exercises could expose new variation factors, and use other on-line support such as browsing tools. It would also be interesting to investigate the variation factors that affect a purely off-line task such as conducting a design inspection.

Training. Some training could have been offered for the guidance and test techniques. It may be worthwhile to use subjects who have been trained in the use of the on-line process guidance system under study. This would, however, prevent hiding the goal of evaluating that system, and might not attain the goal of natural use. Further, it would be useful to train subjects in the application of the test techniques so that they would understand better the task that is demanded of them. The concern of subjects ignoring instructions about a familiar task was probably overestimated.

All or nothing on-line. All subjects, even the ones who used on-line guidance, received some of the necessary materials in paper form. A future experiment might present all specification, code, and other necessary documents in an on-line, searchable form for people who use an on-line process guidance system.

Measurement of motivation and experience. The question sheets asked subjects about their motivation for the exercise and their experience with the test technique. The answers to these questions were not helpful. Motivation is very difficult to measure. Experience might be better measured with a pretest.

Fatigue. Subjects were encouraged to do both exercises in the course of a single morning or afternoon. The fatigue effect identified in the results and the comments from the subjects together suggest that the exercises should have been spread over two days.

Debriefing question sheets. The questions on the debriefing sheets asked for binary answers such as like/dislike of guidance on-line, like/dislike of question sheets on-line, etc. A five-point scale (e.g., “dislike strongly,” “dislike,” “no opinion,” “like,” “like strongly”) would have allowed a subject to express his or her opinions more precisely.

Additional debriefing sessions. The debriefing sessions were conducted shortly after a subject completed the experiment and before any data analyses had been done. It would be useful to conduct additional follow-up sessions following completion of the data analyses to explore in more detail the questions raised by those analyses.

5.7.2 Possible improvements to MVP-S

No subject reported any real difficulty with using MVP-S in the experiment. However, nearly all subjects complained that the interface was awkward to use, and many offered ideas about possible improvements. This experience suggested that we should implement the following changes to the interface.

Short-cuts. Use immediately accessible buttons or other convenient interface facility for frequently accessed features “start process” and “complete process.” These features are currently buried behind a menu and pop-up window.

Overview needed. The system might offer a graphic overview of all steps, possibly a view of just product flow. This would help a user visualize his or her place in the overall process as well as track the status of his or her processes, possibly using colors or other non-textual information.

Too many windows. Subjects reported being irritated with being forced to click “OK” on a large number of windows in order to make them disappear. We need to find a different way to transmit status and other confirmation messages to the users.

Print feature. The system could offer a "print" feature so that users who prefer a plain-paper version of the instructions could gain an overview easily, and could write on the newly printed instructions.

5.7.3 Future work

The following questions were raised by the results of the experiment or by subjects during the debriefing sessions. These questions suggest some future directions for evaluating measurement-based guidance.

Unfamiliar tasks. How do people react when confronted with unfamiliar tasks? We observed very different reactions. Some slowed down while they attempted to learn how to accomplish the task properly. Others raced through without much thought. It is not clear whether this behavior is common to students or most people.

Subjects overlook failures. Can testers be trained to search more diligently for failures? Subjects overlooked a significant number of visible failures in the output of their own test cases. This is consistent with previous work [KL95b]. It appears that testers could improve their performance simply by investing more time in analyzing their test results for failures.

Context switching. Can the cost of interruptions be reduced by keeping developers better informed about the current status of their activities? Many subjects stated that they liked seeing their tasks listed in one place, with the status of that activity clearly marked, and that they found it easy to pick up where they had left off when interrupted.

Activities not involving the computer. Is automated support helpful for off-line constructive activities? For example, what are the consequences of putting a definition of an off-line process such as code reading on the computer? The exercises in the experiment involved using the computer, so an on-line process-centered system could be easily incorporated.

Iteration. Can intelligent support be provided for iteration? Learning is a natural part of any process. After completing a task, personnel frequently realize how they should have done the task, and in some cases will need to iterate through that task at least once more. We would like to support this without requiring replanning.

Experience. The results suggest marked differences in the performance and preferences of inexperienced versus experienced subjects. Future experiments should attempt to recruit highly experienced people to investigate further the benefits suggested by this experiment for experienced people.

5.8 Summary

This chapter discussed an empirical study that compared off-line and on-line measurement-based guidance using subjects who worked alone on small exercises. A number of hypotheses concerning the acceptance and effectiveness of using a process-centered system that supports measurement-based guidance and feedback were stated and evaluated in the course of the study. In this controlled experiment, subjects were equally effective in time and percentage of failures found when using the off-line and on-line guidance techniques. However, acceptance of on-line guidance and quantitative quality models differed in accordance with subject's experience levels.

We draw the conclusion that measurement-based process feedback can realistically be used with automated support. However, much work remains to develop a highly usable interface and to overcome problems that cause people to reject the use of an on-line guidance technique. More work is needed to establish the validity of these results concerning individuals and process-centered environments when different processes and environments are used. Still more work is needed to test the assumptions behind improving communication among teams by using process-centered environments.

The experiment contributed a deeper understanding of several variation factors, specifically a subject's experience and performance. The acceptance of the on-line system depended on each subject's personal work style (a variation factor identified by the experiment), and a subject's use of quantitative quality models seemed to depend on their performance (a second variation factor).

The author was solely responsible for designing and conducting the experiment. However, a great debt is acknowledged to the subjects who participated in the experiment as well as the experts who offered helpful criticism of the design, procedures, and interpretations.

Chapter 6

Conclusion

The dissertation contributed a framework for integrating the goal-oriented measurement technology articulated by the GQM Paradigm with the process modeling technology as supported by the MVP-L process modeling language. The work also provided an existence proof of a system that provides measurement-based guidance and feedback, and stated a series of requirements and assumptions about the interactions between users and process-centered software engineering environments. Finally, the empirical study contributed towards the software engineering community's understanding of the benefits and limitations of using a process-centered software-engineering environment to inform users about software development tasks. Future work might address the following questions:

Interruptions. Can the cost of interruptions be reduced using a process-centered system? We believe that the use of an on-line system that records the current status of a process could help to reduce the burden of context switching and interrupts. The system keeps a highly visible record of a person's progress, and reduces confusion that can arise when too many pieces of paper are lying around.

Type of work. Is automated support helpful for off-line work? The exercises used in this work required a significant amount of work on the computer. Some subjects mentioned that although they found the on-line guidance technique useful for on-line work, they would not want to use such a system if they were doing purely off-line work.

Iteration. How can iteration be supported intelligently? The exercises used in the experiment directed the subjects through a single sequence of testing steps. As part of their debriefing, nearly all subjects indicated that they believed a second iteration through the testing steps would have detected more failures. Further iterations would have allowed the subjects to apply their newly gained knowledge about the testing technique and the program. Intelligent support for previously unplanned iteration through processes would support natural human learning processes and make it possible to collect data about these (often implicit) processes.

Appendix A

Appendix: Instruments

This appendix presents the instruments used in the experiment. These instruments can be used verbatim to repeat the experiment; a electronic copy is available upon request. These instruments include the hard-copy directions, pictures of the process-centered system that the subjects used, explanations of the test techniques, off-line and on-line question sheets, program specifications, and source code.

All instruments were developed for native German speakers. This appendix offers English versions of all documents. All translations are as true to the original as possible. All documents *with the exception of the programs and their outputs* have been translated. The source code is presented without changes, which may make it difficult for people who do not know German to isolate typographical (cosmetic) faults.

A.1 Script used to introduce subjects to MVP-S

1. General

- (a) Two test exercises, for a total of about 3 hours.
- (b) The goal is to find as many failures as possible in as little time as possible.
- (c) Writing on all sheets is naturally OK.
- (d) Time entries should reflect the total time needed.
- (e) Your ID is ...
- (f) Please read the instructions and perform the steps.

More about the on-line tools

2. Menu bar:

- (a) One button per tool
- (b) Quit when finished

3. MVP-S:

- (a) Explain main window
 - i. Column 1: Instance name of a process
 - ii. Column 2: Model name of a process
 - iii. Column 3: Status of a process
 - iv. Menu item "Action/Send Event" to inform the machine of a status change.
 - v. send "start" for the preparation step.
 - vi. Menu item "Info/Context" to get the instructions for each process
- (b) Explain context window
 - i. Comments constitute the directions for each process
 - ii. Entry and exit criteria are also displayed in the window.

4. Question sheet:

- (a) Help for each question appears at the bottom.
- (b) Bug: data are not read in again.
- (c) Data are retrieved later.

A.2 Instructions for “functional testing” as used off-line

In this exercise you will test a program named “cmdline.” Your goal is to find as many failures as possible in as little time as possible. You will generate and execute test cases with the help of the specification. Then you will search for failures. Because isolating faults is not part of this exercise, you will not receive the source code for the program. **Guideline:** In the past, similar exercises required on average circa **60** minutes.

For this exercise you will need the following materials in addition to these instructions: the question sheet “Functional testing,” the instruction sheet “Building equivalence classes,” and the specification of the program “cmdline.” At the end you will receive a sample solution.

Step 1: Preparation

- Read over the question sheet “Functional testing.” Enter your identifier and answer the first two questions.
- If you are unfamiliar with equivalence classes, read through the instructions for developing and using equivalence classes.
- Log on to a computer. Create a new directory for your work, and change into the new directory with the “cd” command. Fetch the necessary files by entering the following command:

```
tar xf ~lott/Expt/ft-cmdline.tar
```

Thereafter the following files must be present:

```
Makefile      cmdline      run-suite     test-dir
```

Step 2: Generate test cases

- You received the specification for the program with your materials. Read through the specification carefully. Use it to derive the equivalence classes. **Guideline:** In the past, on average **16** valid and **4** invalid equivalence classes were developed for similar components.
- Generate test cases by choosing boundary values from the equivalence classes. A test case consists of an input and an expected output. **Guideline:** In the past, on average **20** test cases were generated for similar components.
- When you are finished, enter in the question sheet the number of valid and invalid equivalence classes, the number of test cases, and the amount of time you needed. If you also typed in your test cases while developing them, please try to separate your entries for total time between the generation and the typing-in steps.

Step 3: Type in the test cases

- In this exercise, a test driver is used to apply the test cases to the program. The test driver reads parameter files and invokes the program with the parameters specified in those files. Example: if the program “cmdline” is supposed to be invoked with “cmdline 1 2 3”, then the expression “1 2 3” should be placed in the parameter file.

You must create the parameter files. A test case = a parameter file. The expressions in a parameter file specify one invocation of the program for a one run of the test driver. Parameter files are named with “.test” as the file suffix.

The results are written into a file. A complete test of the component is accomplished in this way.

- Type in your test cases in files under the directory “test-dir”. Follow the specification of the test driver while doing so. When you are finished, enter the amount of time you needed in the question sheet.

Step 4: Run test cases

- Apply the test cases to the component by typing in the command “run-suite”. If you made some typing mistakes, please correct the mistakes now, type “make clean”, and repeat the tests by entering the command “run-suite” again. The results are in individual files under the directory “test-dir” and are also summarized in a file “test-results.summary”.
- When you are finished with this step, enter the amount of time you needed in the question sheet. Print out the test results from the file “test-results.summary”.

Step 5: Search for failures

- Look over the results carefully. Find possible failures by comparing the expected results according to the specification with the output of your test cases. Mark the detected failures in both copies of the output with circles, etc. **Guideline:** In the past, on average 8 failures were detected in similar components.
- When you believe that you have detected all failures, please enter the time you required in the question sheet. Complete the rest of the question sheet and hand it in with your test results.

A.3 Instructions for “functional testing” as used on-line

For this exercise you will need the following materials in addition to these instructions: the instruction sheet “Building equivalence classes,” and the specification of the program “cmdline.” At the end you will receive a sample solution.

1. Log on to a computer. Start the menu bar for the information system by entering the following command:

```
~lott/Expt/menu-ft
```

2. Use the menu bar to start one instance of the information system “MVP-S.” Select the project “Funktionales_Testen,” wherein you take on the role “Test_Ingenieur.”
3. Use the menu bar again to start one instance of the question sheet tool.
4. Please use the information system for the rest of the information about the exercise.

Figures A.1, A.2 and A.3 show the initial menu bar, the MVP-S work context window and the process context windows, respectively. These interfaces support the subjects who use on-line directions to perform functional testing.



Figure A.1: Menu bar to start MVP-S and a question sheet tool

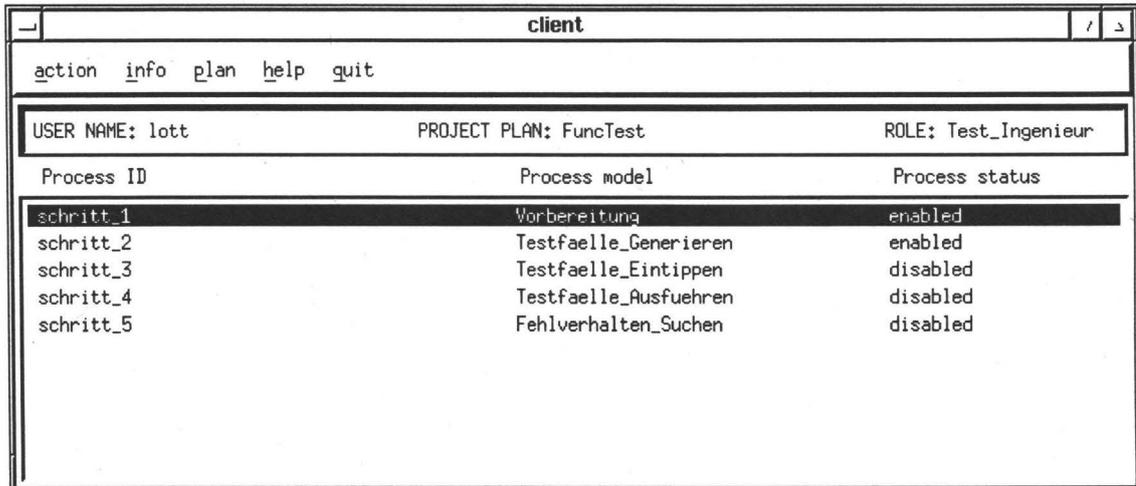


Figure A.2: MVP-S work-context window that supports functional testers

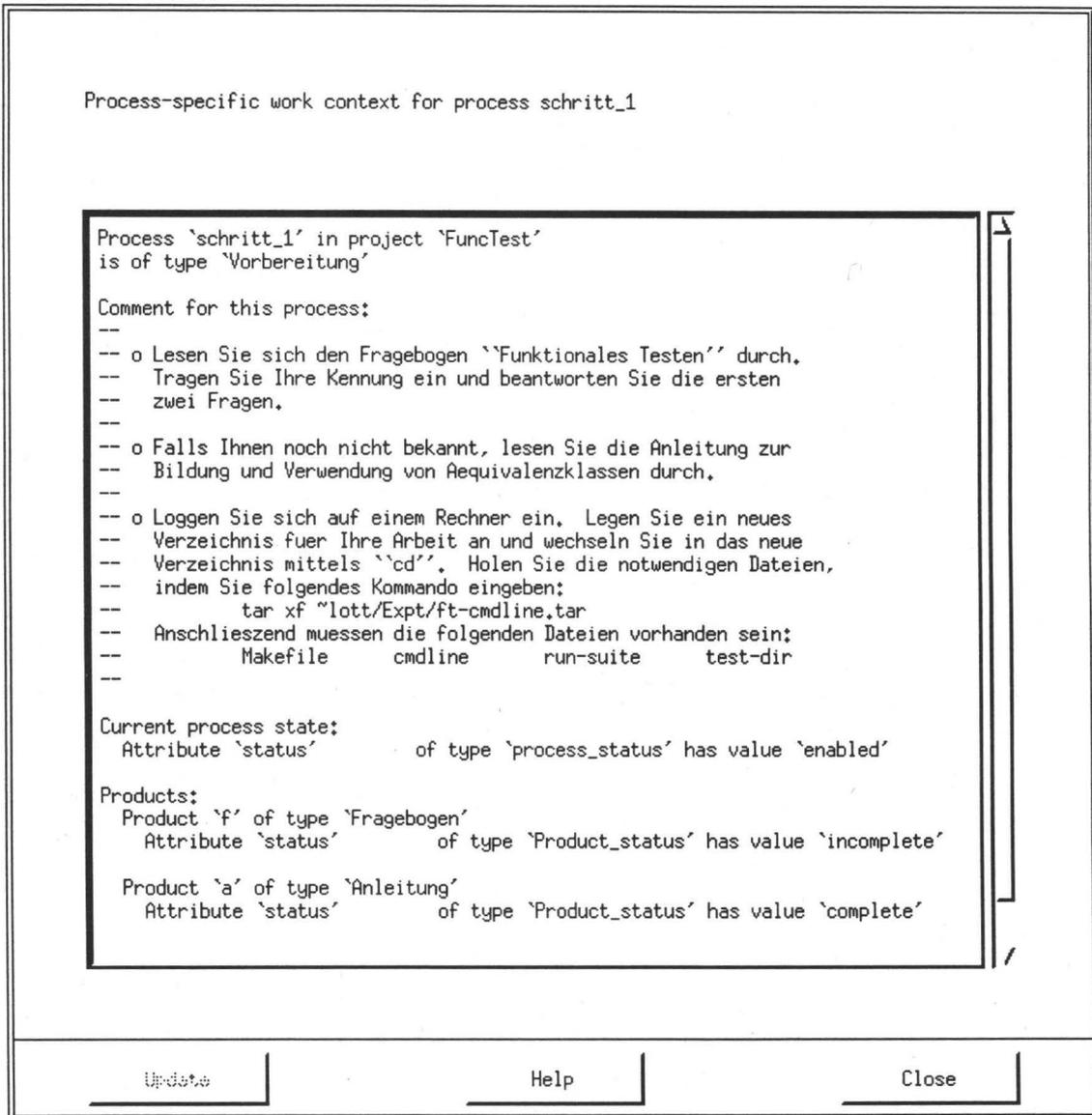


Figure A.3: MVP-S process-context window for functional testing step "schritt_1"

A.4 Question sheet “functional testing” as used off-line

Your ID: _____ Date: _____

Note: For all time-related entries, please deduct time for pauses, etc.

1. How motivated are you for the experiment? [Please mark the scale.]

Estimate						
Value	0	1	2	3	4	5
Comparison	not at all	little	so-so	pretty much	quite a bit	very much

2. When did you start the experiment?

_____ : _____ [Hour:Minute]

3. How many equivalence classes did you derive?

Valid: _____ [Count], Invalid: _____ [Count]

4. How many test cases did you generate?

_____ [Count]

5. How much time did you need to construct the test cases (step 2)?

_____ [Minutes]

6. How much time did you need to type in the test cases (step 3)?

_____ [Minutes]

7. How long did the execution of test cases (step 4) require, if more than zero?

_____ [Minutes]

8. How long did you search for failures (step 5)?

_____ [Minutes]

9. How many different failures did you find?

_____ [Count]

10. When did you end the experiment?

_____ : _____ [Hour:Minute]

11. Would you be inclined to perform another iteration in order to find more failures?

_____ [yes, no]

12. How well do you feel you've mastered functional testing? [Please mark the scale]

Estimate						
Value	0	1	2	3	4	5
Comparison	not at all	little	so-so	pretty much	quite a bit	very much

A.5 Question sheet "functional testing" as used on-line

Figure A.4 shows the on-line question sheet used by subjects who use the process-centered system to perform functional testing.

fb-ft	
C-g eingeben fuer das Menu, C-z fuer Hilfe. Speichern Beenden	
Fragebogen "Funktionales Testen"	
Ihre Kennung: ■ _____	Datum: <u>27. Jan 1996</u>
1. Wie motiviert sind Sie fuer das Experiment? (Schaetzung)	_____
2. Wann haben Sie mit dem Experiment angefangen?	_____
3. Wieviele Aequivalenzklassen haben Sie abgeleitet? Gueltige: _____ Ungueltige: _____	_____
4. Wieviele Testfaelle haben Sie generiert?	_____
5. Wie lange hat die Generierung von Testfaellen (Schr. 2) gedauert?	_____
6. Wie lange hat das Eintippen (Schr. 3) gedauert?	_____
7. Wie lange hat die Durchfuehrung (Schr. 4) gedauert?	_____
8. Wie lange hat die Suche nach Fehlverhalten (Schr. 5) gedauert?	_____
9. Wieviele verschiedene Fehlverhalten haben Sie gefunden?	_____
10. Wann haben Sie das Experiment beendet?	_____
11. Wuerden Sie eine Weitere Iteration anstreben?	_____
12. Wie gut beherrschen Sie funktionales Testen (Schaetzung)?	_____
Kennung: Bitte Ihre Kennung eintragen.	

Figure A.4: On-line question sheet used by functional testers

A.6 Building equivalence classes

Functional testing based on equivalence classes requires the input to be divided into equivalence classes from which similar behavior is expected. The following overview gives heuristics for such a division [Reference: G. Myers, *The Art of Software Testing*, 1979.]

1. If an input condition specifies one region, identify one valid and two invalid equivalence classes.

Example: For input condition "the number A to be entered varies from 1 to 999," define the valid equivalence class $\{1 \leq A \leq 999\}$, and the invalid equivalence classes $\{A < 1, 999 < A\}$.

2. If an input condition specifies a number of values, identify one valid and two invalid equivalence classes.

Example: For input condition "the number of search expressions is minimum 1 and maximum 6," define the valid equivalence class $\{1 \leq \text{number} \leq 6\}$ and the invalid equivalence classes $\{\text{number} = 0, \text{number} > 6\}$.

3. If an input condition specifies a set, and there are grounds for belief that the set elements are treated differently, determine one valid equivalence class for each element in the set, plus one additional invalid class for an element that is not contained in the set.

Example: For input condition "the input is of type gender," we assume different treatment of the gender values. Choose valid equivalence classes $\{\text{male}\}$, $\{\text{female}\}$, $\{\text{neuter}\}$ and the invalid one $\{\text{martian}\}$.

4. If an input condition specifies a "must" situation, identify one valid and one invalid equivalence class.

Example: For input condition "the first character in the word must be a letter," choose the valid equivalence class $\{w \mid w \text{ is a word and the first character is a letter}\}$ and the invalid equivalence class $\{w \mid w \text{ is a word and the first character is not a letter}\}$.

5. If there are grounds for belief that elements in an equivalence class are treated differently, divide the equivalence class appropriately into new ones. Example: $\{\text{even values}\}$, $\{\text{odd values}\}$.

Once the equivalence classes have been determined, test data must be selected. A procedure named "boundary value analysis" specifies which data are chosen from the equivalence classes. Instead of choosing just any element from an equivalence class, values are used that are near or on the boundaries of the equivalence classes. Instead of using only the input conditions, test cases are also determined by observing the result space. One can work according to the following schema:

1. Assign each equivalence class a unique number.
2. Until all valid equivalence classes are covered, write one test that covers as many as possible of the valid, not-yet-covered equivalence classes.
3. For each invalid equivalence class, write one test that covers this and only this one.

A.7 Specification of program “cmdline”

Name

cmdline – Syntactic and semantic analysis of a command line

Usage

cmdline –hilfe

cmdline –mass <measure> [Search-option] File [File ..]

Description

cmdline analyzes a command line of a measurement tool for syntactic and partially for semantic correctness. The units that are eventually supposed to be measured are stored in files. However, the measurement functionality is not part of this tool. The current version of **cmdline** only evaluates the command-line parameters.

The user must give the program at a minimum one measure and one file as argument. In addition to the arguments (measure, file, etc.), exactly one so-called search option may be given. The order of the measurement options and search options is not important. When the command line is evaluated, all arguments beginning with the first non-option are treated as file names. The existence of such files is not checked.

If successful (i.e., if the syntax and meaning of the arguments is found to be legal), a summary of the arguments is printed out. Otherwise, an explanatory error message is printed. The summary consists of the measure to be computed, the search options if any were given (including a value for -unter and -ueber), and a list of the files to be read.

Options

Options and arguments for options can be abbreviated to their unique prefixes. In the following text, the unique prefix of each option appears before the square brackets. A maximum of one search option may be used.

- –h[ilfe]
Help option. A helpful text is printed and nothing else is done.
- –?
Help option. See above.
- –mas[s] <measure>
Valid measures are GKO[M], LKO[M], GKH[M], LKH[M], GI[HE], and LI[HE].
- –a[lle]
Search option. The program should evaluate the measure for all units.
- –max
Search option. The program should determine the maximum value of the measure for all units.

- `-mi[n]`
Search option. The program should determine the minimum value of the measure for all units.
- `-du[rchschnitt]`
Search option. The program should determine the average value of the measure for all units.
- `-un[ter] <value >`
Search option. The program should identify all units for which the measure lies under the boundary value. The `<value>` may be any real number.
- `-ue[ber] <value >`
Search option. The program should identify all units for which the measure lies over the boundary value. The `<value>` may be any real number.

Example

```
% cmdline -mass GKHM -alle datei1
Die Aufgabe ist:
Mass: GKHM
Suche: -alle
Anzahl der zu lesenden Dateien: 1
Die Dateien sind:
    datei1
```

Authors

Baumgärtner, Claßen, Gieseke, Lott.

A.8 Source code of program "cmdline"

cmdline.h:

```
#ifndef CMDLINEDOTH
#define CMDLINEDOTH

#define HILFE 0
#define MASS 1
#define GKOM 2
#define LKOM 3
#define LKHM 4
#define GKHM 5
#define LIHE 6
#define GIHE 7
#define ALLE 8
#define MAX 9
#define DURCH 10
#define MIN 11
#define UNTER 12
#define UEBER 13

struct keyword_entry {
    char *keyword;
    int min_len;
    int id;
};

struct aufgabe {
    double zahl;
    char *mass;
    char *such;
};

#endif
```

cmdline.c:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
extern double atof();
#include "cmdline.h"

static struct keyword_entry keyword_table[] = {
    {"-?", 2, HILFE},
    {"-hilfe", 2, HILFE},
    {"-mass", 4, MASS},
    {"GKOM", 3, GKOM},
    {"LKOM", 3, LKOM},
    {"LKHM", 3, LKHM},
    {"GKHM", 3, GKHM},
    {"LIHE", 2, LIHE},
    {"GIHE", 2, GIHE},
    {"-alle", 2, ALLE},
    {"-max", 4, MAX},
```

```

    {"-durchschnitt", 3, DURCH},
    {"-min",          3, MIN},
    {"-unter",        3, UNTER},
    {"-ueber",        3, UEBER},
};
static int keyword_table_size = sizeof(keyword_table) /
                                sizeof(struct keyword_entry) - 1;

void usage(progn)
    char *progn;
{
    fprintf(stderr, "Verwendung: %s -mass <MASS> [ Suchption ] Datei [ Datei ... ]\n",
              progn);
    exit(1);
}

static int is_str_keyword(arg, valid_arg, min_vergl_len)
    char *arg, *valid_arg;
    int min_vergl_len;
{
    int arglen, rc;

    rc = 0;
    arglen = strlen(arg);
    if (strncmp(arg, valid_arg, arglen) == 0)
        rc = 1;

    return rc;
}

static int code_string(str)
    char *str;
{
    int i, rc;

    rc = -1;
    for (i = 0; i < keyword_table_size; ++i) {
        if (is_str_keyword(str, keyword_table[i].keyword, keyword_table[i].min_len)) {
            rc = keyword_table[i].id;
            break;
        }
    }
    return rc;
}

void print_aufgabe(aufg)
    struct aufgabe *aufg;
{
    if ( (aufg->mass != NULL && strcmp(aufg->mass, "-hilfe") == 0)
        || (aufg->mass == NULL && aufg->such == NULL)) {
        usage("cmdline");
        return;
    }
}

```

```

printf("Die Aufgabe ist:\n");
if (aufg->mass)
    printf("Mass: %s\n", aufg->mass);
if (aufg->such) {
    printf("Suche: %s\n", aufg->such);
    if (strcmp(aufg->such, "-unter") == 0 || strcmp(aufg->such, "-ueber") == 0)
        printf("Zahl: %.1f\n", aufg->zahl);
}
}

```

```

void print_dateien(argi, argc, argv)
    int argi, argc;
    char **argv;
{
    int i;

    if (argc == argi)
        printf("Es fehlt eine Datei\n");
    else
    {
        printf("Anzahl der zu lesenden Dateien: %d\n", argc - argi);
        printf("Die Dateien sind:\n");
        for (i = argi; i < argc; ++i)
            printf("  %s\n", argv[i]);
    }
}

```

```

int process_switches(argc, argv, aufg)
    int argc;
    char **argv;
    struct aufgabe *aufg;
{
    int i, rc = 0, position = 0;

    aufg->zahl = 0.0;
    aufg->mass = NULL;
    aufg->such = NULL;

    for (i = 1; i < argc; ++i) {
        switch (code_string(argv[i])) {

            case HILFE:
                aufg->mass = "-hilfe";
                break;

            case MASS:
                if (aufg->mass != NULL) {
                    fprintf(stderr, "Es sind zuviele Masse angegeben\n");
                    rc = -1;
                }
                else {
                    if (i++ >= argc) {
                        fprintf(stderr, "Es ist kein Mass angegeben\n");
                        rc = -1;
                    }
                }
        }
    }
}

```

```

else {
    switch (code_string(argv[i])) {
    case GKOM:
        aufg->mass = "GKOM";
        break;
    case LKOM:
        aufg->mass = "LKOM";
        break;
    case LKHM:
        aufg->mass = "LKOM";
        break;
    case GKHM:
        aufg->mass = "GKHM";
        break;
    case LIHE:
        aufg->mass = "LIHE";
        break;
    case GIHE:
        aufg->mass = "GIHE";
        break;
    default:
        fprintf(stderr, "Es ist kein gueltiges Mass angegeben\n");
        rc = -1;
        break;
    }
}
break;

case ALLE:
    if (aufg->such != NULL) {
        fprintf(stderr, "Es sind zuviele Suchoptionen angegeben\n");
        rc = -1;
    }
    else
        aufg->such = "-alle";
    break;

case MAX:
    if (aufg->such != NULL) {
        fprintf(stderr, "Es sind zuviele Suchoptionen angegeben\n");
        rc = -1;
    }
    else
        aufg->such = "-max";
    break;

case DURCH:
    if (aufg->such != NULL) {
        fprintf(stderr, "Es sind zuviele Suchoptionen angegeben\n");
        rc = -1;
    }
    else
        aufg->such = "-durchschnitt";
    break;

```

```

case MIN:
    if (aufg->such != NULL) {
        fprintf(stderr, "Es sind zuviele Suchoptionen angegeben\n");
        rc = -1;
    }
    else
        aufg->such = "-min";

case UNTER:
    if (aufg->such != NULL) {
        fprintf(stderr, "Es sind zuviele Suchoptionen angegeben\n");
        rc = -1;
    }
    else {
        aufg->such = "-unter";
        ++i;
        if (i >= argc || !isdigit(*argv[i])) {
            fprintf(stderr, "Es ist keine Zahl angegeben\n");
            rc = -1;
        }
        else {
            aufg->zahl = atoi(argv[i]);
        }
    }
    break;

case UEBER:
    if (aufg->such != NULL) {
        fprintf(stderr, "Es sind zuviele Suchargumente angegeben\n");
        rc = -1;
    }
    else {
        aufg->such = "-ueber";
        ++i;
        if (i >= argc || !isdigit(*argv[i])) {
            fprintf(stderr, "Es ist keine Zahl angegeben\n");
            rc = -1;
        }
        else {
            aufg->zahl = atof(argv[i]);
        }
    }
    break;

default:
    position = i;
    break;
}

if (rc < 0 || position == i)
    break;
}

if (rc < 0)

```

```

    fprintf(stderr, "Die Optionen bzw. Argumente sind fehlerhaft\n");
else
{
    print_aufgabe(aufg);
    print_dateien(position, argc, argv);
}
return rc;
}

/*
 * Hier beginnt die Testumgebung.
 * Bitte die Testumgebung nicht testen, keine Abstraktionen bilden etc.
 */
int main(argc, argv)
    int argc;
    char **argv;
{
    struct aufgabe a;
    int rc;

    if (argc == 1) {
        usage(*argv);
        rc = -1;
    }
    else
        rc = process_switches(argc, argv, &a);

    return rc;
}

```

A.9 Sample solution for “functional testing”

The component “cmdline” exhibits the following failures.

1. The option -ueber is not recognized.
2. A misspelled word (“Suchption”, should be “Suchoption”) appears in the output.
3. Abbreviated options are generally interpreted as unique and associated with the first matching option. For example, “-m” is recognized as “-mass” although the unique prefix “-ma” was not seen.
4. Search options without an accompanying measure are not treated as an error.
5. If the option “-mass” is used without a corresponding legal measure argument, no error message is printed provided that a file name (really any text) follows. However, if no argument follows the option, the program crashes.
6. The option LKHM is not processed correctly; instead, LKOM is mistakenly reported.
7. If the “-min” option is given, an error message complaining about too many search options appears because the code for “-unter” is also mistakenly executed.
8. Only the integer part of a real number is used.
9. Negative numbers are not recognized as valid numbers.
10. If all options were recognized correctly, missing data files are not detected. The command name and supplied options are mistakenly listed as files to be read.

The next page lists the equivalence classes and test cases necessary to reveal the failures described above.

Equivalence classes:

Valid classes	Invalid classes
$G_1 := \{\text{exactly 1 measure}\}$	$U_1 := \{\text{no measure}\}, U_2 := \{\text{more than 1 measure}\}$
$G_2 := \{\text{max. 1 Option}\}$	$U_3 := \{\text{more than 1 option}\}$
$G_3 := \{\text{min. 1 file}\}$	$U_4 := \{\text{no file}\}$
$G_4 := \{\text{uniquely abbreviated option}\}$	$U_5 := \{\text{ambiguously abbreviated option}\}$
$G_5 := \{\text{over/under value real}\}$	$U_6 := \{\text{missing over/under value}\}$
$G_6 := \{\text{Order measure, option}\}$	
$G_7 := \{\text{Order option, measure}\}$	
$G_{81} := \{\text{Option -max}\}$	
$G_{82} := \{\text{Option -min}\}$	
$G_{83} := \{\text{Option -durch}\}$	
$G_{84} := \{\text{Option -unter}\}$	
$G_{85} := \{\text{Option -ueber}\}$	
$G_{86} := \{\text{Option -alle}\}$	
$G_{87} := \{\text{Option -hilfe -?}\}$	
$G_{91} := \{\text{Maß -LKOM}\}$	$U_7 := \{\text{unknown measure}\}$
$G_{92} := \{\text{Maß -GKOM}\}$	
$G_{93} := \{\text{Maß -GKHM}\}$	
$G_{94} := \{\text{Maß -LKHM}\}$	
$G_{95} := \{\text{Maß -GIHE}\}$	
$G_{96} := \{\text{Maß -LIHE}\}$	

Test cases:

Nr.	Covered classes	Invocation of the program	Fail. nr.
1	U_1	cmdline -max Datei1	4
2	U_2	cmdline -mass LKOM -mass GKOM -max Datei1	—
3	U_3	cmdline -mass LKOM -alle -min Datei	7
4	U_4	cmdline -mass LKOM -durch	10
5	U_5	cmdline -mass LKOM -m Datei1	3
6	U_6	cmdline -mass GIHE -unter Datei1	—
7	U_7	cmdline -mass T Datei1	—
8	$G_1, G_2, G_3, G_{92}, G_{82}, G_6$	cmdline -mass GKOM -min Datei1	7
9	$G_4, G_6, G_7, G_{91}, G_{83}$	cmdline -du -mass LKO Datei1	—
10	G_{93}, G_{85}, G_5	cmdline -mass GKHM -ueber 4.2 Datei1	1
11	G_{94}, G_{81}, G_2	cmdline -mass LKHM -max Datei1 Datei2	6
12	G_{95}, G_{84}, G_6	cmdline -mass GIHE -unter 3.14 Datei1	8
13	G_{95}, G_{84}, G_6	cmdline -mass GIHE -unter -2 Datei1	9
14	G_{96}, G_{86}	cmdline -mass LIHE -alle Datei1	—
15	G_{87}	cmdline -hilfe	2
16	U_1, U_4	cmdline -mass	5

Note 1: Valid equivalence classes are only named the first time that they are covered.

Note 2: One obtains test case number 16 when one covers two invalid equivalence classes in a single test case. This contradicts the guidelines for using equivalence classes, but is the only possibility for revealing failure 5. This example shows that there is no perfect method.

A.10 Instructions for “structural testing” as used off-line

In this exercise you will test a program named “tokens.” Your goal is to find as many failures as possible in as little time as possible. You will generate and execute test cases with the help of the specification and the source code. These test cases should lead to 100 percent coverage of branches and other code constructs. You will execute the test cases and then search for failures. Isolating faults is not part of this exercise. **Guideline:** In the past, similar exercises required on average circa 70 minutes.

For this exercise you will need the following materials in addition to these instructions: the question sheet “Structural testing,” the instruction sheet “Coverage criteria,” and the specification and source code for the program “tokens.” At the end you will receive a sample solution.

Step 1: Preparation

- Read over the question sheet “Structural testing.” Enter your identifier and answer the first two questions.
- If you are unfamiliar with code coverage criteria, read through the instructions for attaining branch coverage etc.
- Log on to a computer. Create a new directory for your work, and change into the new directory with the “cd” command. Fetch the necessary files by entering the following command:

```
tar xf ~lott/Expt/st-tokens.tar
```

Thereafter the following files must be present:

```
Makefile    gct-map    run-suite    test-dir    tokens
```

Step 2: Generate test cases

- You received the specification and code for the program with your materials. Read through both carefully. (If you should notice possible faults in this or any of following steps, mark them. However, please do not invest any time in a precise analysis.)
- Begin to generate test data that will lead to 100% coverage of the following criteria: branch coverage, loop coverage, multiple condition coverage, and relational operator coverage. A test case consists of an input and an expected output. Think carefully in this step in order to save yourself time later. **Guideline:** In the past, on average 6 test cases were generated for similar components.
- When you are finished with this step, enter the amount of time you required in the question sheet.

Step 3: Type in the test cases

- In this exercise, a test driver is used to apply the test cases to the program. The test driver reads parameter files and invokes the program with the parameters specified in those files (options, files, etc.). If the program requires *input files*, these must be made available and mentioned in the parameter file. Example: if the program “tokens” is supposed to be

invoked with “tokens < input”, than the expression “< input” should be placed in the parameter file.

You must create not only the parameter files but also the input files. A test case = a parameter file. The expressions in a parameter file specify one invocation of the program for one run of the test driver. Parameter files are named with “.test” as the file suffix. Input files should be named sensibly, for example “empty”; they may not, however, use the file suffix “.test”.

The results are written into a file. A complete test of the program is accomplished in this way.

- Type in your test cases in files under the directory “test-dir”. Follow the specification of the test driver while doing so. When you are finished, enter the amount of time you needed in the question sheet.

Step 4: Run test cases and generate more

- Apply the test cases to the component by typing in the command “run-suite”. The results are in individual files under the directory “test-dir” and are also summarized in a file “test-results.summary”. Look at the summary of coverage with the help of the commands “gsummary” and “greport”. For example:

```
% gsummary test-dir/GCTLOG
% greport test-dir/GCTLOG
```

- Try to bring all coverage values up to 100% by generating further tests, or convince yourself that 100% coverage is not possible for some very good reasons. **Guideline:** In the past, on average **85%** total coverage was attained for similar components. You can add new tests in directory “test-dir”. After you have changed the test data, run the tests again by issuing the following sequence of commands:

```
% make clean
% run-suite
% gsummary test-dir/GCTLOG
% greport test-dir/GCTLOG
```

- When you are finished with this step, enter the number of test cases, the coverage values attained, and the amount of time you needed in the question sheet. Print out the test results from the file “test-results.summary”.

Step 5: Search for failures

- Look over the results carefully. Find possible failures by comparing the expected results according to the specification with the output of your test cases. Mark the detected failures in both copies of the output with circles, etc. **Guideline:** In the past, on average **6** failures were detected in similar components.
- When you believe that you have detected all failures, please enter the time you required in the question sheet. Complete the rest of the question sheet and hand it in with your test results.

A.11 Instructions for “structural testing” as used on-line

For this exercise you will need the following materials in addition to these instructions: the instruction sheet “Coverage criteria,” and the specification and source code for the program “tokens.” At the end you will receive a sample solution.

1. Log on to a computer. Start the menu bar for the information system by entering the following command:

```
~lott/Expt/menu-st
```

2. Use the menu bar to start one instance of the information system “MVP-S.” Select the project “Strukturelles_Testen,” wherein you take on the role “Test_Ingenieur.”
3. Use the menu bar again to start one instance of the question sheet tool.
4. Please use the information system for the rest of the information about the exercise.

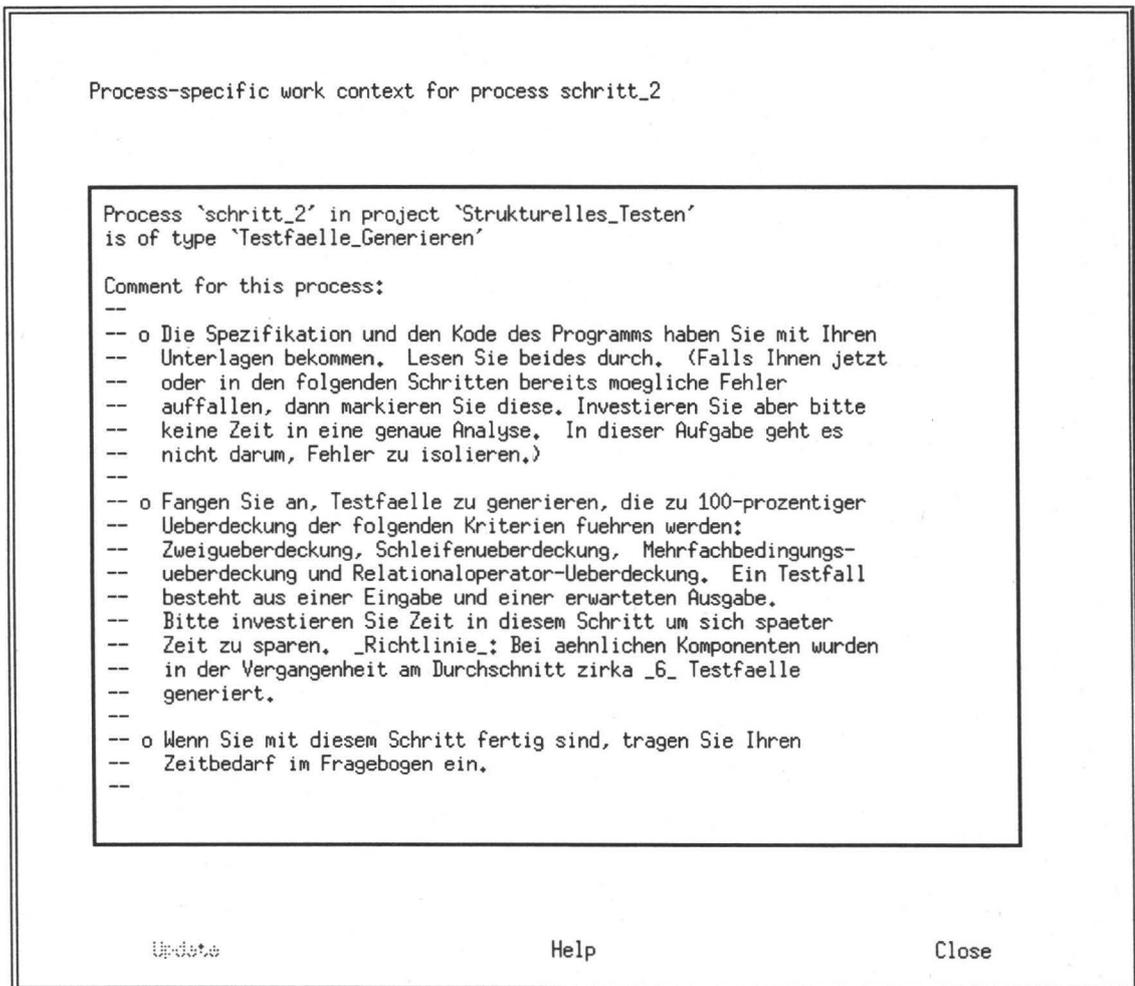


Figure A.5: MVP-S process-context window for structural testing step "schritt_2"

A.12 Question sheet “structural testing” as used off-line

Your ID: _____ Date: _____

Note: For all time-related entries, please deduct time for pauses, etc.

1. How motivated are you for the experiment? [Please mark the scale.]

Estimate						
Value	0	1	2	3	4	5
Comparison	not at all	little	so-so	pretty much	quite a bit	very much

2. When did you start the experiment?

_____ : _____ [Hour:Minute]

3. How much time did you need to construct the test cases (step 2)?

_____ [Minutes]

4. How much time did you need to type in the test cases (step 3)?

_____ [Minutes]

5. How long did the execution/generation of test cases (step 4) require?

_____ [Minutes]

6. How many test cases did you finally generate?

_____ [Count]

7. List the coverage values that you finally attained with your test cases:

Binary branch _____ [%], Switch _____ [%], Loop _____ [%], Operator Instrumentation _____ [%], Summary of all Condition Types _____ [%]

8. How long did you search for failures (step 5)?

_____ [Minutes]

9. How many different failures did you find?

_____ [Count]

10. When did you end the experiment?

_____ : _____ [Hour:Minute]

11. Would you be inclined to perform another iteration in order to find more failures?

_____ [yes, no]

12. Have you mastered functional testing? [Please mark the scale]

Estimate						
Value	0	1	2	3	4	5
Comparison	not at all	little	so-so	pretty much	quite a bit	very much

A.13 Question sheet "structural testing" as used on-line

Figure A.6 shows the on-line question sheet used by subjects who use the process-centered system to perform structural testing.

fb-st	
C-g eingeben fuer das Menu, C-z fuer Hilfe. Speichern Beenden	
Fragebogen "Structurelles Testen"	
Ihre Kennung: ■ _____	Datum: <u>27. Jan 1996</u>
1. Wie motiviert sind Sie fuer das Experiment? (Schaetzung)	_____
2. Wann haben Sie mit dem Experiment angefangen?	_____
3. Wie lange hat die Generierung von Testfaellen (Schr. 2) gedauert?	_____
4. Wie lange hat das Eintippen (Schr. 3) gedauert?	_____
5. Wie lange hat die Durchfuehrung/Generierung (Schr. 4) gedauert?	_____
6. Wieviele Testfaelle haben Sie letztendlich generiert?	_____
7. Geben Sie die Coverage-Werte an, die Sie letztendlich erreicht haben.	
Binary branch:	_____
Switch:	_____
Loop:	_____
Operator Instrumentation:	_____
Summary of all Condition Types::	_____
8. Wie lange hat die Suche nach Fehlverhalten (Schr. 5) gedauert?	_____
9. Wieviele verschiedene Fehlverhalten haben Sie gefunden?	_____
10. Wann haben Sie das Experiment beendet?	_____
11. Wuerden Sie eine Weitere Iteration anstreben?	_____
12. Wie gut beherrschen Sie strukturelles Testen (Schaetzung)?	_____
Kennung: Bitte Ihre Kennung eintragen.	

Figure A.6: On-line question sheet used by structural testers

A.14 Explanation of coverage criteria

Binary branch & switch instrumentation

Fulfilling the branch coverage criteria makes certain that each branch in a component was executed at least once. Branches in C programs are created by `if`, `?`, `for`, `while`, and `do-while` statements. Each of these statements creates two conditions (branches): the evaluation of the test expression must yield true once and false once. In GCT terminology, this type of branch coverage is called **binary branch instrumentation**.

Branches are also created by the `switch-case` construct. To test all branches, all `case` labels must be branched to at least once. This includes the `default` label, even if it was not explicitly written in the source code. Each `case` label produces a single condition. In GCT terminology, this type of branch coverage is called **switch instrumentation**.

Loop instrumentation

A `do-while` loop creates two conditions for this criteria: one for executing the loop exactly one time, and one for executing the loop more than one time. The `for` and `while` loop add a third condition: the idea of executing the loop zero times; i.e., the test expression is false when it is evaluated for the first time.

Multiple condition instrumentation

Multiple conditions are expressions constructed using the operators “logical and” (`&&`) and/or “logical or” (`||`). To fulfill these criteria for a two-part expression (i.e., an expression with exactly one logical operator), each side of the expression must be true once and false once. In other words, a two-part multiple-condition expression creates four conditions: false/false, true/false, false/true, and true/true. Nested expressions create correspondingly more conditions. When checking the coverage of multiple conditions, all expressions that evaluate to a boolean value are checked, not only the test expressions from `if` statements.

Relational operator instrumentation

Relational operators are the operators that compare values with each other (e.g., `=`, `!=`, `<`, `<=`, etc.). The relational operators (`<`, `<=`, `>`, `>=`) are often the cause of faults, whether because of swapped operators (e.g., `<` instead of `>`), incorrect use (e.g., `<` instead of `<=`), or incorrect boundary values (e.g., “`a < 99`” instead of “`a < 100`”). A relational operator creates two (sometimes three) conditions:

1. For all operators: The left side must equal the right side.
2. For all operators: The left side must not equal the right side.
3. For comparisons of relative size, the following boundary values should be tested:
 - (a) For the operators `<=` und `>`: the left side must be 1 smaller than the right side
 - (b) For the operators `>=` und `<`: the left side must be 1 larger than the right side

Note: when you write your test cases, try to cover as much functionality of the program as possible with each case. This will help you avoid an unnecessarily large number of test cases.

A.15 Specification of program “tokens”

Name

tokens – sort and count alphanumeric tokens

Usage

tokens [**-ai**] [**-c chars**] [**-m count**]

Description

tokens reads words from the standard input (“stdin”). A word is a series of sequential characters from a given character set. This character set includes by default the alphabetic characters A–Z and a–z as well as the digits 0–9. Depending on the options (see below), the digits can be eliminated from the character set, or other characters can be added to it. Characters that are not in the character set (e.g., spaces) divide words from each other, but are otherwise ignored. At the end of the input, each word is printed exactly once with its frequency of appearance in the input. Thereby the words are sorted in increasing lexicographic order.

Options

- “-a”: Allow only alphabetic characters in tokens (no digits 0–9).
- “-c chars”: Allow ‘chars’ to be part of tokens.
- “-i”: Ignore the difference between upper and lower case by mapping all input to lower case.
- “-m count”: The value ‘count’ indicates the minimum frequency needed for a word to be printed.

Example

To sort and count all alphabetical words in file “xyz”:

```
tokens -a < xyz
```

Limits

The maximum count of unique words is limited by the memory of the computer. The maximum frequency count of a single word is limited by the length of an “integer”.

Author

Gary Perlman

A.16 Source code of program “tokens”

The program “tokens” is copyright 1992 by Gary Perlman. Adapted with permission.

```
/*COPYRIGHT (c) Copyright 1992 Gary Perlman */

#include <stdio.h>
#include <string.h>
#include <assert.h>

int    Ignore = 0;
int    Mincount = 0;
int    Alpha = 0;
char   MapAllowed[256];

typedef struct tnode
{
    char   *contents;
    int    count;
    struct tnode *left;
    struct tnode *right;
} TNODE;

void treeprint(tree) TNODE *tree;
{
    if (tree != NULL)
    {
        treeprint(tree->left);
        if (tree->count > Mincount)
            printf("%7d\t%s\n", tree->count, tree->contents);
        treeprint(tree->right);
    }
}

TNODE *
install(string, tree) char *string; TNODE * tree;
{
    int cond;
    assert(string != NULL);
    if (tree == NULL)
    {
        if (tree = (TNODE *) calloc(1, sizeof(TNODE)))
        {
            tree->contents = strdup(string);
            tree->count = 1;
        }
    }
    else
    {
        cond = strcmp(string, tree->contents);
        if (cond < 0)
            tree->left = install(string, tree->left);
        else if (cond == 0)
            tree->count++;
        else
    }
}
```

```

        tree->right = install(string, tree->right);
    }
    return(tree);
}

char *
getword(ioptr) FILE *ioptr;
{
    static char string[1024];
    char *ptr = string;
    register int c;
    assert(ioptr != NULL);
    for (;;)
    {
        c = getc(ioptr);
        if (c == EOF)
            if (ptr == string)
                return(NULL);
            else
                break;
        if (!MapAllowed[c])
            if (ptr == string)
                continue;
            else
                break;
        *ptr++ = MapAllowed[c];
    }
    *ptr = '\0';
    return(string);
}

void tokens(ioptr) FILE *ioptr;
{
    TNODE *root = NULL;
    char *s;
    assert(ioptr != NULL);
    while (s = getword(ioptr))
        root = install(s, root);
    treeprint(root);
}

int main(argc, argv) int argc; char ** argv;
{
    int c, errcnt = 0;
    extern char *optarg;

    while ((c = getopt(argc, argv, "ac:im:")) != EOF)
        switch(c)
        {
            case 'a': Alpha = 0; break;
            case 'c':
                while (*optarg)
                {
                    MapAllowed[*optarg] = *optarg;
                    optarg++;
                }
            }
}

```

```

    }
    break;
case 'i': Ignore = 1; break;
case 'm': Mincount = atoi(optarg); break;
default: errcnt++;
}
if (errcnt)
{
    fprintf(stderr, "Usage: %s [-i] [-c chars] [-m count]\n", *argv);
    return(1);
}
for (c = 'a'; c <= 'z'; c++)
    MapAllowed[c] = c;
for (c = 'A'; c <= 'Z'; c++)
    MapAllowed[c] = Ignore ? c - 'A' + 'a' : c;
if (!Alpha)
    for (c = '0'; c <= '9'; c++)
        MapAllowed[c] = c;
tokens(stdin);
return(0);
}

```

A.17 Sample solution for “structural testing”

The component “tokens” has the following faults and exhibits the following failures.

1. Fault in function “treeprint”, line 25: The symbol “>” should be “>=”.
Causes failure: If a boundary value n is given with the “-m” argument, the value $n + 1$ is used instead of just n .
2. Fault in function “getword”, line 64/77: The length of the input is not checked.
Causes failure: The program dumps core if a very long word is read.
3. Fault in function “main”, line 97 (circa): The array “MapAllowed” is never initialized.
Causes failure: Compiler-dependent; non-alphanumeric symbols could be mistakenly accepted.
4. Fault in function “main”, line 101: The variable “Alpha” is initialized with 0 instead of 1.
Causes failure: The argument “-a” has no effect.
5. Fault in function “main”, line 110: The program does not check whether a valid number was supplied.
Causes failure: Invalid numbers are accepted without an error message.
6. Fault in function “main”, line 115: The argument “-a” is not part of the usage message.
Causes failure: The help message says nothing about the “-a” argument.

The following test cases attain the maximum possible coverage values for the program.

1. Test case: Unknown option

Invocation with: -q

Input: none

Expected output: Error message, usage message for the program.

2. Test case: No options, empty file.

Invocation with: < /dev/null

Input: none.

Expected output: none.

3. Test case: One option, empty file.

Invocation with: -c ” < /dev/null

Input: none.

Expected output: : none.

4. Test case: Few options, very short file.

Invocation with: -c / < shortfile

Input: A file named “shortfile” with the single word “nonl” and no further characters before the end of the file; i.e., no so-called “newline.”

Expected output: The following single line:

1 nonl

5. Test case: Many options, short file.

Invocation with: -a -i -c /. -m 1 < infile

Input: A file named “infile” with the following entries:

word1
word1
word1
word2
word2
*
b
a

Expected output: The following lines:

1 a
1 b
3 word1
2 word2

End of the necessary test cases.

The following coverage values result from executing the test cases listed above.

```
% gsummary test-dir/GCTLOG
BINARY BRANCH INSTRUMENTATION (38 conditions total)
2 ( 5.26%) not satisfied.
36 (94.74%) fully satisfied.

SWITCH INSTRUMENTATION (5 conditions total)
0 ( 0.00%) not satisfied.
5 (100.00%) fully satisfied.

LOOP INSTRUMENTATION (21 conditions total)
7 (33.33%) not satisfied.
14 (66.67%) fully satisfied.

OPERATOR INSTRUMENTATION (15 conditions total)
0 ( 0.00%) not satisfied.
15 (100.00%) fully satisfied.

SUMMARY OF ALL CONDITION TYPES (79 total)
9 (11.39%) not satisfied.
70 (88.61%) fully satisfied.
```

The values that are less than 100% can be explained further using the command “greport”. That command yields the following output:

```
% greport test-dir/GCTLOG
"tokens.c", line 38: if was taken TRUE 5, FALSE 0 times.
"tokens.c", line 64: loop zero times: 0, one time: 3, many times: 8.
"tokens.c", line 118: loop zero times: 0, one time: 0, many times: 3.
"tokens.c", line 120: loop zero times: 0, one time: 0, many times: 3.
"tokens.c", line 122: if was taken TRUE 3, FALSE 0 times.
"tokens.c", line 123: loop zero times: 0, one time: 0, many times: 3.
```

Better coverage value cannot be obtained, as discussed below:

[Line 38:] This “if”-condition tests whether “calloc” actually allocated memory. It is difficult to cause calloc to fail, especially for users who are not familiar with the shell’s “limit” command.

[Line 64:] This is a loop with empty start and termination criteria. Therefore this loop is always executed at least once.

[Lines 118, 120, and 123:] These “for” loops have hard-coded, fixed termination criteria, and are always executed more than once.

[Line 122:] This “if” condition is always true due to a fault in the program (the “Alpha” variable is not assigned a value).

6. How understandable were the instructions? Were the instructions made more or less understandable by moving them onto the computer screen?

7. Would a graphical or other non-text version of the instructions have been better, or was the textual description best suited for the goal of understanding the exercise?

8. Other comments?

Bibliography

- [AG88] Robert M. Adams and Mark D. Gavoor. Quality function deployment: Its promise and reality. Rockwell International Automotive Operations, Troy Michigan 48084, August 1988.
- [Ama94] Amadeus Software Research. Amadeus Measurement System: User's Guide (version 2.3.4). 12 Owen Court, Irvine, CA 92715, USA, October 1994.
- [Bas75] Victor R. Basili. Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering*, SE-1(4):390–396, December 1975.
- [Bas85] Victor R. Basili. Quantitative evaluation of software methodology. In *Proceedings of the First Pan-Pacific Computer Conference*, Melbourne, Australia, September 1985.
- [Bas92] Victor R. Basili. Software modeling and measurement: The Goal/Question/Metric paradigm. Technical Report CS-TR-2956, Department of Computer Science, University of Maryland, College Park, MD 20742, September 1992.
- [Bas94] Victor R. Basili. A research agenda for ISERN: Validating the Quality Improvement Paradigm. Presented at the 1994 ISERN annual meeting, October 1994.
- [BBF93] Steve Benford, Edmund Burke, and Eric Foxley. Learning to construct quality software with the Ceilidh system. *Software Quality Journal*, 2(3):177–197, September 1993.
- [BBK⁺78] Barry W. Boehm, John R. Brown, Hans Kaspar, Myron Lipow, Gordon J. Macleod, and Michael J. Merrit. *Characteristics of software quality*. North-Holland, 1978.
- [BBL76] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the Second International Conference on Software Engineering*, pages 592–605. IEEE Computer Society Press, October 1976.
- [BCR94a] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Goal Question Metric Paradigm. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 528–532. John Wiley & Sons, 1994.
- [BCR94b] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Measurement. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 646–661. John Wiley & Sons, 1994.
- [BD94] Alfred Bröckers and Christiane Differding. Process description of a development process at Robert Bosch GmbH (in German). Technical Report STTI-94-10-D,

Software Technology Transfer Initiative Kaiserslautern, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1994.

- [BDT96] Alfred Bröckers, Christiane Differding, and Günter Threin. The role of software process modeling in planning industrial measurement programs. In *Proceedings of the Third International Software Metrics Symposium*, Berlin, March 1996. IEEE Computer Society Press.
- [BHH78] G. E. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters*. John Wiley & Sons, New York, 1978.
- [BLRV95] Alfred Bröckers, Christopher M. Lott, H. Dieter Rombach, and Martin Verlage. MVP-L language report version 2. Technical Report 265/95, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1995.
- [BR88] Victor R. Basili and H. Dieter Rombach. The TAME Project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, SE-14(6):758-773, June 1988.
- [BW81] Victor R. Basili and David M. Weiss. Evaluation of a software requirements document by analysis of change data. In *Proceedings of the Fifth International Conference on Software Engineering*, pages 314-323. IEEE Computer Society Press, 1981.
- [BW84] Victor R. Basili and David M. Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, SE-10(6):728-738, November 1984.
- [BWT85] Thomas P. Bowen, Gary B. Wigle, and Jay T. Tsai. Specification of software quality attributes, volume I: Final report. Technical Report RADC-TR-85-37, vol. I, Rome Air Development Center, Griffiss AFB, Rome, NY 13441-5700, February 1985. Available from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145, order number AD-A153 988.
- [Cap92] Cap Gemini Innovation. Process Weaver User's Manual, version PS 1.2. Cap Gemini Innovation, F-38942 Meylan Cedex, France, 1992.
- [CDP95] David C. Carr, Ashok Dandekar, and Dewayne E. Perry. Experiments in process interface descriptions, visualizations and analyses. In W. Schäfer, editor, *Proceedings of the Fourth European Workshop on Software Process Technology*, pages 119-137, Noordwijkerhout, The Netherlands, April 1995. Lecture Notes in Computer Science Nr. 913, Springer-Verlag.
- [Chr94] Alan M. Christie. A practical guide to the technology and adoption of software process automation. Technical Report CMU/SEI-94-TR-007, Software Engineering Institute, Carnegie Mellon University, March 1994.
- [CKO92] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75-90, September 1992.
- [CM83] C. L. Carpenter, Jr. and Gerald E. Murine. Applying software quality metrics. In *ASQC Quality Congress Transactions*, Milwaukee, Wisconsin, 1983. American Society for Quality Control.

- [CML⁺90] R. Conradi, C. C. Malm, E. Lyngra, P. H. Westby, and C. Liu. The EPOS approach to the software process model example problem. In *Collected Solutions from the Sixth International Software Process Workshop*, October 1990.
- [CS66] Donald T. Campbell and Julian C. Stanley. *Experimental and Quasi-Experimental Designs for Research*. Houghton Mifflin, Boston, 1966. ISBN 0-395-30787-2.
- [dBS95] Tineke de Bunje and Alison Saunders. Combining process models and metrics in practice. In W. Schäfer, editor, *Proceedings of the Fourth European Workshop on Software Process Technology*, pages 49–53, Noordwijkerhout, The Netherlands, April 1995. Lecture Notes in Computer Science Nr. 913, Springer-Verlag.
- [Dem86] W. Edwards Deming. *Out of the crisis*. Massachusetts Institute of Technology, Cambridge, Mass., 1986.
- [DG90] Wolfgang Deiters and Volker Gruhn. Managing software processes in the environment MELMAC. *ACM SIGSOFT Software Engineering Notes*, 15(6):193–205, December 1990.
- [DHL96] Christiane Differding, Barbara Hoisl, and Christopher M. Lott. Technology package for the Goal Question Metric Paradigm. Technical Report 281-96, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, April 1996.
- [Dia94] John R. Diamant. Human interaction support in HP SynerVision for SoftBench. In C. Ghezzi, editor, *Proceedings of the Ninth International Software Process Workshop*, pages 44–46. IEEE Computer Society Press, October 1994.
- [Dif93] Christiane Differding. An object model for supporting the GQM paradigm (in German). Master's thesis, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, June 1993.
- [Dio93] Raymond Dion. Process improvement and the corporate balance sheet. *IEEE Software*, 10(4):28–35, July 1993.
- [DV92] W. Decker and Jon Valett. Software management environment (SME) concepts and architecture. Technical Report SEL-89-103, NASA Goddard Space Flight Center, Greenbelt MD 20771, September 1992.
- [Fel95] Raimund L. Feldmann. A tool for data preparation for a measurement program at Robert Bosch GmbH (in German). Projektarbeit STTI-95-01-D, Software Technology Transfer Initiative Kaiserslautern, University of Kaiserslautern, 67653 Kaiserslautern, Germany, February 1995.
- [Fer93] Christer Fernström. Process WEAVER: Adding process support to UNIX. In *Proceedings of the Second International Conference on the Software Process*, pages 12–26. IEEE Computer Society Press, February 1993.
- [FH92] Peter H. Feiler and Watts S. Humphrey. Software process development and enactment: concepts and definitions. Technical Report CMU/SEI-92-TR-04, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, September 1992.

- [GHLR94] P. Giese, B. Hoisl, C. M. Lott, and H. D. Rombach. Data collection in a process-sensitive software engineering environment. In *Proceedings of the Ninth International Software Process Workshop*, pages 47–49, October 1994.
- [GHW95] Christiane Gresse, Barbara Hoisl, and Jürgen Wüst. A process model for planning GQM-based measurement. Technical Report STTI-95-04-E, Software Technology Transfer Initiative, University of Kaiserslautern, 67653 Kaiserslautern, Germany, October 1995.
- [Gib94] W. Wayt Gibbs. Software's chronic crisis. *Scientific American*, pages 86–95, September 1994.
- [Gla94] Robert L. Glass. The software-research crisis. *IEEE Software*, 11(6):42–47, November 1994.
- [Gru91] Volker Gruhn. Analysis of software process models in the software process management environment MELMAC. In Fred Long, editor, *Proceedings of the Conference on Software Engineering Environments*, pages 67–90, New York, London, November 1991. Ellis Horwood.
- [Ham94a] Dirk Hamann. A tool for data collection for a measurement program at Robert Bosch GmbH (in German). Projektarbeit STTI-94-11-D, Software Technology Transfer Initiative Kaiserslautern, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1994.
- [Ham94b] Dirk Hamann. A tool for data collection for a measurement program at Robert Bosch GmbH, system documentation (in German). Projektarbeit STTI-94-12-D, Software Technology Transfer Initiative Kaiserslautern, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1994.
- [HB92] Joel Henry and Bob Blasewitz. Process definition: theory and reality. *IEEE Software*, 9:103–105, November 1992.
- [Hew94] Hewlett Packard Company. SynerVision marketing literature, 1992-1994.
- [HF94] Tracy Hall and Norman Fenton. Implementing software metrics – the critical success factors. *Software Quality Journal*, 3(4):195–208, December 1994.
- [Hoi94] Barbara Hoisl. A process model for planning GQM-based measurement. Technical Report STTI-94-06-E, Software-Technology-Transfer-Initiative Kaiserslautern, University of Kaiserslautern, 67653 Kaiserslautern, Germany, April 1994.
- [HSW91] Watts S. Humphrey, Terry R. Snyder, and Ronald R. Willis. Software process improvement at Hughes Aircraft. *IEEE Software*, 8:11–23, July 1991.
- [Huf93] Karen E. Huff. Software process measurement session summary. In Wilhelm Schäfer, editor, *Proceedings of the Eighth International Software Process Workshop*, pages 18–21. IEEE Computer Society Press, March 1993.
- [Hum89] Watts S. Humphrey. *Managing the Software Process*. Addison Wesley, Reading, Massachusetts, 1989.
- [IMF90] A. Irvine, C. McGowan, and P Feiler. IDEF0/SADT solutions to ISPW problem. In *Collected Solutions from the Sixth International Software Process Workshop*, October 1990.

- [KA83] M. Kogure and Y. Akao. Quality function deployment and CWQC in Japan. *Quality Progress*, October 1983.
- [Kad92] R. Kadia. Issues encountered in building a flexible software development environment. In H. Weber, editor, *Proceedings of the Fifth ACM SIGSOFT/SIGPLAN Symposium on Software Development Environments*, pages 169–180, 1992. Appeared as ACM SIGSOFT Software Engineering Notes 17(5), December 1992.
- [Kat89] Takuya Katayama. A hierarchical and functional software process description and its enactment. In *Proceedings of the Eleventh International Conference on Software Engineering*, pages 343–352. IEEE Computer Society Press, May 1989.
- [Kat90] Takuya Katayama, editor. *Proceedings of the Sixth International Software Process Workshop*. IEEE Computer Society Press, October 1990.
- [KB93] Balachander Krishnamurthy and Naser S. Barghouti. Provence: a process visualization and enactment environment. In Ian Sommerville and Manfred Paul, editors, *Proceedings of the Fourth European Software Engineering Conference*, pages 451–465. Lecture Notes in Computer Science Nr. 717, Springer-Verlag, 1993.
- [KBS90] Gail Kaiser, N. S. Barghouti, and M. H. Sokolsky. Preliminary experience with process modeling in the MARVEL software development environment kernel. In *Proceedings of the 23rd Annual Hawaii International Conference on System Sciences*, volume II, pages 131–140. IEEE Computer Society Press, January 1990.
- [KC90] Michael Karr and Thomas E. Cheatham. A solution to the ISPW-6 software process modeling example. In *Collected Solutions from the Sixth International Software Process Workshop*, October 1990.
- [Kel89] Mark I. Kellner. Software process modeling: value and experience. In *SEI Technical Review*, pages 23–54. Software Engineering Institute, Pittsburgh, Pennsylvania 15213, 1989.
- [KFF⁺90] Marc I. Kellner, Peter H. Feiler, Anthony Finkelstein, Takuya Katayama, Leon J. Osterweil, Maria H. Penedo, and H. Dieter Rombach. Software process modeling example problem. In Takuya Katayama, editor, *Proceedings of the Sixth International Software Process Workshop*, pages 19–29. IEEE Computer Society Press, October 1990.
- [KFP88] Gail E. Kaiser, Peter H. Feiler, and Steven S. Popovich. Intelligent assistance for software development and maintenance. *IEEE Software*, 5:40–49, May 1988.
- [Kin89] Bob King. *Better Designs in Half the Time: Implementing QFD Quality Function Deployment in America*. GOAL/QPC, Methuen, MA, 3rd edition, 1989.
- [KL95a] Erik Kamsties and Christopher M. Lott. An empirical evaluation of three defect-detection techniques. Technical Report ISERN 95-02, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, May 1995.
- [KL95b] Erik Kamsties and Christopher M. Lott. An empirical evaluation of three defect-detection techniques. In W. Schäfer and P. Botella, editors, *Proceedings of the Fifth European Software Engineering Conference*, pages 362–383. Lecture Notes in Computer Science Nr. 989, Springer-Verlag, September 1995.

- [KMKT90] Shinji Kusumoto, Ken-ichi Matsumoto, Tohru Kikuno, and Koji Torii. Ginger: Data collection and analysis system. Technical Report SS 90-5, Osaka University, Toyonaka, Osaka 560, Japan, 1990.
- [KNMS⁺92] C. D. Klingler, M. Neviasser, A. Marmor-Squires, C. M. Lott, and H. D. Rombach. A case study in process representation using MVP-L. In *Proceedings of the Seventh Annual Conference on Computer Assurance (COMPASS 92)*, pages 137-146, June 1992.
- [KR90] Marc I. Kellner and H. Dieter Rombach. Session summary: Comparisons of software process descriptions. In Takuya Katayama, editor, *Proceedings of the Sixth International Software Process Workshop*, pages 7-18. IEEE Press, October 1990.
- [LHR95] Christopher M. Lott, Barbara Hoisl, and H. Dieter Rombach. The use of roles and measurement to enact project plans in MVP-S. In W. Schäfer, editor, *Proceedings of the Fourth European Workshop on Software Process Technology*, pages 30-48, Noordwijkerhout, The Netherlands, April 1995. Lecture Notes in Computer Science Nr. 913, Springer-Verlag.
- [Lot93] Christopher M. Lott. Process and measurement support in SEEs. *ACM SIGSOFT Software Engineering Notes*, 18(4):83-93, October 1993.
- [Lot94] Christopher M. Lott. Measurement support in software engineering environments. *International Journal of Software Engineering & Knowledge Engineering*, 4(3):409-426, September 1994.
- [LR93] Christopher M. Lott and H. Dieter Rombach. Measurement-based guidance of software projects using explicit project plans. *Information and Software Technology*, 35(6/7):407-419, June/July 1993.
- [Mat90] Ken-ichi Matsumoto. *A programmer performance model and its measurement environment*. PhD thesis, Osaka University, Toyonaka, Osaka 560, Japan, August 1990.
- [McG90] Frank E. McGarry. Results of 15 years of measurement in the SEL. In *Proceedings of the Fifteenth Annual Software Engineering Workshop*. NASA Goddard Space Flight Center, Greenbelt MD 20771, November 1990.
- [MDTR93] Vahid Mashayekhi, Janet M. Drake, Wei-Tek Tsai, and John Riedl. Distributed, collaborative software inspection. *IEEE Software*, 10:66-75, September 1993.
- [MIK⁺87] Ken-ichi Matsumoto, Katsuro Inoue, Hideo Kudo, Yuki Sugiyama, and Koji Torii. Error life span and programmer performance. In *Proceedings of the Eleventh Annual International Computer Software and Application Conference (COMPSAC)*, pages 259-265. IEEE Computer Society Press, October 1987.
- [MKKT93] Ken-ichi Matsumoto, Shinji Kusumoto, Tohru Kikuno, and Koji Torii. A new framework of measuring software development processes. In *Proceedings of the First International Software Metrics Symposium*, pages 108-118. IEEE Computer Society Press, May 1993.

- [MP90] Frank E. McGarry and R. Pajerski. Towards understanding software - 15 years in the SEL. In *Proceedings of the Fifteenth Annual Software Engineering Workshop*. NASA Goddard Space Flight Center, Greenbelt MD 20771, November 1990.
- [MRW77] James A. McCall, Paul K. Richards, and Gene F. Walters. Factors in software quality, volume I: Concepts and definitions of software quality. Technical Report RADC-TR-77-369, vol. I, Rome Air Development Center, Griffiss AFB, Rome, NY 13441-5700, July 1977. Available from Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145, order number AD-A049 014.
- [MS92] Peiwei Mi and Walt Scacchi. Process integration in CASE environments. *IEEE Software*, 9:45–53, March 1992.
- [Mur80] Gerald E. Murine. Applying software quality metrics in the requirements analysis phase of a distributive system. Technical report, Metriqs Incorporated, 390 Oak Avenue, Carlsbad, California 92008, 1980.
- [Nat90] National Aeronautics and Space Administration. Software Engineering Laboratory (SEL) Database Organization and User's Guide, Revision 1. Technical Report SEI-89-101, NASA Goddard Space Flight Center, Greenbelt MD 20771, February 1990.
- [OB92] Markku Oivo and Victor R. Basili. Representing software engineering models: The TAME goal oriented approach. *IEEE Transactions on Software Engineering*, 18(10):886–898, October 1992.
- [Oiv90] Markku Oivo. *Knowledge-based Support for Embedded Computer Software Analysis and Design*. VTT Publication 68, Espoo, September 1990. ISBN 951-38-3763-7.
- [Pff93] Shari Lawrence Pfleeger. Lessons learned in building a corporate metrics program. *IEEE Software*, 10:67–74, May 1993.
- [Pff95] Shari Lawrence Pfleeger. Maturity, models and goals: How to build a metric plan. *Journal of Systems and Software*, 31(2):143–155, November 1995.
- [PS90a] Burkhard Peuschel and Wilhelm Schäfer. ISPW–6 exercise solution. In *Collected Solutions from the Sixth International Software Process Workshop*, October 1990.
- [PS90b] Adam A. Porter and Richard W. Selby. Empirically guided software development using metric-based classification trees. *IEEE Software*, 7(2):46–54, March 1990.
- [PS91] Maria H. Penedo and Christine Shu. Acquiring experience with the modeling and implementation of the project life-cycle process. *IEE Software Engineering Journal*, 6(5):259–274, September 1991.
- [PSW92] Burkhard Peuschel, Wilhelm Schäfer, and Stefan Wolf. A knowledge-based software development environment supporting cooperative work. *International Journal of Software Engineering & Knowledge Engineering*, 2(1):79–106, 1992.
- [RH68] Raymond J. Rubey and R. Dean Hartwick. Quantitative measurement of program quality. In *Proceedings of the 23rd National Conference*, pages 671–677. ACM Publications, August 1968.

- [RLL93] Raimo Rask, Petteri Laamanen, and Kalle Lyytinen. Simulation and comparison of Albrecht's function point and DeMarco's function bang metrics in a CASE environment. *IEEE Transactions on Software Engineering*, 19(7):661–671, July 1993.
- [Rom89] H. Dieter Rombach. The role of measurement in ISEEs. In Carlo Ghezzi and John McDermid, editors, *Proceedings of the Second European Software Engineering Conference*, pages 65–85. Lecture Notes in Computer Science Nr. 387, Springer-Verlag, September 1989.
- [Rom91a] H. Dieter Rombach. MVP-L: A language for process modeling in-the-large. Technical Report CS-TR-2709, Department of Computer Science, University of Maryland, College Park, MD, 20742, 1991.
- [Rom91b] H. Dieter Rombach. Practical benefits of goal-oriented measurement. In N. Fenton and B. Littlewood, editors, *Software Reliability and Metrics*, pages 217–235. Elsevier Applied Science, London, 1991.
- [RU89a] H. Dieter Rombach and Bradford T. Ulery. Establishing a measurement based maintenance improvement program: Lessons learned in the SEL. Technical Report CS-TR-2252, Department of Computer Science, University of Maryland, College Park, MD 20742, May 1989.
- [RU89b] H. Dieter Rombach and Bradford T. Ulery. Improving software maintenance through measurement. *Proceedings of the IEEE*, 77(4):581–595, April 1989.
- [RUV92] H. Dieter Rombach, Bradford T. Ulery, and Jon Valett. Toward full life cycle control: Adding maintenance measurement to the SEL. *Journal of Systems and Software*, 18(2):125–138, May 1992.
- [SAY85] Toshihiko Sunazuka, Motoei Azuma, and Noriko Yamagishi. Software quality assessment technology. In *Proceedings of the Eighth International Conference on Software Engineering*, pages 142–149. IEEE Computer Society Press, August 1985.
- [SHO89] Stanley M. Sutton, Jr., Dennis Heimbigner, and Leon Osterweil. APPL/A: A prototype language for software process programming. Technical Report CU-CS-448-89, Department of Computer Science, University of Colorado at Boulder, Boulder, CO, October 1989.
- [SJM⁺89] Richard W. Selby, Greg James, Kent Madsen, Joan Mahoney, Adam A. Porter, and Douglas C. Schmidt. Classification tree analysis using the Amadeus measurement and empirical analysis system. In *Proceedings of the Fourteenth Annual Software Engineering Workshop*. NASA Goddard Space Flight Center, Greenbelt MD 20771, 1989.
- [SPSB91] Richard W. Selby, Adam A. Porter, Doug C. Schmidt, and Jim Berney. Metric-driven analysis and feedback systems for enabling empirically guided software development. In *Proceedings of the Thirteenth International Conference on Software Engineering*, pages 288–298. IEEE Computer Society Press, May 1991.
- [Tic85] Walter Tichy. RCS—a system for version control. *Software-Practice and Experience*, 15(7):637–654, July 1985.

- [TLPH95] Walter F. Tichy, Paul Lukowicz, Lutz Prechelt, and Ernst A. Heinz. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software*, 28(1):9–18, January 1995.
- [TMK95] Koji Torii, Ken-ichi Matsumoto, and Shinji Kusumoto. GINGER: A quantitative analysis environment for improving programmer performance. Technical report, Graduate School of Information Science, Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara 630-01, Japan, April 1995. Submitted to TSE.
- [Ver94] Martin Verlage. Multi-view modeling of software processes. In Brian C. Warboys, editor, *Proceedings of the Third European Workshop on Software Process Technology*, pages 123–127, Grenoble, France, 1994. Nr. 772, Springer-Verlag.
- [Vis94] Giuseppe Visaggio. Process improvement through data reuse. *IEEE Software*, 11(4):76–85, July 1994.
- [VZ95] Lawrence G. Votta and Mary Zajac. A design process improvement case study using process waiver data. In W. Schäfer and P. Botella, editors, *Proceedings of the Fifth European Software Engineering Conference*, pages 44–58. Lecture Notes in Computer Science Nr. 989, Springer-Verlag, September 1995.
- [WF91] Kurt C. Wallnau and Peter H. Feiler. Tool integration and environment architectures. Technical Report CMU/SEI-91-TR-11, Software Engineering Institute, Carnegie Mellon University, May 1991.
- [WM79] Gene F. Walters and James A. McCall. Software quality metrics for life-cycle cost-reduction. *IEEE Transactions on Reliability*, R-28(3):212–220, August 1979.
- [Zul93] Richard E. Zultner. TQM for technical teams. *Communications of the ACM*, 36(10):79–91, October 1993.