Technical Report
CMU/SEI-93-TR-6
ESC-TR-93-183

Taxonomy-Based Risk Identification

Marvin J. Carr
Suresh L. Konda
Ira Monarch
F. Carol Ulrich
Clay F. Walker

June 1993

Technical Report

CMU/SEI-93-TR-6
ESC-TR-93-183
June 1993

Taxonomy-Based Risk Identification

Marvin J. Carr
Suresh L. Konda
Ira Monarch
F. Carol Ulrich

Taxonomy Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Suite C201, Pittsburgh, PA 15212. Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8274 or toll-free in the U.S. — 1-800 225-3842).

Technical Report
CMU/SEI-93-TR-6
ESC-TR-93-183
June 1993

Taxonomy-Based Risk Identification

Marvin J. Carr
Suresh L. Konda
Ira Monarch
F. Carol Ulrich
Risk Taxonomy Project
and
Clay F. Walker
Resident Affiliate,
Computer Sciences Corporation

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212.
Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145. Phone: (703) 274-7633.

Document Discrepancy Report (DDR)

Please use this form to submit your comments on this document (CMU/SEI-93-TR-6). Indicate the page number, section number, and paragraph to which your comment pertains. You may send your comments electronically as ASCII text, but please use this format. Send your comments to: Risk Program, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213 or E-mail, drb@sei.cmu.edu

Date Submitted:	Date Responded:	Control No.:
Document Name & No.:		
Your Name:	Phone:	
Company:	FAX:	
Address:		
City:	State:	Zip:
Descrepancy: Page Paragraph		
Proposed Change:		
Disposition (SEI Use Only):		

Document Discrepancy Report (DDR)

Table of Contents

List of Figures

Taxonomy-Based Risk Identification

Abstract: This report describes a method for facilitating the systematic and repeatable identification of risks associated with the development of a software-dependent project. This method, derived from published literature and previous experience in developing software, was tested in active government-funded defense and civilian software development projects for both its usefulness and for improving the method itself. Results of the field tests encouraged the claim that the described method is useful, usable, and efficient. The report concludes with some macro-level lessons learned from the field tests and a brief overview of future work in establishing risk management on a firm footing in software development projects.

1 Introduction

The risk identification method presented here is the first step in a comprehensive and continuous risk management method being developed by the SEI to enhance the probability of project success. Subsequent technical reports will address the analysis, planning, tracking, and control aspects of risk management while, throughout, addressing the issue of risk communication.

The method is based upon the SEI taxonomy[1] of software development risks. The taxonomy provides a framework for organizing and studying the breadth of software development issues and hence provides a structure for surfacing and organizing software development risks.

The method described in this report consists of the taxonomy-based questionnaire (TBQ) instrument and a process for its application. The taxonomy organizes software development risks into 3 levels—class, element, and attribute. The TBQ consists of questions under each taxonomic attribute designed to elicit the range of risks and concerns potentially affecting the software product. The application process is designed such that the questionnaire can be used in a practical and efficient manner consistent with the objective of surfacing project risks. Both the TBQ and the application process have been developed using extensive expertise and field tests under a variety of conditions.

This report is organized as follows. Chapter 2 provides the context for interpreting the method and contains a brief background covering risk management in general and risk identification in particular. Chapter 3 contains descriptions of the software development taxonomy, the TBQ and its application in the context of field tests. Chapter 4 describes the technical approaches used in developing the risk identification method. Chapter 5 consists of the lessons learned from the use of the method. Chapter 6 concludes the report with some recommendations and

1. A taxonomy is a scheme that partitions a body of knowledge and defines the relationships among the pieces. It is used for classifying and understanding the body of knowledge. See *IEEE Software Engineering Standards Collection*, IEEE-STD-610.12, 1990.

briefly outlines future work in risk identification in particular and risk management in general. Finally, three appendices contain definitions of the software development taxonomy groups, the TBQ, and a glossary of terms.

2 The Context

The perspective taken in this work is that risks are inherent in any software development activity. Furthermore, risk taking is essential to progress, and failure is often a key part of learning. On the other hand, the inevitability of risks does not imply the inability to recognize and manage risks to minimize potential negative consequences while retaining the opportunities for creating new and better software.

Our work with both government and industry indicates only isolated cases of software risk management. Existing approaches to risk management tend to be ad hoc, undocumented, incomplete, and dependent on the experience and risk orientation of key project personnel. Furthermore, communication of software development risk is poor, incomplete, or even nonexistent. There are several reasons for this lack of communication, the most prevalent being reluctance on the part of customers, developers, management, and software personnel to accept that risk is neither bad nor good, but is always present, and can be effectively dealt with to increase the probability of project success. The very term, risk, often has negative connotations and the consequent cultural pressure to deny its existence thus exposing the project to unwanted surprises leading to difficulties and failures.[2]

On the positive side, most project risks are usually known by project personnel (though they might not use the term risk to describe them) and as a consequence can be surfaced and managed. Often project personnel experience risk as an uneasy feeling, a concern, or a doubt about some aspect of the software they are developing. However, because of communication problems described above, these risks are not brought to the attention of project management.

The SEI Risk Program is predicated on the assumption that a disciplined and systematic method of managing software development risk is necessary and feasible to control the quality, cost, and schedule of software products. Our approach, based on a stylized model of management problem solving, addresses the full range of activities to effectively manage software development risks and consists of methods, such as the one described in this report, suitable to each activity. This admittedly long-term effort will build upon, where feasible, the “good” practices of industry, academe, and experts. The SEI will develop methods to supplement and integrate existing capabilities.

The SEI risk management paradigm (see Figure 2-1) shows the different activities composing software development risk management. The paradigm is represented as a circle to emphasize that risk management is a continuous process while the arrows show the logical and temporal flow of information between the activities in risk management. Communication is placed in the center of the paradigm because it is both the conduit through which all information flows and, often, is the major obstacle to risk management. In essence, the paradigm is a framework

2. See Kirkpatrick, R. J.; Walker, J. A.; & Firth, R. “Software Development Risk Management: An SEI Appraisal,” *SEI Technical Review*, 1992.

for software risk management. From this framework, a project may structure a risk management practice best fitting into its project management structure. A brief summary of each risk management paradigm activity follows.

Identify. Before risks can be managed, they must be identified. Identification surfaces risks before they become problems and adversely affect a project. The SEI has developed techniques for surfacing risks by the application of a disciplined and systematic process that encourages project personnel to raise concerns and issues for subsequent analysis. One such technique, the taxonomy-based questionnaire, is described in subsequent chapters of this report.

Analyze. Analysis is the conversion of risk data into risk decision-making information. Analysis provides the basis for the project manager to work on the “right” risks.

Plan. Planning turns risk information into decisions and actions (both present and future). Planning involves developing actions to address individual risks, prioritizing risk actions, and creating an integrated risk management plan.The plan for a specific risk could take many forms. For example:

- Mitigate the impact of the risk by developing a contingency plan (along with an identified triggering event) should the risk occur.
- Avoid a risk by changing the product design or the development process.
- Accept the risk and take no further action, thus accepting the consequences if the risk occurs.
- Study the risk further to acquire more information and better determine the characteristics of the risk to enable decision making.

The key to risk action planning is to consider the future consequences of a decision made today.

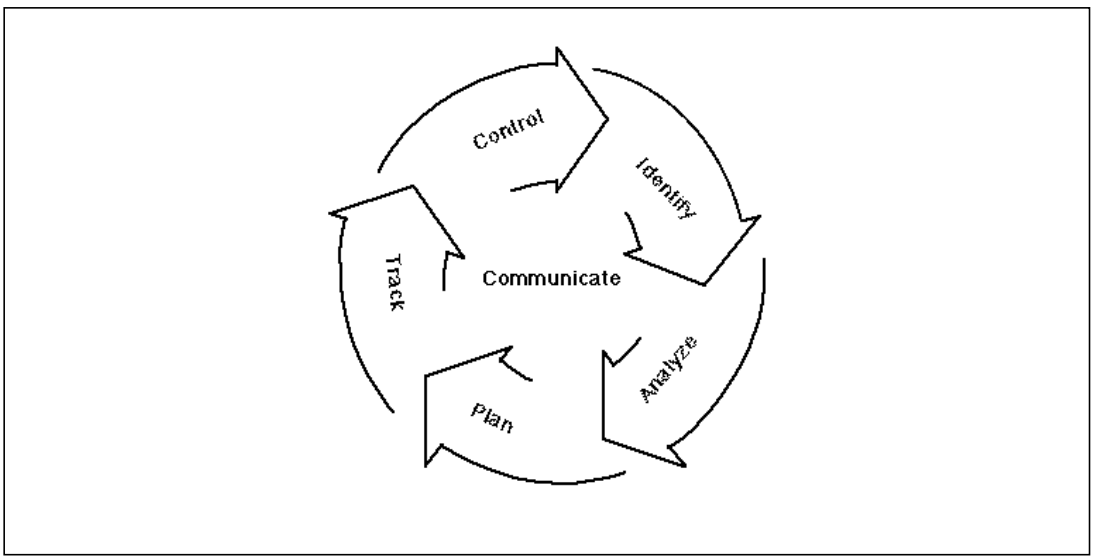

Figure 2-1 Risk Management Model

Track. Tracking consists of monitoring the status of risks and actions taken to ameliorate risks. Appropriate risk metrics are identified and monitored to enable the evaluation of the status of risks themselves and of risk mitigation plans. Tracking serves as the “watch dog” function of management.

Control. Risk control corrects for deviations from planned risk actions. Once risk metrics and triggering events have been chosen, there is nothing unique about risk control. Rather, risk control melds into project management and relies on project management processes to control risk action plans, correct for variations from plans, respond to triggering events, and improve risk management processes.

Communicate. Risk communication lies at the center of the model to emphasize both its pervasiveness and its criticality. Without effective communication, no risk management approach can be viable. While communication facilitates interaction among the elements of the model, there are higher level communications to consider as well. To be analyzed and managed correctly, risks must be communicated to and between the appropriate organizational levels and entities. This includes levels within the development project and organization, within the customer organization, and most especially, across that threshold between the developer, the customer, and, where different, the user. Because communication is pervasive, our approach is to address it as integral to every risk management activity and not as something performed outside of, and as a supplement to, other activities.

The remainder of this report focuses on risk identification and is based on the simple premise that without effective and repeatable risk identification methods, truly effective risk management is impossible; i.e., you can’t manage what you don’t know about. In keeping with this approach, the described identification method also begins to address the communication issue central to risk management.

3 The Risk Identification Method

The risks in a software development project can be known, unknown, or unknowable. Known risks are those that one or more project personnel are aware of—if not explicitly as risks, at least as concerns. The unknown risks are those that would be surfaced (i.e., become known) if project personnel were given the right opportunity, cues, and information. The unknowable risks are those that, even in principle, none could foresee. Hence these risks, while potentially critical to project success, are beyond the purview of any risk identification method.

Of the known risks some may already have been communicated to project management. The focus of the risk identification method described here is on risks that are known whether or not they have yet been communicated to project management, and on unknown risks.

The risk identification method achieves the desired focus through the interdependence of the TBQ instrument, and its application process. That is, the use of the one without the other would, in general, fail to reach the desired goals of surfacing and communicating risks to project management.

The SEI risk identification method is based on the following assumptions:

- Software development risks are generally known by the project's technical staff but are poorly communicated.
- A structured and repeatable method of risk identification is necessary for consistent risk management.
- Effective risk identification must cover all key development and support areas of the project.
- The risk identification process must create and sustain a non-judgmental and non-attributive risk elicitation environment so that tentative or controversial views are heard.
- No overall judgment can be made about the success or failure of a project based solely on the number or nature of risks uncovered.

The SEI taxonomy of software development maps the characteristics of software development and hence of software development risks. The TBQ consists of a list of non-judgmental questions to elicit issues and concerns (i.e., potential risks) and risks in each taxonomic group. Hence, the questionnaire ensures that all risk areas are systematically addressed, while the application process is designed to ensure that the questions are asked of the right people and in the right manner to produce optimum results.

The method described in this report presents a disciplined and systematic way to identify risk in a software-dependent system development. This method allows risks to be identified without justification and without a proposed solution. We believe this is the first step in establishing vital communication within an organization.

The TBQ application is semi-structured. The questions and their sequence are used as a defining but not as a limiting instrument. That is, the questions are asked in a given sequence, but the discussion is not restricted to that sequence. This is done to permit context- and culture-sensitive issues to arise in as "natural" a manner as possible. A completely structured interview, while arguably yielding more reliable data for subsequent analysis across different projects, may also yield less valid data.[3] Since for us the pragmatics of risk management were paramount, the semi-structured format was chosen. In effect, the TBQ method can be described as a form of structured brainstorming.

The risk identification method surfaces and clarifies the uncertainties and concerns of a project's technical and managerial staff. They are close to the problems at their level and have the experience and knowledge to recognize potential problems in technical, procedural, and contractual areas.

3.1 The Software Development Risk Taxonomy

Central to the risk identification method is the software development taxonomy. The taxonomy provides a framework for organizing and studying the breadth of software development issues. Hence, it serves as the basis for eliciting and organizing the full breadth of software development risks—both technical and non-technical. The taxonomy also provides a consistent framework for the development of other risk management methods and activities.

The software taxonomy is organized into three major *classes*.

1. **Product Engineering**. The technical aspects of the work to be accomplished.
2. **Development Environment**. The methods, procedures, and tools used to produce the product.
3. **Program Constraints**. The contractual, organizational, and operational factors within which the software is developed but which are generally outside of the direct control of the local management.

These taxonomic classes are further divided into *elements* and each element is characterized by its *attributes*.

Figure 3-1 contains a schematic of the taxonomy. The complete taxonomy, described from the software development risk perspective, is contained in Appendix A where Figure A-1 shows the detailed class-element-attribute structure of the taxonomy.

The next three subsections present a brief description of the software taxonomy to the class-element level.

---

3. See Suchman, L.; & Jordan, B. "Interactional Troubles in Face-to-Face Survey Interviews." *Journal of the American Statistical Association* 85, 409 (1990): 232-253.

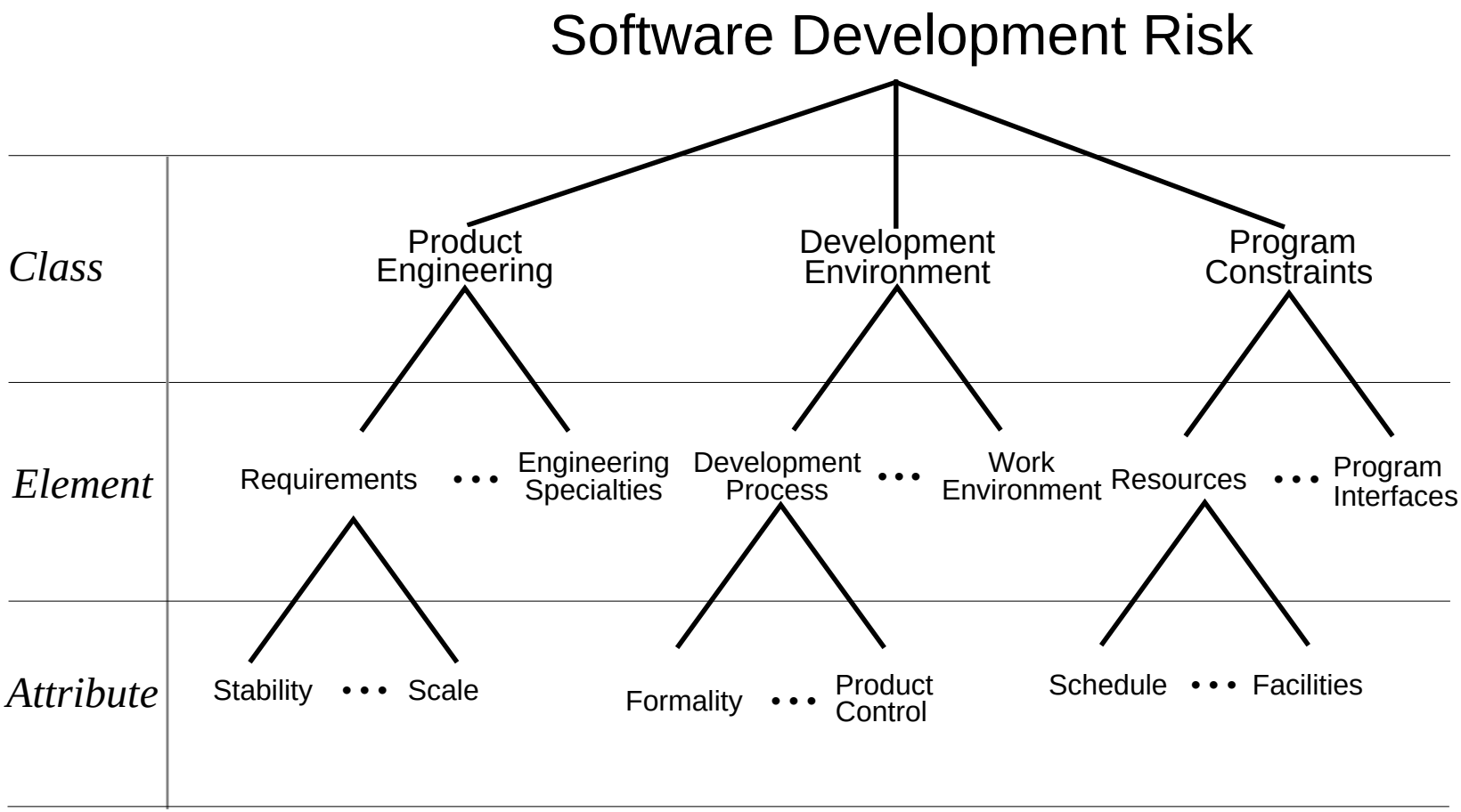

Figure 3-1 Software Development Risk Taxonomy

3.1.1 Product Engineering Class

The product engineering class consists of the intellectual and physical activities required to build the product to be delivered to the customer. It includes the complete system hardware, software, and documentation. The class focuses on the work to be performed, and includes the following elements:

- **Requirements**. The definition of what the software product is to do, the needs it must meet, how it is to behave, and how it will be used. This element also addresses the feasibility of developing the product and the scale of the effort.
- **Design**. The translation of requirements into an effective design within project and operational constraints.
- **Code and Unit Test.** The translation of software designs into code that satisfies the requirements allocated to individual units.
- **Integration and Test**. The integration of units into a working system and the validation that the software product performs as required.
- **Engineering Specialities**. Product requirements or development activities that may need specialized expertise such as safety, security, and reliability.

3.1.2 Development Environment Class

The development environment class is concerned with the project environment in which a software product is engineered. This environment consists of the following elements:

- **Development Process**. The definition, planning, documentation, suitability, enforcement, and communication of the methods and procedures used to develop the product.
- **Development System**. The tools and supporting equipment used in product development, such as computer-aided software engineering (CASE) tools, simulators, compilers, and host computer systems.
- **Management Process**. The planning, monitoring, and controlling of budgets and schedules; controlling factors involved in defining, implementing, and testing the product; the project manager's experience in software development, management, and the product domain; and the manager's expertise in dealing with external organizations including customers, senior management, matrix management, and other contractors.
- **Management Methods**. The methods, tools, and supporting equipment that will be used to manage and control the product development, such as monitoring tools, personnel management, quality assurance, and configuration management.
- **Work Environment**. The general environment within which the work will be performed, including the attitudes of people and the levels of cooperation, communication, and morale.

3.1.3 Program Constraints Class

The program constraints class consists of the “externals” of the project—the factors that are outside the direct control of the project but can still have major effects on its success. Program constraints include the following elements:

- **Resources**. The external constraints imposed on schedule, staff, budget, or facilities.
- **Contract**. The terms and conditions of the project contract.
- **Program Interfaces**. The external interfaces to customers, other contractors, corporate management, and vendors.

3.2 The Taxonomy-Based Questionnaire (TBQ)

The TBQ consists of questions at the attribute level along with specific cues and follow-up probe questions (the complete TBQ is reproduced in Appendix B). Because the TBQ is comprehensive, it contains questions that may not be relevant for all stages of a software development life cycle, for specific software domains, or for specific project organizations. Typically, the questionnaire is tailored to a particular project and its stage in the development life cycle by deleting questions not relevant to it. For example, a project without any subcontractors would have the subcontractor questions deleted.

To achieve clarity in the questions, many terms have been given specific definitions (see Appendix C). While these definitions are useful for understanding and using the instrument, they remain general to make the TBQ applicable to a wide range of projects.

Figure 3-2 contains an excerpt from the TBQ from the product engineering class, the requirements element, and performance attribute. The bracketed statement provides the questioner with a generalized context for the questions. Each “starter” question may have additional probe questions based on the initial response. These probe questions are prefaced by a parenthesized “Yes” or “No” indicating the type of initial response that activates the probe question. For instance, if the answer to the starter question “Has a performance analysis been done?” is “Yes,” then the probe questions “What is your level of confidence in the performance analysis?” and “Do you have a model to track performance through design and implementation?” are asked.

Implicit in the TBQ is the probing of areas that contain issues, concerns, or risks. That is, the interview protocol requires the interviewer to always follow up on responses that seem to indicate a potential problem (as described in the next chapter). For instance, if the response to question 23 were “No,” then the interviewer would probe for the issues, concerns, or risks to the project due to lack of performance analysis.

Questions and probes may also have “cues” (bulleted lists) associated with them that list areas that may be covered by the question. The cues are used after the participants have had the opportunity to respond freely. For instance, in question 22 in Figure 3-2, the probe “Are there

any problems with performance?" lists possible types of performance problems the software might exhibit—problems with throughput, scheduling asynchronous real-time events, and so forth.

3.3 Field Testing the TBQ

To validate the ability of the TBQ to elicit project risks, we tested the TBQ on a series of active projects within various companies. The lessons learned from each field test were incorporated into the next version of the TBQ to be tested again. This section describes the TBQ field test process.

In determining the TBQ field test process, the aim was to balance the needs of a thorough risk identification against the "costs" in terms of the time demands on both project personnel and the risk identification team. The method described in this section evolved from several field tests and represents our best understanding of how to use the TBQ under limited time constraints. The method described here is meant as a guide to organizations integrating TBQ risk identification into their risk management process. Implications for TBQ application drawn from information or experience obtained from these tests are presented in Chapter 5.

A. Product Engineering

2. Design

d. Performance
[Are there stringent response time or throughput requirements?]

[22] Are there any problems with performance?

- Throughput
- Scheduling asynchronous real-time events
- Real-time response
- Recovery timelines
- Response time
- Database response, contention, or access

[23] Has a performance analysis been done?

(Yes) (23.a) What is your level of confidence in the performance analysis?

(Yes) (23.b) Do you have a model to track performance through design and implementation?

Figure 3-2 Sample Taxonomy-Based Question

The TBQ field test process consists of four distinct activities: management commitment, team selection and training, risk identification, and identification conclusion. Figure 3-3 shows the process used in the field tests. Each of the steps in the process is described in greater detail below.

3.3.1 Management Commitment

Three activities need to take place before a field test can be conducted: executive commitment, project selection, and participant selection.

3.3.1.1 Executive Commitment

Our experience in field testing the TBQ showed that a lack of serious acceptance of the benefits of risk identification by project managers and their immediate superiors led to significant delay or even cancelation of the identification. Hence, our first step was to present an executive briefing to obtain executive commitment. This briefing gave an overview of the process with specifics as to personnel involved and cost to the project, as well as the benefits of conducting a risk identification.

3.3.1.2 Project Selection

Once executive commitment was obtained, the next step was the selection of a project for the field test. Two main criteria were found useful for selecting a project: that the project manager saw a benefit in doing a risk identification, and that the project had significant software content. Once a specific project was selected, it was found necessary that the client organization des-

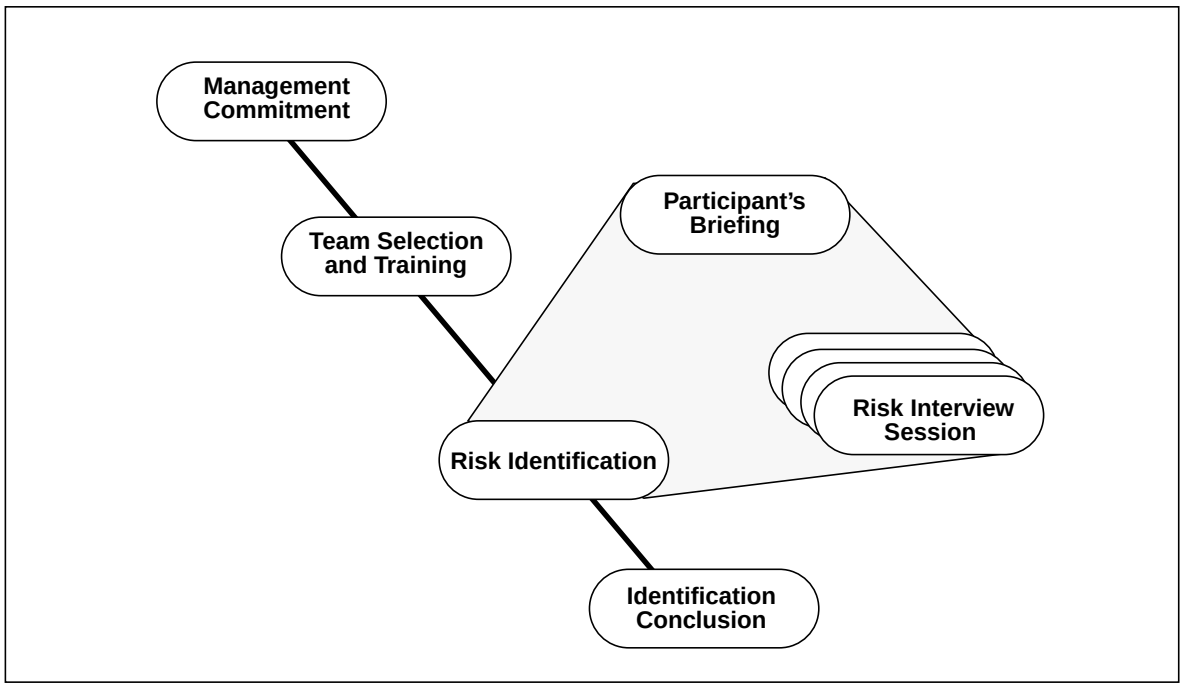

Figure 3-3 Risk Identification Process

ignate a site coordinator as the interface between the SEI identification team and the project being reviewed. The site coordinator was given the field test logistics with respect to participants, meeting space, schedule, and support and was responsible for the overall coordination of project personnel for the risk identification sessions.

3.3.1.3 Interview Participant Selection

The risk identification interview protocol requires an environment in which uncertainties, concerns, issues, and risks can be raised without fear of attribution or the need for a solution. This is essential so that critical risks, while possibly unavoidable, do not stay unarticulated because of the risk avoidance culture of the organization.[4] It was decided to achieve an environment of non-attribution by having each interview group consist only of peers. That is, those participating in a given interview were chosen such that there are no reporting relationships among them.

For a comprehensive risk identification it was also important to select participants who provided adequate vertical and horizontal project coverage. Vertical coverage refers to the project's organizational hierarchy, while horizontal coverage refers to the functional groups within or relevant to the project. Combining the need for adequate coverage and the pragmatic constraints of a field test, we found four interviews to be optimum. The four groups were technical leads, developers, support functions (SQA, Test, Integration, and so on), and project management. The project management group may be the project manager alone or, at his/her request, may include the chief technical or operations manager. In practice, the specific characteristics of the software project in question should guide the actual number of interview groups required for thorough risk identification. For several reasons (group dynamics, follow-up probing, observing and reacting to non-verbal responses, etc.), groups were limited to a maximum of five participants.

3.3.2 Team Selection and Training

To prepare for the transition of the TBQ into developing organizations, the SEI field test team was augmented with members from the client organization. Client team members selected to conduct the risk identification, in the collective, need a working knowledge of software development processes, the technology used on the project, the application domain, and the government contracting arena. In keeping with the need to have only peer relations within an interview, no reporting relationships should exist between team members and interview participants.

The entire team needed to be trained, which was done the day before the risk identification sessions. Training covered the TBQ, the roles, process, and interviewing protocol. The training also included a simulated risk identification exercise in which individuals on the team assume the various roles in an application. Role playing was used to give the team members practice in using the questionnaire and the facilitation methods for eliciting risks from partici-

4. See Kirkpatrick, R. J.; Walker, J. A.; & Firth, R. "Software Development Risk Management: An SEI Appraisal," *SEI Technical Review*, 1992.

pants. An integral part of the client team member training is observing experienced team members during actual interviews. This on-the-job training gives client team members a model to follow when conducting interviews. Client team members can then undertake critical roles during the course of the identification including that of an interviewer (generally after the second interview).

A key aspect of team training has been the provision of specific knowledge of the project under examination. This was provided to team members by a project briefing given by project personnel. It was found adequate if the project briefing was sufficiently detailed to acquaint team members with the characteristics, terminology, and organizational structure of the project.

3.3.3 Risk Identification

The risk identification session began with a briefing to all risk identification participants consisting of a description of the TBQ method and a summary of the schedule and process to be followed over the duration of the risk identification. In the interests of minimizing the overall time spent on identification, all participants selected by the project were asked to attend. Attendees may also include personnel who might take an active role in future risk identification or other risk management activities for the project.

The field test risk identification process consists of a series of interviews with the groups of selected project personnel. Each interview sessions has two parts:

1. **Question and Answer**. This segment involves the use of the TBQ and context-sensitive probe questions to elicit issues, concerns, or risks that might jeopardize the successful completion of the project.
2. **Issue Clarification**. This segment involves the clarification of the wording

is that many organizations are risk averse. That is, a notion exists that any project having a list of risks is in trouble and needs help fast. Leaving only a single copy of the identified issues and risks allows the project manager to retain control over the situation. This also gives the manager the opportunity to first distinguish among risks, concerns, and misinformation before circulating the results. Premature release of this information could needlessly force or hamper the manager's actions and expose the project to unjustified negative attention.

The process described above evolved over 15 field tests, undergoing significant refinement in each. We do not recommend the risk identification process described here as the *only* way to accomplish effective risk identification. We have, however, found it to be effective and efficient.

6. Specifically, the results were given to the manager as a list consisting of the issue statements and their taxonomy group classifications with no individual or interview group identification information.

4 Technical Approach

There is a dilemma in developing any method: on the one hand the method should be based on empirical data; on the other hand, the collection of any data is predicated upon a method. The problem was resolved by using the two-step procedure described below.

4.1 Taxonomy-Based Questionnaire Derivation

The taxonomy itself is based on three sources of information. The first is the published literature on software development and software risks.[7] The second, and primary, source is the rich experience of several SEI staff covering development and acquisition in the civilian and military sectors. The third source is the analysis of field data and some software literature using text processing tools to derive a terminological network for software development. This network, as an indicator of conceptual structure, was then evaluated and modified by software development technical experts.

The questions used to elicit risks from project personnel were developed based on both prior expertise in software development and the ongoing analysis of risks identified in field tests and in risk assessments carried out by the SEI (see Section 4.2). In keeping with the ideas detailed in Chapter 3, the questions were neutrally phrased, and additional follow-up probe questions and cues were added to assist both the interview participants and the interviewer.

4.2 Evolutionary Development

To improve and test the method, a series of field tests was conducted on active software development projects. The information obtained from these tests was used in a tight feedback loop to change any part of the method—the taxonomy, the questions, and the application process—prior to the next field test. In essence, the risk identification method was developed using an “evolutionary development” process.

In each field test, a few client personnel selected by the client were trained to be part of the risk identification team. Their roles were either passive (as observers only) or active (as full-fledged members of the team). Their involvement not only provided additional viewpoints into the questionnaire and the application, but also served as a test of the applicability and transferability of the method outside the SEI.

Client participant and client team member feedback and critiques were a part of each field test. These critiques were particularly useful in making modifications to the application process. They were also useful in clarifying the meaning of various terms used and in suggesting additional questions and taxonomic question groups.

---

7. See Boehm, B. “Software Risk Management: Principles and Practices.” *IEEE Software* (January 1990): 32-41; Charette, R. N. *Application Strategies for Risk Analysis*. New York: McGraw Hill, 1990; Air Force Systems Command, *Software Risk Management,* AFSCP/AFLCP 800-45, 1987; and the *IEEE Standard Glossary for Software Engineering,* IEEE-STD-610.12, 1990.

Field notes on the interview process and discussions, and surfaced issues and risks were analyzed using expert judgment and computational tools. The results of this analysis were also used to improve the instrument and change the application process for the next field test.

In all, 15 field tests were conducted in 11 organizations. Of these projects, 1 was at precontract award,[8] 1 in requirements analysis, 1 at alpha testing, 2 in the maintenance phase, and the rest somewhere between preliminary design review and critical design review. Domain coverage of the projects included C^3I (command, control, communication, and intelligence), satellite, civilian C^2 (command and control), and information systems.

8. The TBQ was modified in this case. Some questions were removed as being obviously inapplicable at this stage, and some of the remainder were altered in the verb tense used. Given the size and characteristics of the proposal team, only two interviews were needed to achieve the desired coverage.

5 Lessons Learned

The field tests exposed the risk identification method to a broad spectrum of companies covering several application domains, development methods, and risk management techniques. During the field tests, several observations were made that may help organizations to install more proactive and suitable risk management processes. These observations are summarized below.

5.1 The Method Is Effective and Efficient

The taxonomy-based method has proven effective and efficient in surfacing both acknowledged and previously unacknowledged risks in a wide range of domains and across the life cycle of software development. According to feedback from participants, especially project management, it appeared that all known and communicated project risks were surfaced by the method. In addition, previously unknown issues were raised, which surprised project management.

Some projects had been previously subjected to other risk identification methods. Feedback from these projects was that the other methods took more time and produced less interesting results or were thought to be less effective in surfacing all issues, concerns, and risks. All project managers and most participants felt that the time involved in this risk identification method was well spent. Some projects continue to work actively with the SEI, and a few are beginning to use the TBQ method internally.

5.2 Process Is Important

The interview group size of five is about the maximum feasible. Any additional coverage of project personnel should be achieved by increasing the number of interviews and not the number of participants in an interview.

It is particularly important that each interview group be composed of peers only. Even in organizations where staff across the project hierarchy felt that they had open and honest communications, the presence of a reporting relationship in an interview group ***always*** had an inhibiting effect on the subordinates. Furthermore, there should be no reporting relationship between any member of the risk identification team and those being interviewed.

The interview protocol, which is primarily based on allowing the discussion to proceed naturally through the use of non-judgmental questions and probes, is also important. It is critical that the interview team exhibit a neutral, non-evaluative posture. However, the interviewer needs to guide the discussion to ensure that it does not get bogged down, deviate from the task at hand—risk identification—and move into problem solving, for instance, which is a natural tendency for such exercises.

Additionally, maintenance of the proper interview schedule is important to ensure the formality of the activity. This also enables interview participants to manage their calendars so they can be left entirely undisturbed during the interview itself. An interruption of an interview has far greater impact than one might suppose and should be avoided as much as possible.

Field test interview sessions were artificially constrained to two-and-a-half hours.[9] Thus interviews often did not cover the entire TBQ. To ensure coverage of the TBQ over the risk identification, the starting point was tailored to the specific group being interviewed.[10] This coupled with the free flow of discussions that is at the core of the interview protocol, surfaced risks across taxonomic groups. There was near consensus among participants that a two-and-a-half hour interview was an upper limit. Additional time required for fuller coverage of the TBQ should be scheduled as a separate session.

While issue identification by participants was effective, revisiting issues after the actual question-and-answer session was important. The current approach of issue clarification via issue classification and equivalence identification was effective. In addition, issue clarification also facilitated clearer communication of risks and prepared the way for further analysis.

5.3 Facilitation Skills Can Be Transitioned

The field tests showed that individuals can be effectively trained to conduct a risk identification using the taxonomy. Furthermore, these individuals, while software and domain knowledgeable, need not be experts in the areas and, finally, need not have detailed project-specific knowledge to conduct a risk identification. However, it is most helpful if these individuals have some prior training and experience in facilitation skills. In any event, all team members found that observing an experienced team member during a field test interview session was a crucial part of the training.[11]

9. Recall that the primary purpose of the field test was to test the TBQ and its application, not to test their use in a full-fledged risk identification.

10. Developers started with the product engineering class, specialty engineers with the development environment class, and the project manager with the program constraints class.

11. That is, on-the-job training was necessary to effectively conduct an interview.

6 Conclusions

We have described a method for facilitating the systematic and repeatable identification of risks associated with of a software-dependent development project. This method, while originally derived from some published literature and previous experience in developing software, was tested in active software development projects for both its usefulness and for improving the method itself. While considerable work remains to be done in developing additional methods not only for risk identification but also for risk analysis, planning, tracking, and control, results of the field tests encourage the claim that the described method is useful, usable, and efficient. Below are some recommendations and a brief overview of future work needed for establishing risk management on a firm footing in software development projects.

Project risks change over time in both characteristics (probability, impact, time frame) and content—i.e., the risk of yesterday could be the problem of today or cease to be a risk altogether, and new risks could arise. Consequently, the project should repeat the risk identification and follow-up processes periodically during the project life cycle.

For acquiring organizations, we recommend that formal risk identification be conducted during the concept definition phase of an acquisition to determine potential project risks. The knowledge gained could then be used to directly reduce acquisition risk by either removing or mitigating high-risk items in the acquisition, or by evaluating the contractor proposals for dealing with the risks.

For developer organizations, we currently recommend that formal risk identification should be conducted during the proposal phase and at major project milestones thereafter. Informal risk identification should be scheduled and performed more frequently. In all cases, the contractor should report to the customer via existing management reporting channels those risks that pose significant danger to the project.

This report describes the first of a series of methods for risk management. Integration of the methods into a comprehensive risk management program is a long-term goal of the SEI. In the immediate future, the SEI will be exploring methods to analyze the issues and risks that surfaced in risk identification to judge their validity and relative criticality. This will be done using both qualitative and quantitative approaches. Next, to facilitate action planning, we will work on approaches to help classify risks in terms of their sources, their consequences, and most important, their inter-relationships.

The SEI will continue to update the TBQ regularly, based on changes requested by the user community or indicated by the analysis of data captured from actual usage in projects. These changes will be controlled with a formal configuration management process using the Document Discrepancy Report included in this report.

To enhance and accelerate the acceptance of the SEI risk identification method by all relevant parties in the software development industry, working and special interest groups will be organized with the help of industry groups, e.g., software engineering process groups (SEPGs) and consortia, e.g., software process improvement network (SPIN). These groups will also be active participants in the further evolution of the taxonomy-based risk identification method.

References

[AirForce 87] Air Force Systems Command, *Software Risk Management*. AFSCP/AFLCP 800-45, 1987.

[Boehm 90] Boehm, B. “Software Risk Management: Principles and Practices.” *IEEE Software* (January 1990): 32-41.

[Charette 90] Charette, R. N. *Application Strategies for Risk Analysis*. New York: McGraw Hill, 1990.

[IEEE 91] *IEEE Software Engineering Standards Collection*, Spring 1991.

[IEEE 90] *IEEE Standard Glossary for Software Engineering,* IEEE-STD-610.12, 1990.

[Kirkpatrick 92] Kirkpatrick, R. J.; Walker, J. A.; & Firth, R. “Software Development Risk Management: An SEI Appraisal,” *SEI Technical Review*, Pittsburgh, Pa.: 1992.

[Suchman 90] Suchman, L.; & Jordan, B. “Interactional Troubles in Face-to-Face Survey Interviews.” *Journal of the American Statistical Association 85*, 409 (1990): 232-253.

Appendix A Taxonomic Group Definitions

This appendix provides the definitions of the taxonomic groups in the class, element, and attribute categories of the Software Development Risk Taxonomy. An overview of the taxonomy groups and their hierarchical organization is provided in Figure A-1.

The taxonomy might be used to classify many different factors associated with the development of software-dependent systems such as development tasks, quality procedures, or sources or consequences of risk. However, the definitions as presented here are designed to facilitate classification of the risks themselves, as associated with the development process.

A. Product Engineering

1. Requirements
   a. Stability
   b. Completeness
   c. Clarity
   d. Validity
   e. Feasibility
   f. Precedent
   g. Scale
2. Design
   a. Functionality
   b. Difficulty
   c. Interfaces
   d. Performance
   e. Testability
   f. Hardware Constraints
   g. Non-Developmental Software
3. Code and Unit Test
   a. Feasibility
   b. Testing
   c. Coding/Implement-ation
4. Integration and Test
   a. Environment
   b. Product
   c. System
5. Engineering Specialties
   a. Maintainability
   b. Reliability
   c. Safety
   d. Security
   e. Human Factors
   f. Specifications

B. Development Environment

1. Development Process
   a. Formality
   b. Suitability
   c. Process Control
   d. Familiarity
   e. Product Control
2. Development System
   a. Capacity
   b. Suitability
   c. Usability
   d. Familiarity
   e. Reliability
   f. System Support
   g. Deliverability
3. Management Process
   a. Planning
   b. Project Organization
   c. Management Experience
   d. Program Interfaces
4. Management Methods
   a. Monitoring
   b. Personnel Management
   c. Quality Assurance
   d. Configuration Management
5. Work Environment
   a. Quality Attitude
   b. Cooperation
   c. Communication
   d. Morale

C. Program Constraints

1. Resources
   a. Schedule
   b. Staff
   c. Budget
   d. Facilities
2. Contract
   a. Type of Contract
   b. Restrictions
   c. Dependencies
3. Program Interfaces
   a. Customer
   b. Associate Contractors
   c. Subcontractors
   d. Prime Contractor
   e. Corporate Management
   f. Vendors
   g. Politics

Figure A-1 Taxonomy of Software Development Risks

A. Product Engineering

Product engineering refers to the system engineering and software engineering activities involved in creating a system that satisfies specified requirements and customer expectations. These activities include system and software requirements analysis and specification, software design and implementation, integration of hardware and software components, and software and system test.

The elements of this class cover traditional software engineering activities. They comprise those technical factors associated with the deliverable product itself, independent of the processes or tools used to produce it or the constraints imposed by finite resources or external factors beyond program control.

Product engineering risks generally result from requirements that are technically difficult or impossible to implement, often in combination with inability to negotiate relaxed requirements or revised budgets and schedules; from inadequate analysis of requirements or design specification; or from poor quality design or coding specifications.

1. Requirements

Attributes of the requirements element cover both the quality of the requirements specification and also the difficulty of implementing a system that satisfies the requirements.

The following attributes characterize the requirements element.

a) Stability

The stability attribute refers to the degree to which the requirements are changing and the possible effect changing requirements and external interfaces will have on the quality, functionality, schedule, design, integration, and testing of the product being built.

The attribute also includes issues that arise from the inability to control rapidly changing requirements. For example, impact analyses may be inaccurate because it is impossible to define the baseline against which the changes will be implemented.

b) Completeness

Missing or incompletely specified requirements may appear in many forms, such as a requirements document with many functions or parameters “to be defined”; requirements that are not specified adequately to develop acceptance criteria, or inadvertently omitted requirements. When missing information is not supplied in a timely manner, implementation may be based on contractor assumptions that differ from customer expectations.

When customer expectations are not documented in the specification, they are not budgeted into the cost and schedule.

c) Clarity

This attribute refers to ambiguously or imprecisely written individual requirements that are not resolved until late in the development phase. This lack of a mutual contractor and customer understanding may require re-work to meet the customer intent for a requirement.

d) Validity

This attribute refers to whether the aggregate requirements reflect customer intentions for the product. This may be affected by misunderstandings of the written requirements by the contractor or customer, unwritten customer expectations or requirements, or a specification in which the end user did not have inputs.

This attribute is affected by the completeness and clarity attributes of the requirements specifications, but refers to the larger question of the system as a whole meeting customer intent.

e) Feasibility

The feasibility attribute refers to the difficulty of implementing a single technical or operational requirement, or of simultaneously meeting conflicting requirements. Sometimes two requirements by themselves are feasible, but together are not; they cannot both exist in the same product at the same time.

Also included is the ability to determine an adequate qualification method for demonstration that the system satisfies the requirement.

f) Precedent

The precedent attribute concerns capabilities that have not been successfully implemented in any existing systems or are beyond the experience of program personnel or of the company. The degree of risk depends on allocation of additional schedule and budget to determine the feasibility of their implementation; contingency plans in case the requirements are not feasible as stated; and flexibility in the contract to allocate implementation budget and schedule based on the outcome of the feasibility study.

Even when unprecedented requirements are feasible, there may still be a risk of underestimating the difficulty of implementation and committing to an inadequate budget and schedule.

g) Scale

This attribute covers both technical and management challenges presented by large complex systems development.

Technical challenges include satisfaction of timing, scheduling and response requirements, communication among processors, complexity of system integration, analysis of inter-component dependencies, and impact due to changes in requirements.

Management of a large number of tasks and people introduces a complexity in such areas as project organization, delegation of responsibilities, communication among management and peers, and configuration management.

2. Design

The attributes of the design element cover the design and feasibility of algorithms, functions or performance requirements, and internal and external product interfaces. Difficulty in testing may begin here with failure to work to testable requirements or to include test features in the design.The following attributes characterize the design element.

a) Functionality

This attribute covers functional requirements that may not submit to a feasible design, or use of specified algorithms or designs without a high degree of certainty that they will satisfy their source requirements. Algorithm and design studies may not have used appropriate investigation techniques or may show marginal feasibility.

b) Difficulty

The difficulty attribute refers to functional or design requirements that may be extremely difficult to realize. Systems engineering may design a system architecture difficult to implement, or requirements analysis may have been based on optimistic design assumptions.

The difficulty attribute differs from design feasibility in that it does not proceed from pre-ordained algorithms or designs.

c) Interfaces

This attribute covers all hardware and software interfaces that are within the scope of the development program, including interfaces between configuration items, and the techniques for defining and managing the interfaces. Special note is taken of non-developmental software and developmental hardware interfaces.

d) Performance

The performance attribute refers to time-critical performance: user and real-time response requirements, throughput requirements, performance analyses, and performance modeling throughout the development cycle.

e) Testability

The testability attribute covers the amenability of the design to testing, design of features to facilitate testing, and the inclusion in the design process of people who will design and conduct product tests.

f) Hardware Constraints

This attribute covers target hardware with respect to system and processor architecture, and the dependence on hardware to meet system and software performance requirements. These constraints may include throughput or memory speeds, real-time response capability, database access or capacity limitations, insufficient reliability, unsuitability to system function, or insufficiency in the amount of specified hardware.

g) Non-Developmental Software

Since non-developmental software (NDS) is not designed to system requirements, but selected as a “best fit,” it may not conform precisely to performance, operability or supportability requirements.

The customer may not accept vendor or developer test and reliability data to demonstrate satisfaction of the requirements allocated to NDS. It may then be difficult to produce this data to satisfy acceptance criteria and within the estimated NDS test budget.

Requirements changes may necessitate re-engineering or reliance on vendors for special purpose upgrades.

3. Code and Unit Test

Attributes of this element are associated with the quality and stability of software or interface specifications, and constraints that may present implementation or test difficulties.

a) Feasibility

The feasibility attribute of the code and unit test element addresses possible difficulties that may arise from poor design or design specification or from inherently difficult implementation needs.

For example, the design may not have quality attributes such as module cohesiveness or interface minimization; the size of the modules may contribute complexity; the design may not be specified in sufficient detail, requiring the programmer to make assumptions or design decisions during coding; or the design and interface specifications may be changing, perhaps without an approved detailed design baseline; and the use of developmental hardware may make an additional contribution to inadequate or unstable interface specification. Or, the nature of the system itself may aggravate the difficulty and complexity of the coding task.

b) Unit Test

Factors affecting unit test include planning and preparation and also the resources and time allocated for test.

Constituents of these factors are: entering unit test with quality code obtained from formal or informal code inspection or verification procedures; pre-planned test cases that have been verified to test unit requirements; a test bed consisting of the necessary hardware or emulators, and software or simulators; test data to satisfy the planned test; and sufficient schedule to plan and carry out the test plan.

c) Coding/Implementation

This attribute addresses the implications of implementation constraints. Some of these are: target hardware that is marginal or inadequate with regard to speed, architecture, memory size or external storage capacity; required implementation languages or methods; or differences between the development and target hardware.

4. Integration and Test

This element covers integration and test planning, execution, and facilities for both the contractual product and for the integration of the product into the system or site environment.

a) Environment

The integration and test environment includes the hardware and software support facilities and adequate test cases reflecting realistic operational scenarios and realistic test data and conditions.

This attribute addresses the adequacy of this environment to enable integration in a realistic environment or to fully test all functional and performance requirements.

b) Product

The product integration attribute refers to integration of the software components to each other and to the target hardware, and testing of the contractually deliverable product. Factors that may affect this are internal interface specifications for either hardware or software, testability of requirements, negotiation of customer agreement on test criteria, adequacy of test specifications, and sufficiency of time for integration and test.

c) System

The system integration attribute refers to integration of the contractual product to interfacing systems or sites. Factors associated with this attribute are external interface specifications, ability to faithfully produce system interface conditions prior to site or system integration, access to the system or site being interfaced to, adequacy of time for testing, and associate contractor relationships.

5. Engineering Specialities

The engineering specialty requirements are treated separately from the general requirements element primarily because they are often addressed by specialists who may not be full time on the program. This taxonomic separation is a device to ensure that these specialists are called in to analyze the risks associated with their areas of expertise.

a) Maintainability

Maintainability may be impaired by poor software architecture, design, code, or documentation resulting from undefined or un-enforced standards, or from neglecting to analyze the system from a maintenance point of view.

b) Reliability

System reliability or availability requirements may be affected by hardware not meeting its reliability specifications or system complexity that aggravates difficulties in meeting recovery timelines. Reliability or availability requirements allocated to software may be stated in absolute terms, rather than as separable from hardware and independently testable.

c) Safety

This attribute addresses the difficulty of implementing allocated safety requirements and also the potential difficulty of demonstrating satisfaction of requirements by faithful simulation of the unsafe conditions and corrective actions. Full demonstration may not be possible until the system is installed and operational.

d) Security

This attribute addresses lack of experience in implementing the required level of system security that may result in underestimation of the effort required for rigorous verification methods, certification and accreditation, and secure or trusted development process logistics; developing to unprecedented requirements; and dependencies on delivery of certified hardware or software.

e) Human Factors

Meeting human factors requirements is dependent on understanding the operational environment of the installed system and agreement with various customer and user factions on a mutual understanding of the expectations embodied in the human factors requirements. It is difficult to convey this understanding in a written specification. Mutual agreement on the human interface may require continuous prototyping and demonstration to various customer factions.

f) Specifications

This attribute addresses specifications for the system, hardware, software, interface, or test requirements or design at any level with respect to feasibility of implementation and the quality attributes of stability, completeness, clarity, and verifiability.

B. Development Environment

The development environment class addresses the project environment and the process used to engineer a software product. This environment includes the development process and system, management methods, and work environment. These environmental elements are characterized below by their component attributes.

1. Development Process

The development process element refers to the process by which the contractor proposes to satisfy the customer's requirements. The process is the sequence of steps—the inputs, outputs, actions, validation criteria, and monitoring activities—leading from the initial requirement specification to the final delivered product. The development process includes such phases as requirements analysis, product definition, product creation, testing, and delivery. It includes both general management processes such as costing, schedule tracking, and personnel assignment, and also project-specific processes such as feasibility studies, design reviews, and regression testing.

This element groups risks that result from a development process that is inadequately planned, defined and documented; that is not suited to the activities necessary to accomplish the project goals; and that is poorly communicated to the staff and lacks enforced usage.

a) Formality

Formality of the development process is a function of the degree to which a consistent process is defined, documented, and communicated for all aspects and phases of the development.

b) Suitability

Suitability refers to the adequacy with which the selected development model, process, methods, and tools support the scope and type of activities required for the specific program.

c) Process Control

Process control refers not only to ensuring usage of the defined process by program personnel, but also to the measurement and improvement of the process based on observation with respect to quality and productivity goals. Control may be complicated due to distributed development sites.

d) Familiarity

Familiarity with the development process covers knowledge of, experience in, and comfort with the prescribed process.

e) Product Control

Product control is dependent on traceability of requirements from the source specification through implementation such that the product test will demonstrate the source requirements. The change control process makes use of the traceability mechanism in impact analyses and reflects all resultant document modifications including interface and test documentation.

2. Development System

The development system element addresses the hardware and software tools and supporting equipment used in product development. This includes computer aided software engineering tools, simulators, compilers, test equipment, and host computer systems.

a) Capacity

Risks associated with the capacity of the development system may result from too few workstations, insufficient processing power or database storage, or other inadequacies in equipment to support parallel activities for development, test, and support activities.

b) Suitability

Suitability of the development system is associated with the degree to which it is supportive of the specific development models, processes, methods, procedures, and activities required and selected for the program. This includes the development, management, documentation, and configuration management processes.

c) Usability

Usability refers to development system documentation, accessibility and workspace, as well as ease of use.

d) Familiarity

Development system familiarity depends on prior use of the system by the company and by project personnel as well as adequate training for new users.

e) Reliability

Development system reliability is a measure of whether the needed components of the development system are available and working properly whenever required by any program personnel.

f) System Support

Development system support involves training in use of the system, access to expert users or consultants, and repair or resolution of problems by vendors.

g) Deliverability

Some contracts require delivery of the development system. Risks may result from neglecting to bid and allocate resources to ensure that the development system meets all deliverable requirements.

3. Management Process

The management process element pertains to risks associated with planning, monitoring, and controlling budget and schedule; with controlling factors involved in defining, implementing, and testing the product; with managing project personnel; and with handling external organizations including the customer, senior management, matrix management, and other contractors.

a) Planning

The planning attribute addresses risks associated with developing a well-defined plan that is responsive to contingencies as well as long-range goals and that was formulated with the input and acquiescence of those affected by it. Also addressed are managing according to the plan and formally modifying the plan when changes are necessary.

b) Project Organization

This attribute addresses the effectiveness of the program organization, the effective definition of roles and responsibilities, and the assurance that these roles and lines of authority are understood by program personnel.

c) Management Experience

This attribute refers to the experience of all levels of managers with respect to management, software development management, the application domain, the scale and complexity of the system and program, the selected development process, and hands-on development of software.

d) Program Interfaces

This attribute refers to the interactions of managers at all levels with program personnel at all levels, and with external personnel such as the customer, senior management, and peer managers.

4. Management Methods

This element refers to methods for managing both the development of the product and program personnel. These include quality assurance, configuration management, staff development with respect to program needs, and maintaining communication about program status and needs.

a) Monitoring

The monitoring includes the activities of obtaining and acting upon status reports, allocating status information to the appropriate program organizations, and maintaining and using progress metrics.

b) Personnel Management

Personnel management refers to selection and training of program members and ensuring that they: take part in planning and customer interaction for their areas of responsibility; work according to plan; and receive the help they need or ask for to carry out their responsibilities.

c) Quality Assurance

The quality assurance attribute refers to the procedures instituted for ensuring both that contractual processes and standards are implemented properly for all program activities, and that the quality assurance function is adequately staffed to perform its duties.

d) Configuration Management

The configuration management (CM) attribute addresses both staffing and tools for the CM function as well as the complexity of the required CM process with respect to such factors as multiple development and installation sites and product coordination with existing, possibly changing, systems.

5. Work Environment

The work environment element refers to subjective aspects of the environment such as the amount of care given to ensuring that people are kept informed of program goals and information, the way people work together, responsiveness to staff inputs, and the attitude and morale of the program personnel.

a) Quality Attitude

This attribute refers to the tendency of program personnel to do quality work in general and to conform to specific quality standards for the program and product.

b) Cooperation

The cooperation attribute addresses lack of team spirit among development staff both within and across work groups and the failure of all management levels to demonstrate that best efforts are being made to remove barriers to efficient accomplishment of work.

c) Communication

Risks that result from poor communication are due to lack of knowledge of the system mission, requirements, and design goals and methods, or to lack of information about the importance of program goals to the company or the project.

d) Morale

Risks that result from low morale range across low levels of enthusiasm and thus low performance, productivity or creativity; anger that may result in intentional damage to the project or the product; mass exodus of staff from the project; and a reputation within the company that makes it difficult to recruit.

C. Program Constraints

Program constraints refer to the “externals” of the project. These are factors that may be outside the control of the project but can still have major effects on its success or constitute sources of substantial risk.

1. Resources

This element addresses resources for which the program is dependent on factors outside program control to obtain and maintain. These include schedule, staff, budget, and facilities.

a) Schedule

This attribute refers to the stability of the schedule with respect to internal and external events or dependencies and the viability of estimates and planning for all phases and aspects of the program.

b) Staff

This attribute refers to the stability and adequacy of the staff in terms of numbers and skill levels, their experience and skills in the required technical areas and application domain, and their availability when needed.

c) Budget

This attribute refers to the stability of the budget with respect to internal and external events or dependencies and the viability of estimates and planning for all phases and aspects of the program.

d) Facilities

This attribute refers to the adequacy of the program facilities for development, integration, and testing of the product.

2. Contract

Risks associated with the program contract are classified according to contract type, restrictions, and dependencies.

a) Type of Contract

This attribute covers the payment terms (cost plus award fee, cost plus fixed fee, etc.) and the contractual requirements associated with such items as the Statement of Work, Contract Data Requirements List, and the amount and conditions of customer involvement.

b) Restrictions

Contract restrictions and restraints refer to contractual directives to, for example, use specific development methods or equipment and the resultant complications such as acquisition of data rights for use of non-developmental software.

c) Dependencies

This attribute refers to the possible contractual dependencies on outside contractors or vendors, customer-furnished equipment or software, or other outside products and services.

3. Program Interfaces

This element consists of the various interfaces with entities and organizations outside the development program itself.

a) Customer

The customer attribute refers to the customer's level of skill and experience in the technical or application domain of the program as well as difficult working relationships or poor mechanisms for attaining customer agreement and approvals, not having access to certain customer factions, or not being able to communicate with the customer in a forthright manner.

b) Associate Contractors

The presence of associate contractors may introduce risks due to conflicting political agendas, problems of interfaces to systems being developed by outside organizations, or lack of cooperation in coordinating schedules and configuration changes.

c) Subcontractors

The presence of subcontractors may introduce risks due to inadequate task definitions and subcontractor management mechanisms, or to not transferring subcontractor technology and knowledge to the program or corporation.

d) Prime Contractor

When the program is a subcontract, risks may arise from poorly defined task definitions, complex reporting arrangements, or dependencies on technical or programmatic information.

e) Corporate Management

Risks in the corporate management area include poor communication and direction from senior management as well as non-optimum levels of support.

f) Vendors

Vendor risks may present themselves in the forms of dependencies on deliveries and support for critical system components.

g) Politics

Political risks may accrue from relationships with the company, customer, associate contractors or subcontractors, and may affect technical decisions.

Appendix B Taxonomy-Based Questionnaire

A. Product Engineering

1. Requirements

a. Stability

[Are requirements changing even as the product is being produced?]

[1] Are the requirements stable?

(No) (1.a) What is the effect on the system?

- Quality
- Functionality
- Schedule
- Integration
- Design
- Testing

[2] Are the external interfaces changing?

b. Completeness

[Are requirements missing or incompletely specified?]

[3] Are there any TBDs in the specifications?

[4] Are there requirements you know should be in the specification but aren't?

(Yes) (4.a) Will you be able to get these requirements into the system?

[5] Does the customer have unwritten requirements/expectations?

(Yes) (5.a) Is there a way to capture these requirements?

[6] Are the external interfaces completely defined?

c. Clarity

[Are requirements unclear or in need of interpretation?]

[7] Are you able to understand the requirements as written?

(No) (7.a) Are the ambiguities being resolved satisfactorily?

(Yes) (7.b) There are no ambiguities or problems of interpretation?

d. Validity

[Will the requirements lead to the product the customer has in mind?]

[8] Are there any requirements that may not specify what the customer really wants?

(Yes) (8.a) How are you resolving this?

[9] Do you and the customer understand the same thing by the requirements?

(Yes) (9.a) Is there a process by which to determine this?

[10] How do you validate the requirements?

- Prototyping
- Analysis
- Simulations

e. Feasibility

[Are requirements infeasible from an analytical point of view?]

[11] Are there any requirements that are technically difficult to implement?

(Yes) (11.a) What are they?

(Yes) (11.b) Why are they difficult to implement?

(No) (11.c) Were feasibility studies done for these requirements?

(Yes) (11.c.1) How confident are you of the assumptions made in the studies?

f. Precedent

[Do requirements specify something never done before, or that your company has not done before?]

[12] Are there any state-of-the-art requirements?

- Technologies
- Methods
- Languages
- Hardware

(No) (12.a) Are any of these new to you?

(Yes) (12.b) Does the program have sufficient knowledge in these areas?

(No) (12.b.1) Is there a plan for acquiring knowledge in these areas?

g. Scale

[Do requirements specify a product larger, more complex, or requiring a larger organization than in the experience of the company?]

[13] Is the system size and complexity a concern?

(No) (13.a) Have you done something of this size and complexity before?

[14] Does the size require a larger organization than usual for your company?

2. Design

a. Functionality

[Are there any potential problems in meeting functionality requirements?]

[15] Are there any specified algorithms that may not satisfy the requirements?

(No) (15.a) Are any of the algorithms or designs marginal with respect to meeting requirements?

[16] How do you determine the feasibility of algorithms and designs?

- Prototyping
- Modeling
- Analysis
- Simulation

b. Difficulty

[Will the design and/or implementation be difficult to achieve?]

[17] Does any of the design depend on unrealistic or optimistic assumptions?

[18] Are there any requirements or functions that are difficult to design?

(No) (18.a) Do you have solutions for all the requirements?

(Yes) (18.b) What are the requirements?

- Why are they difficult?

c. Interfaces

[Are the internal interfaces (hardware and software) well defined and controlled?]

[19] Are the internal interfaces well defined?

- Software-to-software
- Software-to-hardware

[20] Is there a process for defining internal interfaces?

(Yes) (20.a) Is there a change control process for internal interfaces?

[21] Is hardware being developed in parallel with software?

(Yes) (21.a) Are the hardware specifications changing?

(Yes) (21.b) Have all the interfaces to software been defined?

(Yes) (21.c) Will there be engineering design models that can be used to test the software?

d. Performance

[Are there stringent response time or throughput requirements?]

[22] Are there any problems with performance?

- Throughput
- Scheduling asynchronous real-time events
- Real-time response
- Recovery timelines
- Response time
- Database response, contention, or access

[23] Has a performance analysis been done?

(Yes) (23.a) What is your level of confidence in the performance analysis?

(Yes) (23.b) Do you have a model to track performance through design and implementation?

e. Testability

[Is the product difficult or impossible to test?]

[24] Is the software going to be easy to test?

[25] Does the design include features to aid testing?

[26] Do the testers get involved in analyzing requirements?

f. Hardware Constraints

[Are there tight constraints on the target hardware?]

[27] Does the hardware limit your ability to meet any requirements?

- Architecture
- Memory capacity
- Throughput
- Real-time response
- Response time
- Recovery timelines
- Database performance
- Functionality
- Reliability
- Availability

g. Non-Developmental Software

[Are there problems with software used in the program but not developed by the program?]

If re-used or re-engineered software exists

If COTS software is being used

[29] Are there any problems with using COTS (commercial off-the-shelf) software?

- Insufficient documentation to determine interfaces, size, or performance
- Poor performance
- Requires a large share of memory or database storage
- Difficult to interface with application software
- Not thoroughly tested
- Not bug free
- Not maintained adequately
- Slow vendor response

[30] Do you foresee any problem with integrating COTS software updates or revisions?

3. Code and Unit Test

a. Feasibility

[Is the implementation of the design difficult or impossible?]

[31] Are any parts of the product implementation not completely defined by the design specification?

[32] Are the selected algorithms and designs easy to implement?

b. Testing

[Are the specified level and time for unit testing adequate?]

[33] Do you begin unit testing before you verify code with respect to the design?

[34] Has sufficient unit testing been specified?

[35] Is there sufficient time to perform all the unit testing you think should be done?

[36] Will compromises be made regarding unit testing if there are schedule problems?

c. Coding/Implementation

[Are there any problems with coding and implementation?]

[37] Are the design specifications in sufficient detail to write the code?

[38] Is the design changing while coding is being done?

[39] Are there system constraints that make the code difficult to write?

- Timing
- Memory
- External storage

[40] Is the language suitable for producing the software on this program?

[41] Are there multiple languages used on the program?

(Yes) (41.a) Is there interface compatibility between the code produced by the different compilers?

[42] Is the development computer the same as the target computer?

(No) (42.a) Are there compiler differences between the two?

If developmental hardware is being used

[43] Are the hardware specifications adequate to code the software?

[44] Are the hardware specifications changing while the code is being written?

4. Integration and Test

a. Environment

[Is the integration and test environment adequate?]

[45] Will there be sufficient hardware to do adequate integration and testing?

[46] Is there any problem with developing realistic scenarios and test data to demonstrate any requirements?

- Specified data traffic
- Real-time response
- Asynchronous event handling
- Multi-user interaction

[47] Are you able to verify performance in your facility?

[48] Does hardware and software instrumentation facilitate testing?

(Yes) (48.a) Is it sufficient for all testing?

b. Product

[Is the interface definition inadequate, facilities inadequate, time insufficient?]

[49] Will the target hardware be available when needed?

[50] Have acceptance criteria been agreed to for all requirements?

(Yes) (50.a) Is there a formal agreement?

[51] Are the external interfaces defined, documented, and baselined?

[52] Are there any requirements that will be difficult to test?

[53] Has sufficient product integration been specified?

[54] Has adequate time been allocated for product integration and test?

If COTS

[55] Will vendor data be accepted in verification of requirements allocated to COTS products?

(Yes) (55.a) Is the contract clear on that?

c. System

[System integration uncoordinated, poor interface definition, or inadequate facilities?]

[56] Has sufficient system integration been specified?

[57] Has adequate time been allocated for system integration and test?

[58] Are all contractors part of the integration team?

[59] Will the product be integrated into an existing system?

(Yes) (59.a) Is there a parallel cutover period with the existing system?

(No) (59.a.1) How will you guarantee the product will work correctly when integrated?

[60] Will system integration occur on customer site?

5. Engineering Specialties

a. Maintainability

[Will the implementation be difficult to understand or maintain?]

[61] Does the architecture, design, or code create any maintenance difficulties?

[62] Are the maintenance people involved early in the design?

[63] Is the product documentation adequate for maintenance by an outside organization?

b. Reliability

[Are the reliability or availability requirements difficult to meet?]

[64] Are reliability requirements allocated to the software?

[65] Are availability requirements allocated to the software?

(Yes) (65.a) Are recovery timelines any problem?

c. Safety

[Are the safety requirements infeasible and not demonstrable?]

[66] Are safety requirements allocated to the software?

(Yes) (66.a) Do you see any difficulty in meeting the safety requirements?

[67] Will it be difficult to verify satisfaction of safety requirements?

d. Security

[Are the security requirements more stringent than the current state of the practice or program experience?]

[68] Are there unprecedented or state-of-the-art security requirements?

[69] Is it an Orange Book system?

[70] Have you implemented this level of security before?

e. Human Factors

[Will the system will be difficult to use because of poor human interface definition?]

[71] Do you see any difficulty in meeting the Human Factors requirements?

(No) (71.a) How are you ensuring that you will meet the human interface requirements?

If prototyping

(Yes) (71.a.1) Is it a throw-away prototype?

(No) (71.a.1a) Are you doing evolutionary development?

(Yes) (71.a.1a.1) Are you experienced in this type of development?

(Yes) (71.a.1a.2) Are interim versions deliverable?

(Yes) (71.a.1a.3) Does this complicate change control?

f. Specifications

[Is the documentation adequate to design, implement, and test the system?]

[72] Is the software requirements specification adequate to design the system?

[73] Are the hardware specifications adequate to design and implement the software?

[74] Are the external interface requirements well specified?

[75] Are the test specifications adequate to fully test the system?

If in or past implementation phase

[76] Are the design specifications adequate to implement the system?

- Internal interfaces

B. Development Environment

1. Development Process

a. Formality

[Will the implementation be difficult to understand or maintain?]

[77] Is there more than one development model being used?

- Spiral
- Waterfall
- Incremental

(Yes) (77.a) Is coordination between them a problem?

[78] Are there formal, controlled plans for all development activities?

- Requirements analysis
- Design
- Code
- Integration and test
- Installation
- Quality assurance
- Configuration management

(Yes) (78.a) Do the plans specify the process well?

(Yes) (78.b) Are developers familiar with the plans?

b. Suitability

[Is the process suited to the development model, e.g., spiral, prototyping?]

[79] Is the development process adequate for this product?

[80] Is the development process supported by a compatible set of procedures, methods, and tools?

c. Process Control

[Is the software development process enforced, monitored, and controlled using metrics? Are distributed development sites coordinated?]

[81] Does everyone follow the development process?

(Yes) (81.a) How is this insured?

[82] Can you measure whether the development process is meeting your productivity and quality goals?

If there are distributed development sites

[83] Is there adequate coordination among distributed development sites?

d. Familiarity

[Are the project members experienced in use of the process? Is the process understood by all staff members?]

[84] Are people comfortable with the development process?

e. Product Control

[Are there mechanisms for controlling changes in the product?]

[85] Is there a requirements traceability mechanism that tracks requirements from the source specification through test cases?

[86] Is the traceability mechanism used in evaluating requirement change impact analyses?

[87] Is there a formal change control process?

(Yes) (87.a) Does it cover all changes to baselined requirements, design, code, and documentation?

[88] Are changes at any level mapped up to the system level and down through the test level?

[89] Is there adequate analysis when new requirements are added to the system?

[90] Do you have a way to track interfaces?

[91] Are the test plans and procedures updated as part of the change process?

2. Development System

a. Capacity

[Is there sufficient work station processing power, memory, or storage capacity?]

[92] Are there enough workstations and processing capacity for all staff?

[93] Is there sufficient capacity for overlapping phases, such as coding, integration and test?

b. Suitability

[Does the development system support all phases, activities, and functions?]

[94] Does the development system support all aspects of the program?

- Requirements analysis
- Performance analysis
- Design
- Coding
- Test
- Documentation
- Configuration management
- Management tracking
- Requirements traceability

c. Usability

[How easy is the development system to use?]

[95] Do people find the development system easy to use?

[96] Is there good documentation of the development system?

d. Familiarity

[Is there little prior company or project member experience with the development system?]

[97] Have people used these tools and methods before?

e. Reliability

[Does the system suffer from software bugs, down-time, insufficient built-in back-up?]

[98] Is the system considered reliable?

- Compiler
- Development tools
- Hardware

f. System Support

[Is there timely expert or vendor support for the system?]

[99] Are the people trained in use of the development tools?

[100] Do you have access to experts in use of the system?

[101] Do the vendors respond to problems rapidly?

g. Deliverability

[Are the definition and acceptance requirements defined for delivering the development system to the customer not budgeted? HINT: If the participants are confused about this, it is probably not an issue from a risk perspective.]

[102] Are you delivering the development system to the customer?

(Yes) (102.a) Have adequate budget, schedule, and resources been allocated for this deliverable?

3. Management Process

a. Planning

[Is the planning timely, technical leads included, contingency planning done?]

[103] Is the program managed according to the plan?

(Yes) (103.a) Do people routinely get pulled away to fight fires?

[104] Is re-planning done when disruptions occur?

[105] Are people at all levels included in planning their own work?

[106] Are there contingency plans for known risks?

(Yes) (106.a) How do you determine when to activate the contingencies?

[107] Are long-term issues being adequately addressed?

b. Project Organization

[Are the roles and reporting relationships clear?]

[108] Is the program organization effective?

[109] Do people understand their own and others' roles in the program?

[110] Do people know who has authority for what?

c. Management Experience

[Are the managers experienced in software development, software management, the application domain, the development process, or on large programs?]

[111] Does the program have experienced managers?

- Software management
- Hands-on software development
- With this development process
- In the application domain
- Program size or complexity

d. Program Interfaces

[Is there poor interface with customer, other contractors, senior and/or peer managers?]

[112] Does management communicate problems up and down the line?

[113] Are conflicts with the customer documented and resolved in a timely manner?

[114] Does management involve appropriate program members in meetings with the customer?

- Technical leaders
- Developers
- Analysts

[115] Does management work to ensure that all customer factions are represented in decisions regarding functionality and operation?

[116] Is it good politics to present an optimistic picture to the customer or senior management?

4. Management Methods

a. Monitoring

[Are management metrics defined and development progress tracked?]

[117] Are there periodic structured status reports?

(Yes) (117.a) Do people get a response to their status reports?

[118] Does appropriate information get reported to the right organizational levels?

[119] Do you track progress versus plan?

(Yes) (119.a) Does management have a clear picture of what is going on?

b. Personnel Management

[Are project personnel trained and used appropriately?]

[120] Do people get trained in skills required for this program?

(Yes) (120.a) Is this part of the program plan?

[121] Do people get assigned to the program who do not match the experience profile for your work area?

[122] Is it easy for program members to get management action?

[123] Are program members at all levels aware of their status versus plan?

[124] Do people feel it's important to keep to the plan?

[125] Does management consult with people before making decisions that affect their work?

[126] Does program management involve appropriate program members in meetings with the customer?

- Technical leaders
- Developers
- Analysts

c. Quality Assurance

[Are there adequate procedures and resources to assure product quality?]

[127] Is the software quality assurance function adequately staffed on this program?

[128] Do you have defined mechanisms for assuring quality?

(Yes) (128.a) Do all areas and phases have quality procedures?

(Yes) (128.b) Are people used to working with these procedures?

d. Configuration Management

[Are the change procedures or version control, including installation site(s), adequate?]

[129] Do you have an adequate configuration management system?

[130] Is the configuration management function adequately staffed?

[131] Is coordination required with an installed system?

(Yes) (131.a) Is there adequate configuration management of the installed system?

(Yes) (131.b) Does the configuration management system synchronize your work with site changes?

[132] Are you installing in multiple sites?

(Yes) (132.a) Does the configuration management system provide for multiple sites?

5. Work Environment

a. Quality Attitude

[Is there a lack of orientation toward quality work?]

[133] Are all staff levels oriented toward quality procedures?

[134] Does schedule get in the way of quality?

b. Cooperation

[Is there a lack of team spirit? Does conflict resolution require management intervention?]

[135] Do people work cooperatively across functional boundaries?

[136] Do people work effectively toward common goals?

[137] Is management intervention sometimes required to get people working together?

c. Communication

[Is there poor awareness of mission or goals, poor communication of technical information among peers and managers?]

[138] Is there good communication among the members of the program?

- Managers
- Technical leaders
- Developers
- Testers
- Configuration management
- Quality assurance

[139] Are the managers receptive to communication from program staff?

(Yes) (139.a) Do you feel free to ask your managers for help?

(Yes) (139.b) Are members of the program able to raise risks without having a solution in hand?

[140] Do the program members get timely notification of events that may affect their work?

(Yes) (140.a) Is this formal or informal?

d. Morale

[Is there a non-productive, non-creative atmosphere? Do people feel that there is no recognition or reward for superior work?]

[141] How is morale on the program?

(No) (141.a) What is the main contributing factor to low morale?

[142] Is there any problem keeping the people you need?

C. Program Constraints

1. Resources

a. Schedule

[Is the schedule inadequate or unstable?]

[143] Has the schedule been stable?

[144] Is the schedule realistic?

(Yes) (144.a) Is the estimation method based on historical data?

(Yes) (144.b) Has the method worked well in the past?

[145] Is there anything for which adequate schedule was not planned?

- Analysis and studies
- QA
- Training
- Maintenance courses and training
- Capital equipment
- Deliverable development system

[146] Are there external dependencies which are likely to impact the schedule?

b. Staff

[Is the staff inexperienced, lacking domain knowledge, lacking skills, or understaffed?]

[147] Are there any areas in which the required technical skills are lacking?

- Software engineering and requirements analysis method
- Algorithm expertise
- Design and design methods
- Programming languages
- Integration and test methods
- Reliability
- Maintainability
- Availability
- Human factors
- Configuration management
- Quality assurance
- Target environment
- Level of security
- COTS
- Reuse software
- Operating system
- Database

- Application domain
- Performance analysis
- Time-critical applications

[148] Do you have adequate personnel to staff the program?

[149] Is the staffing stable?

[150] Do you have access to the right people when you need them?

[151] Have the program members implemented systems of this type?

[152] Is the program reliant on a few key people?

[153] Is there any problem with getting cleared people?

c. Budget

[Is the funding insufficient or unstable?]

[154] Is the budget stable?

[155] Is the budget based on a realistic estimate?

(Yes) (155.a) Is the estimation method based on historical data?

(Yes) (155.b) Has the method worked well in the past?

[156] Have features or functions been deleted as part of a design-to-cost effort?

[157] Is there anything for which adequate budget was not allocated?

- Analysis and studies
- QA
- Training
- Maintenance courses
- Capital equipment
- Deliverable development system

[158] Do budget changes accompany requirement changes?

(Yes) (158.a) Is this a standard part of the change control process?

d. Facilities

[Are the facilities adequate for building and delivering the product?]

[159] Are the development facilities adequate?

[160] Is the integration environment adequate?

2. Contract

a. Type of Contract

[Is the contract type a source of risk to the program?]

[161] What type of contract do you have? (Cost plus award fee, fixed price,....)

(161a) Does this present any problems?

[162] Is the contract burdensome in any aspect of the program?

- SOW (Statement of Work)
- Specifications
- DIDs (Data Item Descriptions)
- Contract parts
- Excessive customer involvement

[163] Is the required documentation burdensome?

- Excessive amount
- Picky customer
- Long approval cycle

b. Restrictions

[Does the contract cause any restrictions?]

[164] Are there problems with data rights?

- COTS software
- Developmental software
- Non-developmental items

c. Dependencies

[Does the program have any dependencies on outside products or services?]

[165] Are there dependencies on external products or services that may affect the product, budget, or schedule?

- Associate contractors
- Prime contractor
- Subcontractors
- Vendors or suppliers
- Customer furnished equipment or software

3. Program Interfaces

a. Customer

[Are there any customer problems such as: lengthy document-approval cycle, poor communication, and inadequate domain expertise?]

[166] Is the customer approval cycle timely?

- Documentation
- Program reviews
- Formal reviews

[167] Do you ever proceed before receiving customer approval?

[168] Does the customer understand the technical aspects of the system?

[169] Does the customer understand software?

[170] Does the customer interfere with process or people?

[171] Does management work with the customer to reach mutually agreeable decisions in a timely manner?

- Requirements understanding
- Test criteria
- Schedule adjustments
- Interfaces

[172] How effective are your mechanisms for reaching agreements with the customer?

- Working groups (contractual?)
- Technical interchange meetings (contractual?)

[173] Are all customer factions involved in reaching agreements?

(Yes) (173.a) Is it a formally defined process?

[174] Does management present a realistic or optimistic picture to the customer?

If there are associate contractors

b. Associate Contractors

[Are there any problems with associate contractors such as inadequately defined or unstable interfaces, poor communication, or lack of cooperation?]

[175] Are the external interfaces changing without adequate notification, coordination, or formal change procedures?

[176] Is there an adequate transition plan?

(Yes) (176.a) Is it supported by all contractors and site personnel?

[177] Is there any problem with getting schedules or interface data from associate contractors?

(No) (177.a) Are they accurate?

If there are subcontractors

c. Subcontractors

[Is the program dependent on subcontractors for any critical areas?]

[178] Are there any ambiguities in subcontractor task definitions?

[179] Is the subcontractor reporting and monitoring procedure different from the program's reporting requirements?

[180] Is subcontractor administration and technical management done by a separate organization?

[181] Are you highly dependent on subcontractor expertise in any areas?

[182] Is subcontractor knowledge being transferred to the company?

[183] Is there any problem with getting schedules or interface data from subcontractors?

If program is a subcontract

d. Prime Contractor

[Is the program facing difficulties with its Prime contractor?]

[184] Are your task definitions from the Prime ambiguous?

[185] Do you interface with two separate prime organizations for administration and technical management?

[186] Are you highly dependent on the Prime for expertise in any areas?

[187] Is there any problem with getting schedules or interface data from the Prime?

e. Corporate Management

[Is there a lack of support or micro management from upper management?]

[188] Does program management communicate problems to senior management?

(Yes) (188.a) Does this seem to be effective?

[189] Does corporate management give you timely support in solving your problems?

[190] Does corporate management tend to micro-manage?

[191] Does management present a realistic or optimistic picture to senior management?

f. Vendors

[Are vendors responsive to programs needs?]

[192] Are you relying on vendors for deliveries of critical components?

- Compilers
- Hardware
- COTS

g. Politics

[Are politics causing a problem for the program?]

[193] Are politics affecting the program?

- Company
- Customer
- Associate contractors
- Subcontractors

[194] Are politics affecting technical decisions?

Appendix C Glossary of Terms

acceptance criteria - The criteria that a system or component must satisfy to be accepted by a user, customer, or other authorized entity. [IEEE-STD-610]

acceptance testing - Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system. [IEEE-STD-610]

application domain - Refers to the nature of the application. Two examples are real-time flight control systems and management information systems.

audit - An independent examination of a work product or set of work products to assess compliance with specifications, standards, contractual agreements, or other criteria. [IEEE-STD-610]

availability - The relative time that an operational product must be available for use. Usually expressed as the ratio of time available for use to some total time period or as specific hours of operation.

baseline - A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures. [IEEE-STD-610]

baseline management - In configuration management, the application of technical and administrative direction to designate the documents and changes to those documents that formally identify and establish baselines at specific times during the life cycle of a configuration item. [IEEE-STD-610]

benchmark - A standard against which measurements or comparisons can be made. [IEEE-STD-610]

COTS (commercial off-the-shelf) - A type of non-developmental software that is supplied by commercial sources.

change control - A part of configuration management that reviews, approves, and tracks progress of alterations in the configuration of a configuration item delivered, to be delivered, or under formal development, after formal establishment of its configuration identification.

configuration - In configuration management, the functional and physical characteristics of hardware or software as set forth in technical documentation or achieved in a product. [IEEE-STD-610]

configuration management - A discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a controlled item, control changes to a configuration item and its documentation, and record and report change processing and implementation status.

configuration management function - The organizational element charged with configuration management.

configuration management system - The processes, procedures, and tools used by the development organization to accomplish configuration management.

critical design review (CDR) - (1) A review conducted to verify that the detailed design of one or more configuration items satisfy specified requirements; to establish the compatibility among the configuration items and other items of equipment, facilities, software, and personnel; to assess risk areas for each configuration item; and, as applicable, to assess the results of producability analyses, review preliminary hardware product specifications, evaluate preliminary test planning, and evaluate the adequacy of preliminary operation and support documents. See also: preliminary design review; system design review. (2) A review as in (1) of any hardware or software component.

customer - The person or organization receiving a product or service. There may be many different customers for individual organizations within a program structure. Government program offices may view the customer as the user organization for which they are managing the project. Contractors may view the program office as well as the user organization as customers.

design specifications - A document that prescribes the form, parts, and details of the product according to a plan.

design-to-cost - Bidding a selected, reduced set of requirements to meet cost objectives.

detailed design - (1) The process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented. See also: software development process. (2) The result of the process in (1).

development computer - The hardware and supporting software system used for software development.

development facilities - The office space, furnishings, and equipment that support the development staff.

development model - The abstract visualization of how the software development functions (such as requirements definition, design, code, test, and implementation) are organized. Typical models are the waterfall model, the iterative model, and the spiral model.

development process - The implemented process for managing the development of the deliverable product. For software, the development process includes the following major activities: translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use. These activities may overlap and may be applied iteratively or recursively.

development sites - The locations at which development work is being conducted.

development system - The hardware and software tools and supporting equipment that will be used in product development including such items as computer-aided software engineering (CASE) tools, compilers, configuration management systems, and the like.

external dependencies - Any deliverables from other organizations that are critical to a product's success.

external interfaces - The points where the software system under development interacts with other systems, sites, or people.

hardware specifications - A document that prescribes the functions, materials, dimensions, and quality that a hardware item must meet.

implementation - The act of preparing a product for use by the customer.

integration - The act of assembling individual hardware and/or software components into a usable whole.

integration environment - The hardware, software, and supporting tools that will be used to support product integration.

integration testing - Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them. See also: component testing; interface testing; system testing; unit testing.

internal interfaces - The points where the software system under development interacts with other components of the system under development.

long-term issues - Issues of strategic importance to the project that can be compromised in the heat of battle. Issues such as employee training and development, establishing and improving processes and procedures, and similar activities are important to the long term viability of the project and the organization.

non-developmental software (NDS) - Deliverable software that is not developed under the contract but is provided by the contractor, the Government, or a third party. NDS may be referred to as reusable software, Government furnished software, or commercially available software, depending on its source.

Orange Book - A security standard set by the U.S. Government as described in *Federal Criteria for Information Technology Security, Volume 1, December 1992.*

preliminary design - The process of analyzing design alternatives and defining the architecture, components, interfaces, and timing and sizing estimates for a system or component. See also: detailed design.

procedure - A written description of a course of action to be taken to perform a given task. [IEEE-STD-610]

process - A sequence of steps performed for a given purpose; for example, the software development process. [IEEE-STD-610]

product integration - The act of assembling individual hardware and software components into a functional whole.

re-engineering - The practice of adapting existing software artifacts or systems to perform new or enhanced functions. Re-engineered artifacts may be substantially different from the existing artifact.

reliability - The degree of dependability that an operational product must meet. Usually expressed as the average time to failure.

requirements analysis - (1) The process of studying user needs to arrive at a definition of system, hardware, or software requirements. (2) The process of studying and refining system, hardware, or software requirements.

reusing - Hardware or software developed in response to the requirements of one application that can be used, in whole or in part, to satisfy the requirements of another application.

safety - The degree to which the software product minimizes the potential for hazardous conditions during its operational mission.

security - The degree to which a software product is safe from unauthorized use.

software architecture - The organizational structure of the software or module.

software life cycle - The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase. [IEEE-STD-610]

software requirement - A condition or capability that must be met by software needed by a user to solve a problem or achieve an objective. [IEEE-STD-610]

software requirements specification (SRS) - Documentation of the essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces. (IEEE-STD-1012-1986])

system integration - The act of assembling hardware and/or software components into a deliverable product.

system requirement - A condition or capability that must be met or possessed by a system or system component to satisfy a condition or capability needed by a user to solve a problem. [IEEE-STD-610]

system testing - Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. See also: component testing; integration testing; interface testing; unit testing.

target computer - The hardware and supporting software system that will actually be used when the software system is fielded.

TBDs - Requirements in formal requirements statements that are to be defined.

test specifications - A document that prescribes the process and procedures to be used to verify that a product meets its requirements.

traceability - The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another. [IEEE-STD-610]

traceability mechanism - Processes and procedures (manual and/or automated) that map all software components and artifacts from source requirements through test cases.

transition plan - A plan (documented in the *Computer Resources Integrated Support Document*) specifying how products are to be transitioned from development to support.

unit - (1) A separately testable element specified in the design of a computer software component. (2) A logically separable part of a computer program. (3) A software component that is not subdivided into other components. [IEEE-STD-610]

unit testing - Testing of individual hardware or software units or groups of related units. See also: component testing; integration testing; interface testing; system testing.

Index

E

F

G

I

M

N

P

Q

R

S

T

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS
None

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for Public Release
Distribution Unlimited

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
CMU/SEI-93-TR-6

5. MONITORING ORGANIZATION REPORT NUMBER(S)
ESC-TR-93-183

6a. NAME OF PERFORMING ORGANIZATION
Software Engineering Institute

6b. OFFICE SYMBOL (if applicable)
SEI

7a. NAME OF MONITORING ORGANIZATION
SEI Joint Program Office

6c. ADDRESS (city, state, and zip code)
Carnegie Mellon University
Pittsburgh PA 15213

7b. ADDRESS (city, state, and zip code)
HQ ESC/ENS
5 Eglin Street
Hanscom AFB, MA 01731-2116

8a. NAME OFFUNDING/SPONSORING ORGANIZATION
SEI Joint Program Office

8b. OFFICE SYMBOL (if applicable)
ESC/ENS

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F1962890C0003

8c. ADDRESS (city, state, and zip code))
Carnegie Mellon University
Pittsburgh PA 15213

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.
63756E	N/A	N/A	N/A

11. TITLE (Include Security Classification)
Taxonomy-Based Risk Identification

12. PERSONAL AUTHOR(S)
Marvin J. Carr, Suresh L. Konda, Ira Monarch, F. Carol Ulrich, Clay F. Walker

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM TO

14. DATE OF REPORT (year, month, day)
June 1993

15. PAGE COUNT
24 & appendix

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)
risk assessment, risk identification, software risk, taxonomy-based questionnaire

19. ABSTRACT (continue on reverse if necessary and identify by block number)

This report describes a method for facilitating the systematic and repeatable identification of risks associated with the development of a software-dependent project. This method, derived from published literature and previous experience in developing software, was tested in active government-funded defense and civilian software development projects for both its usefulness and for improving the method itself. Results of the field tests encouraged the claim that the described method is useful, usable, and efficient. The report concludes with some macro-level lessons learned from the field tests and a brief overview of future work in establishing risk management on a firm footing in software development projects.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ■ SAME AS RPT ☐ DTIC USERS ■

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified, Unlimited Distribution

22a. NAME OF RESPONSIBLE INDIVIDUAL
Thomas R. Miller, Lt Col, USAF

22b. TELEPHONE NUMBER (include area code)
(412) 268-7631

22c. OFFICE SYMBOL
ESC/ENS (SEI)

ABSTRACT — continued from page one, block 19