Decision Support Systems 13 (1995) 223–239
North-Holland

Box-structured requirements determination methods

Alan R. Hevner [a,*] and Harlan D. Mills [b]

[a] *Information Systems, College of Business and Management and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA*
[b] *Computer Science Department, Florida Institute of Technology, Melbourne, FL 32901, USA*

Requirements determination is an iterative process of eliciting, gathering, modeling, specifying, and analyzing system requirements information. It is the most critical, yet least understood, phase of systems development. This paper presents a rigorous approach for performing requirements determination with box-structured methods. By capturing requirements information in black box transactions and transaction hierarchies, intellectual control is maintained over large amounts of requirements information. The results of the box-structured requirements determination methods provide the basis for formal system design techniques. A concise example of box-structured requirements determination is included in an appendix.

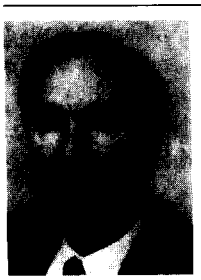
Keywords: Box structures; Requirements determination; System modeling; System development; Specification

1. Requirements determination in the systems development process

The most critical, yet least understood, phase of systems development is requirements determination. Many system development projects fail because of inadequate understandings of problem requirements. Many times, even when a system is completed, it does not solve the target business problem. It has also been observed that the identification and correction of errors in requirements statements consume a major portion of system development time and resources [4]. Requirements determination entails close cooperation between the system development team, the customer, and system users. Behavioral skills are just as important as technical skills in order to get the system requirements "right". While the issues of customer interaction during requirements determination are not the focus of this paper (see [3]), such interaction must be supported by a disciplined set of requirements determination methods. Thus, it is imperative that developers have solid methodological support for this critical phase of systems development.

Requirements determination can be described as an iterative process of four activities:

(1) *Requirements Gathering* – Information on the problem and associated solution requirements is elicited from the customer, potential



Alan R. Hevner is Professor of Information Systems in the College of Business and Management, University of Maryland at College Park. He is a faculty member of the Institute for Systems Research at Maryland and is Director of the M.S. in Systems Engineering program. He has published numerous papers in the research areas of distributed database systems, information system development, and systems engineering. He has a Ph.D. from Purdue University in Computer Science. Professor Hevner is a member of the ACM, IEEE Computer Society, and ORSA.

* Corresponding author. A.R. Hevner: (301) 405-2218, E-mail; ahevner@bmgtmail.umd.edu. H.D. Mills: (407) 569-3722, E-mail; set@zach.fit.edu

Harlan D. Mills is a Professor of Computer Science at the Florida Institute of Technology and President of Software Engineering Technology. He has written or co-authored six books and more than 50 refereed technical journal articles on topics related to software engineering. Dr. Mills received a Ph.D. in Mathematics from Iowa State University. He is an honorary Fellow of Wesleyan University and a Fellow of IBM, the American Computer Programming Association, and the IEEE. He also holds the Warnier Prize for contributions to computer science.

SSDI 0167-9236(94)E0044-E

system operators and users, and domain experts. Techniques, such as interviews, questionnaires, JAD sessions, documentation review, and observation, are used to build an information base for establishing system requirements.

(2) *Requirements Modeling* – Modeling techniques are employed to form the information into representations of desired system behavior. Effective models clearly present system behavior without imposing design restrictions. Graphic models are especially beneficial for communicating with customers to obtain confirmation of system intent and to elicit additional information. The crucial role of consensus building among customers, users, and developers is supported by the effective use of clear system models.

(3) *Requirements Specification* – The requirement specification is a formal, complete representation of desired system behavior. The specification is the detailed output of requirements determination and is the basis for subsequent system design and implementation.

(4) *Requirements Analysis* – Many forms of analysis can and should be performed on requirements models and specifications. During requirements determination, appropriate feasibility and trade-off studies should be performed to identify system opportunities and constraints. The requirement models and specifications must be analyzed for consistency, closure, completeness, and clarity. Analyses are performed to evaluate the effective use of reusable modules and common services.

As befits its importance, many methods and techniques have been developed to perform requirements determination. An excellent survey of the most well-known methods is found in [7]. We observe that the current use of requirements determination methods has several well-known problems:

- The elicitation of system objectives and requirements from customers is a very difficult process. Communication skills among developers, customers, and domain experts are essential. Methods (e.g., JAD sessions [1]) and tools (e.g., group decision support systems) have been devised to support requirements elicitation and gathering. However, many communication obstacles inhibit the collection of accurate requirements from customers [5]. The principal obstacle is finding a convenient way to define actual or desired behavior in a form that is free of implementation complexities so all interested parties can reason about the intended behaviors.
- A majority of methods use the same representation for requirements modeling and specification. It is very difficult, however, for one representation to serve both as a communications interface with the customer and as a formal statement of requirements suitable for rigorous analysis and communication with designers. We believe that graphic forms are appropriate for communication models while more formal languages (e.g., PAISLey [17]) are appropriate for specifications.
- Requirements analysis lacks an established set of metrics to evaluate the "goodness" of the requirements. There are needs for both quantitative and qualitative standards to evaluate requirement consistency, closure, completeness, clarity, etc. The advent of more formal specification languages should provide more formal ways of defining and measuring requirement metrics.
- Most requirement determination methods do not support an integrated system development process. True process integration requires common underlying concepts throughout the complete system development. The final requirement specification should based on the same concept and representation as is used in subsequent phases of system design and implementation.
- Current methods fail to recognize the iterative nature of the system development process. It is foolish to think that complete system requirements can be frozen at the beginning of system development. Controlled, incremental system development is a more realistic and practical paradigm.

The objective of this paper is to present a requirements determination approach that can address and solve the above difficulties. This approach is based on the formal, mathematically-defined concepts and principles of box structures [10, 14]. The box-structured methods presented in this paper are a rigorous, yet practical, means for

performing formal requirements determination. Our approach fits naturally into the integrated Cleanroom Systems Development Process (CSDP) for the rigorous development of near zero-defect systems [6]. In Section 2, we present an overview of CSDP as the context for box-structured requirements determination. The material in Sections 3–6 follows the four requirements activities above. In Section 3, we describe the gathering of requirements information as black box transactions. Then, requirements are modeled into transaction hierarchies as shown in Section 4. The basic transaction hierarchy is extended to include state constraints, procedural constraints, and non-functional requirements. In Section 5, we discuss the use of an extended Box Description Language (BDL) for requirements specification. Requirements analysis techniques to evaluate consistency, closure, completeness, and clarity are proposed in Section 6. Section 7 briefly describes the use of system requirements for the system design and implementation activities. The paper concludes with a short discussion of future research directions. An appendix contains a concise case study of box-structured requirements determination.

2. The cleanroom system development process (CSDP)

Figure 1 provides an overview of the Cleanroom System Development Process (CSDP). While the development activities in the diagram may look familiar, CSDP stands apart from traditional development approaches by emphasizing a number of essential, formal development concepts. The disciplined application of these ideas leads to rigorous systems development under statistical quality control [13]. We identify four Cleanroom concepts as critical to the development of an integrated environment for CSDP [8]:

Incremental Development – Incremental development allows intellectual control over complex systems by dividing the development into manageable increments. Each increment defines a complete "end-to-end" system with added functionality over previous increments. The incremental development plan is the crux of CSDP. Once an initial understanding of the system requirements is achieved, increments are defined based on several criteria, such as increment size, component reuse, development team skills, etc. [6]. In

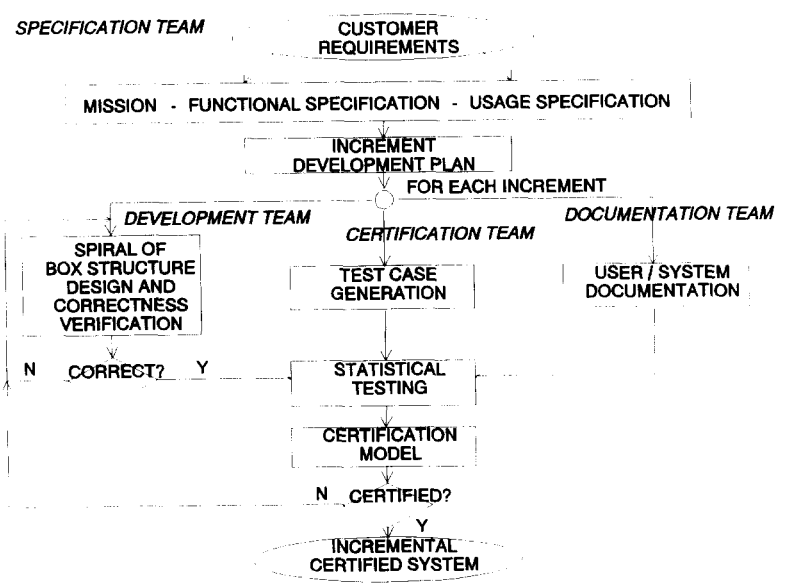


Fig. 1. The cleanroom system development process.

new systems, requirements may not be completely known, but what is known should be stable with additions brought in under control at reasonable intervals. Increments may be left open in certain aspects for later updates. Thus, further steps of requirements determination will be performed with each increment specification. So incremental development allows control over new development in both the size of system parts and their maturity.

Box Structured Analysis and Design – Box structures have an underlying mathematical foundation that permits the scale-up of analysis and design to systems of arbitrary size and complexity. Box structures model system components as data abstractions in three, increasingly detailed forms, the *black box*, the *state box*, and the *clear box*. A black box can be considered a requirements statement for a system component. Its formal description is a mathematical function of stimulus histories to response relationships. A state box encapsulates stimulus histories into state data, and its internal black box is a mathematical function of stimulus and state histories mapped to a response and new state. A clear box is a procedural description that replaces the internal black box of the state box with the designed sequential or concurrent use of other black boxes as subsystems. These internal black boxes will be expanded at the next level of design. Intellectual control over the development of complex systems is maintained by decomposing a system into smaller, more manageable components. These components are designed using box structure representations, and are organized in a *box structure usage hierarchy*. Box structures provide a new level of discipline and common language for specification and design. In particular the black box provides a design-free basis for defining specifications with no commitments to data storage in the eventual design that is needed. In [10], we demonstrate that box structures and objects are essentially equivalent. Thus, Cleanroom combines all of the advantages found in formal and object-oriented development methods.

Correctness Verification – Two types of correctness verification are used in CSDP. During the design activity, each creative design expansion from black box to state box and from state box to clear box can be verified immediately for consistency and closure. A state box (clear box) derivation produces a unique black box (state box). By comparing the derived black box (state box) with the original black box (state box), the design expansion can be verified as consistent. Closure can be determined by ensuring that all stimuli, state, and responses in each box are sufficient and necessary to support the required system functionality. Iteratively, as the design evolves, CSDP calls for the design team to perform thorough functional verifications. In a group setting, the team develops a proof that the design correctly implements the requirement specification for the increment under consideration. These functional verifications bring surprising improvements in design, even for the best software engineers. Software will literally be smaller and faster than thought possible before, with a better basis for being complete and correct, beginning with a specification black box before state boxes and clear boxes are created to meet the specification. The mathematical foundations of functional verification can be found in [11].

Reliability Certification – Testing is not recognized as part of most requirements determination techniques, but it should be. The specification team has the responsibility for discovering and specifying the usage of the desired system as well as the requirements of the system. The certification team uses the requirement specifications and the usage specifications to build a set of random test cases. The certification team can build test cases for an increment in parallel with the increment design since the requirement specifications are sufficient to define system functionality. Once the increment is designed and implemented, it is integrated with previous increments and statistical testing is performed. It is only recently understood how to bring software development under statistical quality control. In addition to usage specifications, a measure of software parts criticality is also needed to define testing in a hierarchy of statistical test cases; including the possibility of a very critical input to appear with probability 1 in a test. The reliability of the implemented system is analyzed via Mean Time To Failure (MTTF) analysis [6].

These four Cleanroom concepts provide rigorous integration throughout all activities in CSDP. The Cleanroom central repository must support the required information representations of these

concepts. It can be seen that development information is stored and manipulated primarily in box structure formats to provide an integrated system development environment [8].

Developing and testing software under statistical quality control is a new discipline that can create software practically zero defect. It is only recently understood that practically all failures in large software systems today are due to previous fixes and not to the original code. Fifteen per cent or more fixes today lead to deeper failures later on. As a result, such software does not become zero defect, but continues with defects no matter how hard people try to get them out. Creating software under intellectual control in Cleanroom can create practically zero defect software, not really imagined before. But it must start with accurate requirements determination to get proper specifications to build from.

Within CSDP, an initial stage of requirements determination is used to establish a starting incremental development plan. Then, for each system increment, requirements activities are performed to specify required system behavior at a greater level of detail. The specification team must be skilled at defining requirements through several levels of abstraction, in other words, building hierarchies of system abstractions. Intellectual control of complex system development is achieved by presenting information to different audiences at the most beneficial level of abstraction for system understanding. The requirements determination activities described in the next four sections are based on the effective use of system black boxes at varying levels of abstraction. These activities are performed iteratively in a spiral development process [12]. Each activity is performed as many times as needed and in whatever order until the systems requirements specification is completed.

3. Gathering requirements as black boxes

The black box is a "pure", design-free representation of desired system behavior. External stimuli enter the black box and responses are returned to the external environment. A meaningful black box system has a well-defined behavior for mapping sets of stimuli into sets of responses. No internal details of system state or procedurality are described in a black box.

At first glance black box descriptions can look difficult, when developers and, even, customers are already aware of state box or clear box descriptions for existing systems under study. Why not use what is already known in system descriptions? But with a little understanding, black box descriptions are not so difficult as on first appearance and uncover new insights in system behavior. For example in the 1950's the inventory systems of the Navy were based on the "K months of supply policy". These systems were discovered to order new inventory in cycles that magnified the variance in orders rather than smoothing them out. The "K months of supply policy" had been used since the 1870's and nothing like that was suspected. It looked reasonable in its state box and clear box forms, and its black box form had never been discovered before. But its black box analysis showed right away that inventory variation was magnified rather than smoothed out. This new understanding changed the way both government and industry ran inventory systems [12].

3.1. Black box transitions and transactions

During requirements determination, black box structures are used to describe required system behavior. Black box requirements are based on sets and functions that can be described in mathematical notation for small systems or subsystems or in well-structured natural language in a given context in larger systems. (In this paper, we consider only deterministic, functional system behavior. Non-deterministic behaviors can be described in box structures via relational mathematics.) In any case, a black box is defined by a mathematical function from histories of stimuli to the next response. This detailed, low-level behavior is termed a system *transition*. Let S be the set of possible stimuli, and R be the set of possible responses of a system or subsystem. The black box transition function, say f, will map historical sequences of such stimuli, in this case S^*, to responses, R, shown in the form, f: $S^* \rightarrow R$.

The description of the transition function f will be very complex for any reasonably complex system with large numbers of possible stimuli and responses. But this is a complexity of the system that must be recognized and addressed. Getting this complexity under control early in specifica-

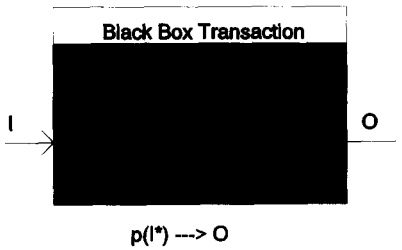


Fig. 2. Black box transaction graphic.

tion and design is much better than letting it go and trying to fix it later when the software doesn't work well.

In order to manage this complexity, we must move to higher levels of system abstraction and describe required system behaviors as system *transactions*. In [12], we define a black box transaction as a pattern of black box transitions in which all responses, but the last, are predictable by the user. The last response is new information. The sequence of stimuli to the transaction is called an *input* and the sequence of responses, including the last response, is called an *output*. In the same manner as a black box transition, the black box transaction is defined formally as a function, say p, from a history of inputs, I^*, to an output, O, in the form, p: $I^* \rightarrow O$. The black box transaction is shown in Figure 2.

In essence, the transition describes low-level, system-oriented behaviors while the transaction describes higher level behaviors that are better understood by humans (e.g., customers, users, and developers). While detailed transitions are needed for eventual system design and implementation, system transactions are presented as requirements at higher levels of abstraction. In box-structured requirements determination, requirements are elicited in terms of transactions, i.e., functions of inputs to outputs. Thus, while the system behavior will be designed and implemented in terms of thousands of individual system transitions, system requirements are typically described with less than one hundred transactions.

One observation is that a typical system will support many kinds of users, many of whom are there to make the system run for the others. For example, an on-line, all-day system will be started every day by operating people, it will need people who tune its performance, people who build the database, people who train other people in its use, and so on, in addition to the principal users to add data, retrieve data, etc. So many kinds of transitions and transactions will be called for every day. All these transactions need to be identified and planned for from the very beginning, not brought in as after thoughts to those of the principal users.

3.2. Discovering black box transactions

The input into the requirements gathering phase is some form of problem statement, typically presented as an English document. Requirements gathering tasks are performed in order to collect all information that will help to determine the particular requirements of a system that solves the presented problem. Our goal, then, is to format this information into black box transactions. To support this goal, we present a simple requirements gathering method consisting of three steps.

Requirements gathering procedure

Step 1: Identify Inputs – Via information gathering tasks, a list of system inputs is generated. It should be recognized that these inputs will be at various levels of abstraction, from databases and files to simple data variables and physical signals (e.g., a clock pulse). All potential and available inputs should be listed. An analysis of the necessity and sufficiency of the inputs will be performed later. We define the list of inputs as $\bar{I} = (I_1, I_2, \ldots, I_i)$.

Step 2: Identify Outputs – A list of required outputs from the system is generated. Close interaction with the customer and users is needed to develop a complete output list. The output list is defined as $\bar{O} = (O_1, O_2, \ldots, O_j)$.

Step 3: Form Black Box Transactions – The black box behaviors that relate the input history to the required outputs are described. A set of black box transactions is generated, $(p_1, p_2, \ldots, p_k)$. Each transaction is a function from the input history to a set of required outputs, i.e., $p_m(\bar{I}^*) \rightarrow O'_m$ where $O'_m \subset \bar{O}$. All required outputs must be produced by one or more transactions.

End of requirements gathering procedure.

The discovery of inputs, outputs, and black box transactions is an iterative process. The next phase of requirements determination, requirements modeling, forms this information into a hierarchical structure of transactions. As this hierarchy expands, further steps of information gathering will be needed to achieve consistency and completeness of the requirement specification. But with a rigorous framework, the information gathering comes under intellectual control. The black box postpones state and procedure invention, but provides a framework for dealing with the black box of a complex system with many different kinds of users and therefore many different black box inputs. As noted before, the need is to identify the entire behavior required in the black box before going into the state box and clear box designs.

4. Modeling requirements in a transaction hierarchy

The ability to handle requirements information at various levels of abstraction is essential in order to maintain intellectual control over the requirements determination process. An abstraction hierarchy of black box transactions is an effective framework for building a model of system requirements. This hierarchy supports both the top-down decomposition of system transactions and the bottom-up composition of transactions into higher level transactions. The identification of reusable subsystems and the recognition of essential system common services are supported by this box-structured modeling process.

4.1. The black box transaction hierarchy

The transaction hierarchy, shown in Figure 3, is constructed by modeling requirements information in meaningful transactions at various levels of abstraction. Typical model development would begin by identifying the top-level (i.e., level 1) system transactions. These transactions would be grouped to encompass the functional requirements for the complete system. Then, using the process of stepwise refinement, each transaction can be decomposed into a group of sub-transactions at the next level of the hierarchy. At each step of refinement, the group of transactions at the next level are verified for consistency with the parent transaction and are analyzed for closure, completeness, and clarity. These analysis procedures are discussed in Section 6.

In parallel with the top-down decomposition of required transactions, an analysis of the bottom-up composition of detailed requirements into higher level abstractions can be performed. This analysis is especially critical when reusable components from libraries or existing systems are available. The modeling of the transaction hierar-

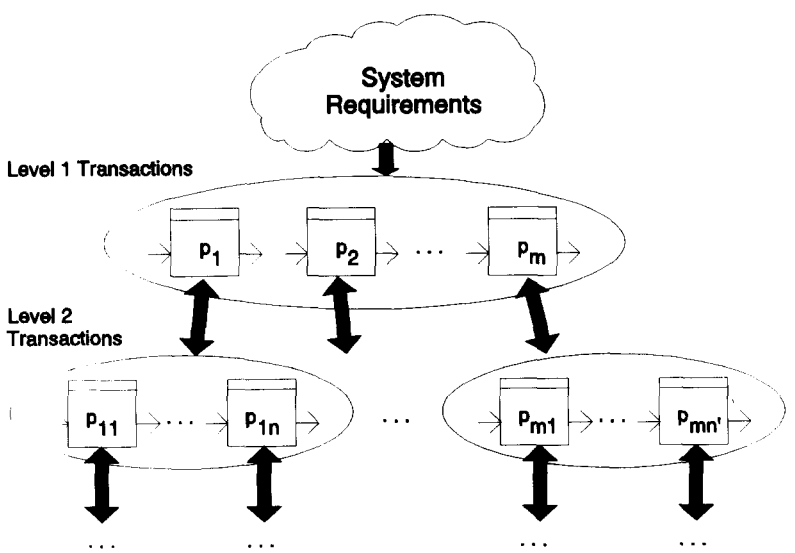


Fig. 3. Black box transaction hierarchy.

chy becomes a challenging, iterative process of matching customer and business needs with the resources available to satisfy those needs.

As an example of requirements matching in the transaction hierarchy, assume there exists a reusable software module with black box transaction behavior, $r(I'^*) \rightarrow O'$, where I' and O' are the inputs to and outputs from the module. Given a black box transaction somewhere in the hierarchy, $p_i(I''^*) \rightarrow O''$, we are able to evaluate the potential for the reusable module to match the transaction requirements. Requirements matching must be done on inputs (I' and I''), outputs (O' and O''), and behaviors (r and p_i). If an exact match is not found, several alternatives can be studied:

(1) Use the reusable module as is and modify the system requirement to accommodate its behavior.
(2) Modify the behavior of the reusable module to match the system requirement.
(3) Modify both the behavior of the reusable module and the system requirement in order to produce an effective match.
(4) Do not use the reusable module and search for other reuse opportunities or develop a module from scratch to satisfy the system requirement.

A detailed matching algorithm is needed, along with a cost tradeoff procedure to evaluate the most effective reuse strategy.

Opportunities also exist during the modeling of the transaction hierarchy to discover required system common services. A common service is a portion of the system that can be reused in several places in the transaction hierarchy. Reusable modules can be used as common services. The discovery and effective placement of common services in the transaction hierarchy model provides important design and implementation efficiencies in later stages of system development. New common services can also be defined and implemented as reusable modules for future system developments.

4.2. Customer communication

The transaction hierarchy is built through many iterations of requirements gathering, modeling, and analysis of the model. This graphic representation of the system requirements is an excellent communication device for interaction with customers, users, and business managers. In this section, we briefly discuss the objectives of customer communications during requirements modeling.

An important deficiency in the current description of Cleanroom methods is the explicit involvement of the customer at defined points in the development process. Total quality principles posit that customer requirements must be understood and met in all systems. Thus, a system that has no software errors is not "top quality" if it does not satisfy customer requirements.

We must achieve more customer involvement during Cleanroom activities in order to improve software quality and engineering productivity. User feedback is essential for discovering defects resulting from inaccurate or incomplete user requirements. The following reasons support the need for improved customer communication.

(1) A higher level of software quality is attained by eliciting and fully understanding customer needs during requirements gathering.
(2) System development time is reduced because early and continuous customer involvement leads to fewer and less severe design modifications.
(3) There has been an increased focus on creativity and innovation to meet the design needs of today's complex systems. Customers are an important source of innovation in system designs. A recent study, for certain product categories, found 70%–90% of innovations were user defined [15].
(4) Customers' requirements will change during the systems development process. Close customer involvement over the complete development life cycle will provide an efficient means of incorporating these changes into the overall system design.

In [3], the integration of a user-interactive systems development process called Joint Application Development (JAD) into CSDP is proposed. A key customer-developer interface is the transaction hierarchy. Customers are clearly able to visualize the abstract structure of the required system. The input-output behaviors of each transaction are stated precisely and reuse and common service opportunities are identified. The customer is able to recognize and correct any re-

quirements misunderstandings immediately. In addition, the intuitive nature of the transaction hierarchy supports the customer to become a full participant in the requirements determination process.

4.3. Extending the transaction hierarchy

Black box transactions are "pure" representations of functional system requirements. Any additional information, such as detailed information flows, control flows, or data structures, constrains the freedom of the system designer to produce the most effective system design. However, we recognize that such constraints may be valid based on the need to integrate with existing systems or human behaviors. Thus, we extend the transaction hierarchy to include three types of additional information:

- State Constraints
- Procedural Constraints
- Non-Functional Requirements

Often system requirements do contain design constraints on such things as the availability and use of data or the need to conform to a defined procedure. The operating or management systems environment within which a system will be embedded may provide opportunities or place demands on the specification and design of the desired system. If a specification can be modified to make more software reusable with equal power, that should be done. In addition, certain "non-functional" requirements, such as performance, behavioral, and documentation standards, can be stated in structured English forms, or in system performance models (e.g., Petri-nets [2]). It is important during requirement reviews that the system owners understand that any non-functional requirements beyond a black box are constraints upon the system's design freedom. In this process, many non-essential requirements can be discovered and eliminated.

We currently represent the information on constraints and non-functional requirements with appropriate models or structured English statements and link these artifacts to the affected transactions in the hierarchy. For example, state constraints could be described by Entity-Relationship Diagrams, procedural constraints could be described by control flow charts, and non-functional requirements could be described by performance models.

5. Requirements specification

Once the transaction hierarchy is accepted by the customer as a true reflection of system requirements, then the requirements are described in a more formal requirements specification language. The representation of requirements in a formal language provides two important advantages:

- Rigorous analysis procedures can be performed on the requirements specification, and
- A consistent, closed, complete, and clear requirements specification is given to the design and implementation team. No ambiguities or unnecessary design constraints hamper the creative tasks of system design.

We propose the use of an extended Box Description Language (BDL) [12] as the requirements specification language. Rigorous languages for requirements specification are quite recent. The formality of programming languages is necessary to make assemblers and compilers possible, but the formality of specification languages is not necessary if the specifications are not to be executed. But bringing specifications under formal control allows an entirely new level of intellectual control.

While the complete details of the requirements specification language are beyond the scope of the paper, the following illustrates a template for describing each black box transaction:

Black Box Transaction ⟨transaction-name⟩
  Parent Transaction ⟨p-transaction-name⟩;
  Sibling Transactions List of ⟨s-transaction-name⟩;
  Child Transactions List of ⟨c-transaction-name⟩;
  Input List of ⟨input-name⟩;
  Output List of ⟨output-name⟩;
  Design Constraints
    State Constraints Links to state models;
    Procedural Constraints Links to control flow models;
  Non-Functional Requirements
    Performance Requirements Links to performance models;

Behavioral Requirements Links to behavior statements;

Documentation Requirements Links to documentation standards;

Additional Requirements Links as necessary;

Behavior

Formal statement of the required system behavior in terms of a function from the input history to the output. The representation of the behavior can range from a mathematical equation to a structured English statement.

End of Black Box Transaction ⟨transaction-name⟩.

The results of the requirements specification phase are a precisely defined black box transaction hierarchy with accompanying design constraints and non-functional requirements. This evolving requirement specification is stored in a repository as the requirements definition of the system. These requirement specifications provide management a whole new capability for the description and control of system development. As noted structured English is itself a form of mathematics that can be created and used with rigor by those who know how.

6. Requirements analysis

Throughout the previous phases of requirements modeling and specification, analysis procedures are applied to measure and evaluate the quality of the system requirement. The following sections discuss several types of requirements analyses.

6.1. Requirement consistency

The rule of consistency is that each group of black box transactions in the transaction hierarchy must be consistent with its higher level parent transaction. In other words, the individual behaviors of the transactions must collectively match the behavior defined in the parent transaction. It is important to note that the interactions of the children transactions are yet to be designed. Thus, the transaction hierarchy does not exhibit referential transparency [12]. The lack of referential transparency precludes a formal verification of consistency as can be performed during box structure system design. However, an informal analysis of consistency throughout the transaction hierarchy is essential.

6.2. Requirement completeness

Via reviews with customers, users, managers, and domain experts (for the first level black box) or amongst team members (for lower level black boxes), the specification team must validate that all system requirements are captured in black boxes. The steps of verifying requirement completeness are:

(1) Make a mapping between each line of the black box and a section of the problem description.
(2) Ensure that all parts of the problem description have been covered.

6.3. Requirement closure

Closure can be validated by ensuring that every black box transaction has necessary and sufficient sets of inputs and outputs. This is termed transaction closure. During this procedure, unnecessary inputs can be deleted and additional needed inputs can be identified and gathered. When requirements are compiled informally by several people, both consistency and closure problems can arise. A single requirements statement assembled by several people under formal discipline of box structures can better insure both consistency and closure. An algorithm for performing transaction closure on black boxes is:

Black Box Closure Algorithm

Given:

$S = (s_1, s_2, \ldots, s_n)$: complete set of stimuli entering the system

$R = (r_1, r_2, \ldots, r_m)$: complete set of responses generated by the system

$F = (f_1, f_2, \ldots, f_p)$: complete set of subfunctions describing the behavior of the black box

Step 1: Check that all responses are generated:

For all r_j in R there exists a subset S_A of S and a f_k in F such that $f_k(S_A) \rightarrow r_j$. In other words, ensure that each response results from at least one stimulus subfunction.

Step 2: Check that all stimuli are used:

For all s_i in S, there exists an S_A where s_i is an element of a subset S_A of S, and there exists a r_j in R and f_k in F, such that $f_k(S_A) \rightarrow r_j$ and

$f_k(S_A\text{-}s_i) \nrightarrow r_j$. In other words, ensure that there is a stimulus subfunction for each stimulus.

Step 3: Check that all subfunctions are used:

For all f_k in F there exists a r_j in R such that $f_k(S_A) \rightarrow r_j$ where S_A is a subset of S. In other words, ensure that all stimulus-response pairs exist.

End of Black Box Closure.

6.4. Requirement clarity

Two forms of clarity are needed for effective requirements determination. The requirements model must present requirements in a form understandable to the business customer and system users. The requirement specification must present requirements in a form appropriate for system developers. The box-structured approach provides the flexibility for system requirements to be stated in the language of the problem domain. Effective use of structured English statements at high levels of the transaction hierarchy allows customers and users to better understand the requirements model. The formal, mathematics-based framework of the BDL specification is a clear starting point for detailed system design, with no unnecessary design constraints.

6.5. Use of reusable modules and common services

In Section 4, we discussed the techniques for discovering reusable modules and common services that match system requirements in the transaction hierarchy. Such discoveries must be analyzed as to their effectiveness and feasibility. Cost trade-off studies can be performed to determine buy versus build tactics for individual system modules. Additional types of analyses that must be considered in the selection of reusable modules and common services include user interface standards and supportable communications protocols.

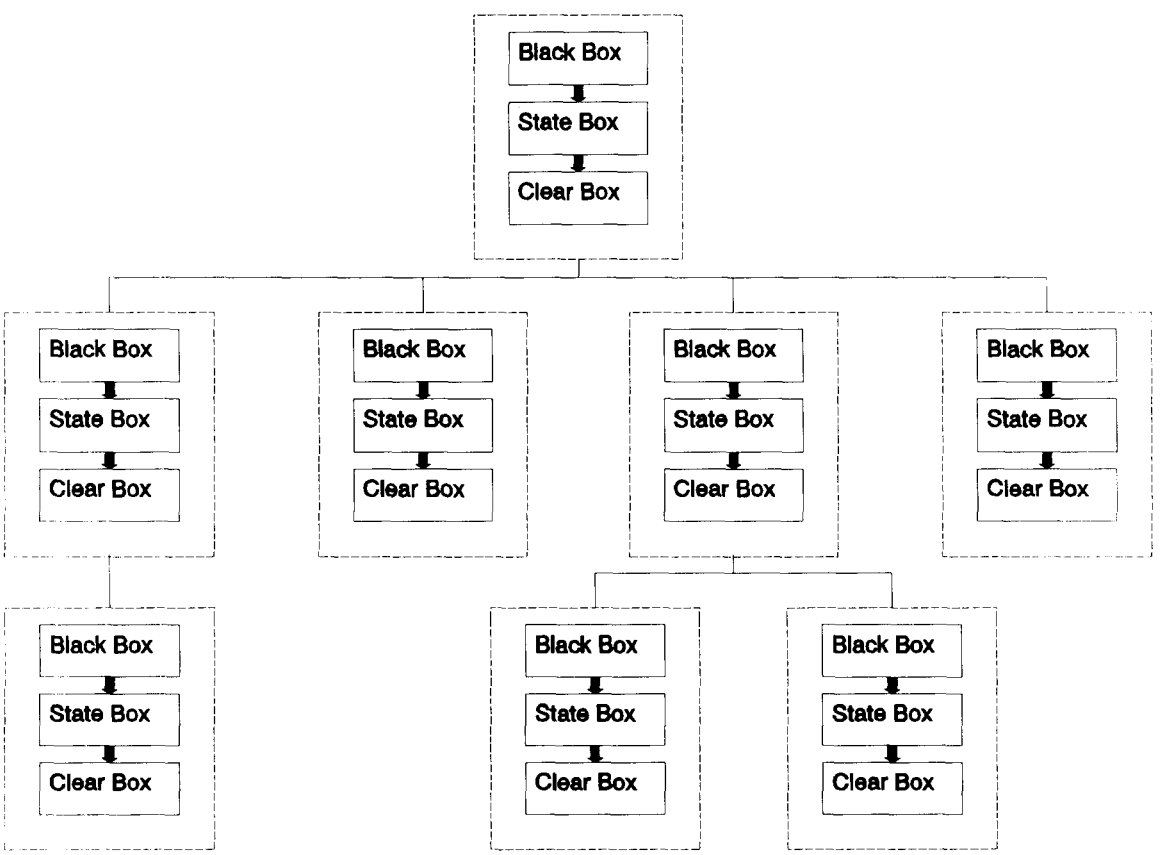


Fig. 4. Box structure usage hierarchy.

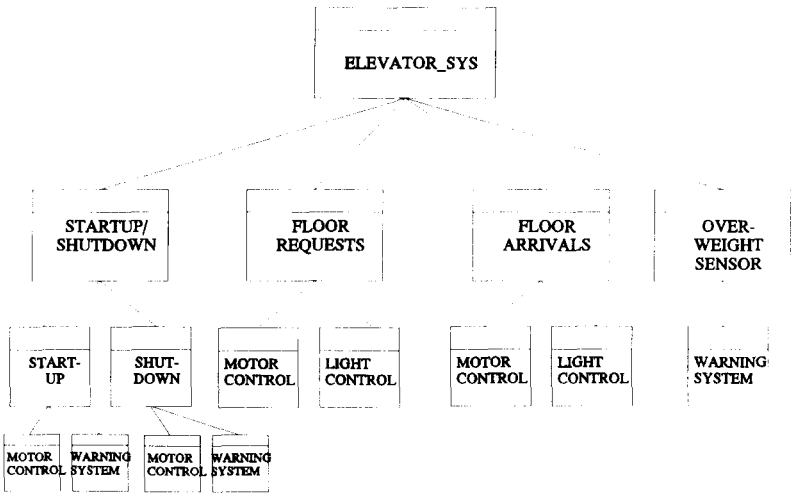


Fig. 5. Elevator system transaction hierarchy.

7. Box-structured system design

In well-defined increments, portions of the system requirement specification are passed to the system development team as shown in Figure 1. The black box transaction hierarchy serves as the basis for the creative system design process. In a top-down manner, each level of the system is designed from the black box to the state box and finally to the complete clear box. All creative design steps can be formally verified as consistent with the black box specification. Within the clear box, the black box transactions at the next system level are defined. These black boxes may or may not match exactly with the black boxes in the transaction hierarchy. New system insights and design opportunities may result in a different system structure. Such alterations should be checked with the requirements team to ensure that no system requirements are violated.

As noted before, box structures provide a formal basis for system specification and design in a single language. Verification and testing are carried out in this formal language. In contrast with programming languages which must be defined formally, it has not seemed necessary for specification and design languages to be formal as long as they are not to be executed. But the new reality of software engineering is that specification and design languages need to be formal for good engineering practices.

The development team extends the hierarchy of system components to a much lower level of detail than is given in the requirements specification. At the lowest level of design, system behaviors are described as detailed box structure transitions from individual stimuli (e.g., keystrokes, clock pulses) to individual responses (e.g., screens, signals, printed characters).

The result of the development team's work is a box structure usage hierarchy of the system increment, as shown in Figure 4. This design is verified consistent with the original system requirement specification. All box structures in the hierarchy are also verified to be complete under transaction closure. The usage hierarchy is sent to the certification team for statistical testing and certification as shown in Figure 1.

8. Conclusions and future research directions

The lack of rigorous and integrated methods for gathering and representing system requirements is a major shortcoming of existing system development methods. In this paper, we have demonstrated that box structures can provide an underlying rigor to the requirements determination phase of system development. A comprehensive set of requirements determination methods are proposed for the four requirements activities:

Requirements Gathering – Requirements information is elicited from customers and formed

into system inputs, outputs, and transactions. No internal data storage is permitted. Describing current systems without internal data seems hard at first, but becomes easier with practice and can surface issues of using internal data not otherwise visible.

Requirements Modeling – Black box transactions are formed into a hierarchical model. The transaction hierarchy is built via iterative steps of bottom-up and top-down requirements analysis. Each transaction, initially a "pure" black box requirement, is augmented with necessary design constraints and non-functional requirements. The extended transaction hierarchy serves as a communications interface with customers.

Requirements Specification – A more formal Box Description Language is used for requirements specification. The BDL is used for rigorous requirements analysis and for detailed communication with designers and implementors.

Requirements Analysis – Many types of analysis should be performed on the requirements model and specification. Reusability and common service analyses are important, as are procedures for evaluating consistency, closure, completeness, and clarity of requirements.

We are currently using the box-structured requirements determination methods, as described in this paper, on several Cleanroom system projects. An example is presented in the appendix to this paper. The use of box structures as a communications medium within the Cleanroom teams and between the teams and the customer have proven to be very useful. A full evaluation of the box-structure requirements methods in practice will be reported in future papers.

We are investigating several essential research directions based on the fundamental ideas presented in this paper:

- More structured, formal means for involving the customer in the system development process are needed. The customer must be an active participant in the requirements determination tasks and must accept "ownership" of the resulting system specification. Methods and tools should be developed to enhance the quantity and quality of customer involvement. We plan to extend our research along the ideas presented in Section 4.2.
- Requirements metrics is an area of utmost importance. We need good qualitative and quantitative measures of requirements quality. Our basic measures of consistency, closure, completeness, and clarity, described in this paper, provide only an indication of the potential for further box structure metrics. Our initial proposal for requirements metrics in Cleanroom is contained in [9].
- Reverse engineering techniques are not being used to advantage in requirements determination. Typically, one or more systems already exist that perform some of the desired system's required functions. Making effective use of the existing systems by reverse engineering them is an important step of understanding and specifying requirements.
- Rigorous requirements determination is tedious and exhausting work. Computer-Aided Systems Engineering (CASE) tools must be designed and applied to requirements gathering, modeling, specification, and analysis activities. Integrated support with the downstream development phases of design, implementation, testing, and documentation are critical for successful CASE utilization [8].
- Reuse and the identification of common services are major considerations in box structured requirements determination. Moving reuse considerations this far forward in the system development process is essential to highlight its importance in improving development productivity and system quality. Research is needed to better understand the reuse tradeoffs described in Section 4 of this paper.
- An automated transformation process to take a transaction hierarchy into a BDL requirement specification would be a tremendously useful tool.

9. Acknowledgements

We acknowledge the many colleagues with whom we are working to better understand and define the Cleanroom methods of systems development. In particular, we thank Shirley Becker, Richard Cobb, Ara Kouchakdjian, and Tom Vagoun for their contributions to this paper.

10. Appendix – elevator case study

We present a brief example of the application of the box-structured requirements determination techniques. A simple elevator control system is used in the case study. A similar example is used to demonstrate structured analysis methods in [16].

10.1. Requirements statement for the elevator system

The elevator requires a system to schedule and control one elevator in a building with 5 floors. The elevator will be used to carry people from one floor to another in the conventional way. The interior of the elevator has 5 destination buttons, one for each floor. These buttons can be illuminated by signal sent from the control unit. There is a floor sensor switch for each floor. When the elevator is at a floor, the elevator closes the switch for that floor and sends a signal to the control unit. The interior of the elevator has one illuminable arrival light for each floor number. The system should illuminate the light for a floor when it arrives at the floor and extinguish the light for a floor when it leaves a floor. Each floor of the building has a panel containing illuminable summon buttons. These buttons can be illuminated by signal sent from the control unit. Each floor except the ground floor and the top floor has two summon buttons, one for Up and one for Down. The elevator motor is controlled from the control unit by commands: Up, Down, Park. The elevator is equipped with an overweight sensor which is turned-on whenever the load capacity of the elevator is exceeded.

10.2. Requirements gathering

All necessary information is gathered into black box formats. Appropriate assumptions are made to fill in any information gaps. Since the elevator control system is not overly complex, we will present the system transactions in stimulus-response terminology designating functional behaviors. The following lists show the available stimuli and required responses for the system.

Elevator Stimuli

S1: S1.1: System Startup
S1.2: System Shutdown
S2: S2.1: UP Summons Button (includes Floor Number)
S2.2: DOWN Summons Button (includes Floor Number)
S2.3: Destination Button (includes Requested Floor Number)
S3: Floor Arrival (includes Arriving Floor Number)
S4: S4.1: Overweight Sensor ON
S4.2: Overweight Sensor OFF

Elevator Responses

R1: R1.1: Startup Elevator System
R1.2: Shutdown Elevator System
R2: R2.1: Turn-on UP Summons Button Light (includes Floor Number)
R2.2: Turn-on DOWN Summons Button Light (includes Floor Number)
R2.3: Turn-on Destination Button Light (includes Floor Number)
R3: R3.1: Motor Control UP
R3.2: Motor Control DOWN
R3.3: Motor Control PARK
R4: Turn-on Arrival Light (includes Floor Number)
R5: R5.1: Turn-off Arrival Light (includes Floor Number)
R5.2: Turn-off UP Summons Button Light (includes Floor Number)
R5.3: Turn-off DOWN Summons Button Light (includes Floor Number)
R5.4: Turn-off Destination Button Light (includes Floor Number)
R6: R6.1: Open Elevator Doors
R6.2: Close Elevator Doors
R7: R7.1: Turn-on Warning Buzzer
R7.2: Turn-off Warning Buzzer

(Note: If the Overweight Sensor is ON, a warning buzzer is sounded and the elevator remains at its current floor until the sensor indication changes to OFF.)

The following top-level system transactions are identified along with several required common service transactions.

System Transactions

T1: Startup/Shutdown
T2: Floor Requests
T3: Floor Arrivals
T4: Overweight Sensor

Common Services

CS1: Light Control
CS2: Motor Control
CS3: Warning System

10.3. Requirements modeling

Figure 5 contains a proposed Transaction Hierarchy for the elevator system. This hierarchy is generated by identifying the required functionality of the top-level system transactions and the use of the system common services. This hierarchical usage structure provides a starting point for the eventual structured design of the system in a box structure usage hierarchy.

10.4. Requirements specification

The requirements specification for the elevator system is a detailed black box description of the required system. Only stimulus history can be used in the black box functions to generate the required system responses. The following specification of the top-level system transactions is presented in black box BDL with embedded structured English to state conditions. The uses of lower level black boxes and common services are indicated by use commands with appropriate stimuli as parameters. For this example, we do not consider any non-functional requirements in the specification.

We assume that elevator safety features are based on mechanical control systems outside of this specification. For example, elevator doors will not open until the correct floor level is achieved and the doors will remain open until all passengers have safely boarded and all warning system problems are resolved (i.e., overweight conditions).

Black box specification for elevator system

begin black box function S* | |S: Elevator
black box sub-function S* | |S1: Startup / Shutdown is
B01.1 **case** S1 is
B01.2 **value** S1.1: System Startup **do**
B01.3 R1.1: Acknowledge startup stimulus and activate system - use warning system to alert customers of activation;
B01.4 **value** S1.2: System Shutdown **do**
B01.5 R1.2: Acknowledge shutdown stimulus and deactivate system - use warning system to alert customers of deactivation;
B01.6 **endcase**;
end.

black box sub-function S* | |S2: Request Buttons is
B02.1 **case** S2 is
B02.2 **value** S2.1: UP Summons Button (* includes Floor Number *) **do**
B02.3 R2.1: **use** Light – Control (UP Summons Button Light, ON, Floor#);
B02.4 **value** S2.2: DOWN Summons Button (* includes Floor Number *) **do**
B02.5 R2.2: **use** Light – Control (DOWN Summons Button Light, ON, Floor#);
B02.6 **value** S2.3: Destination Button (* includes Requested Floor Number *) **do**
B02.7 R2.3: **use** Light – Control (Destination Button Light, ON, Floor#);
B02.8 **endcase**;
(* If elevator is parked, start elevator to requested floor *)
B02.9 **if** No previous Unsatisfied Requests exist in stimulus history
(* An Unsatisfied Request is a Destination Button (S2.3) or a Summons Button (S2.1 or S2.2) stimulus that has not been satisfied by a subsequent Floor Arrival stimulus (S3) in the stimulus history. *)
B02.10 **then**
B02.11 **if** Floor# is equal to most recent Floor Arrival stimulus, S3, in stimulus history
B02.12 **then**
B02.13 R6.1: Open Elevator Doors;
B02.14 R6.2: Close Elevator Doors:
B02.15 **else**
B02.16 **if** Floor# is greater than most recent Floor Arrival stimulus, S3, in the stimulus history
B02.17 **then**
B02.18 R3.1: **use** Motor – Control (UP);
B02.19 **else**
B02.20 R3.2: **use** Motor – Control (DOWN);
B02.21 **endif**;
B02.22 **endif**;
B02.23 **endif**;
end.

black box sub-function S* | |S3: Floor Arrival is
B03.1 R4: **use** Light – Control (Arrival Light, ON, Floor#);
(° Decide whether to stop at this floor. *)

B03.2 **if** A Satisfiable Request exists for Floor# (* A Satisfiable Request is defined as the appearance in the stimulus history of stimulus S2.3 – Destination Request for Floor# or stimulus S2.1 – UP Summons Button for Floor# (stimulus S2.2 – DOWN Summons Button for Floor#) in the Current Direction of the elevator since the last arrival stimulus S4 at this same Floor#. The Current Direction of the elevator is UP if the most recent Floor Arrival stimulus in the stimulus history is less than the current Floor#, else the Current Direction is DOWN. *)
B03.3 **then**
B03.4 R3.3: **use** Motor – Control (PARK, Floor#);
B03.5 R6.1: Open Elevator Doors;
(* Turn off appropriate lights by using Light Control common service. *)
B03.6 **if** Destination Button Light for Floor# is ON
B03.7 **then**
B03.8 R5.4: **use** Light – Control (Destination Button Light, OFF, Floor#);
B03.9 **if** Current Direction is UP and UP Summons Button Light on Floor# is ON
B03.10 **then**
B03.11 R5.2: **use** Light – Control (UP Summons Button Light, OFF, Floor#);
B03.12 **if** Current Direction is DOWN and DOWN Summons Button Light on Floor# is ON
B03.13 **then**
B03.14 R5.3: **use** Light – Control (DOWN Summons Button Light, OFF, Floor#);
(* Determine movement of elevator. *)
B03.15 **if** Current Direction is UP
B03.16 **then**
B03.17 **if** An Unsatisfied Request exists for a Floor# greater than the current Floor#
B03.18 **then**
B03.19 R3.1: **use** Motor – Control (UP);
B03.20 **else**
B03.21 **if** An Unsatisfied Request exists for a Floor# less than the current Floor#
B03.22 **then**
B03.23 R3.2: **use** Motor – Control (DOWN);
B03.24 **else**
B03.25 R3.3: **use** Motor – Control (PARK, Floor#);
B03.26 **endif**;
B03.27 **endif**;
B03.28 **else** (* Current Direction is DOWN *)
B03.29 **if** An Unsatisfied Request exists for a Floor# less than the current Floor#
B03.30 **then**
B03.31 R3.2: **use** Motor – Control (DOWN);
B03.32 **else**
B03.33 **if** An Unsatisfied Request exists for a Floor# greater than the current Floor#
B03.34 **then**
B03.35 R3.1: **use** Motor – Control (UP);
B03.36 **else**
B03.37 R3.3: **use** Motor – Control (PARK, Floor#);
B03.38 **endif**;
B03.39 **endif**;
B03.40 **endif**;
B03.41 R6.2: Close Elevator Doors;
B03.42 **endif**;
B03.43 R4.1: **use** Light – Control (Arrival Light, OFF, Floor#); **end.**

black box sub-function S*||S4: Overweight Sensor is
B04.1 **case** S4 is
B04.2 **value** S4.1: Overweight Sensor ON **do**
B04.3 R7.1: **use** Warning – System (Buzzer, ON);
B04.4 **value** S4.2: Overweight Sensor OFF **do**
B04.5 R7.2: **use** Warning – System (Buzzer, OFF);
B04.6 **endcase**;
end.

10.5. Requirements analysis

During the development of the requirement model and the requirement specification, several analyses were performed to ensure correctness and quality. For example:

Requirements Consistency – The transaction hierarchy is evaluated as to the consistency of system decomposition and composition decisions.

Requirements Completeness – Each sentence of the problem statement is matched with the specific section of the requirements specification that handles that part of the system.

Black Box Closure – All stimuli and responses are verified to be necessary and sufficient to solve the problem.

Requirement Clarity – The notation and terminology used in the requirements models and specification are analyzed for clear meaning.

Use of Common Services – Common services, such as the Motor – Control, Light – Control, and Warning – System, are used to effective advantage in the specification.

11. References

[1] J. August, Joint Application Design: the Group Session Approach to System Design (Prentice Hall, Inc., 1991).

[2] S. Becker and A. Hevner, A Dynamic System Modelling Tool Using Box Structures and Petri Nets, Proceedings of the Second International Working Conference on Dynamic Modeling of Information Systems, Washington D.C. (July 1991).

[3] S. Becker, E. Carmel and A. Hevner, Integrating Joint Application Development (JAD) into Cleanroom Development with ICASE, Proceedings of the 26th Annual Hawaii International Conference on System Sciences, Decision Support and Knowledge-Based Systems Track, Hawaii (January 1993).

[4] B. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer 21, No. 5 (May 1988).

[5] T. Byrd, K. Cossick and R. Zmud, A Synthesis of Research on Requirements Analysis and Knowledge Acquisition Techniques, MIS Quarterly 16, No. 1 (March 1992).

[6] R. Cobb and H. Mills, Engineering Software under Statistical Quality Control, IEEE Software 7, No. 6 (November 1990).

[7] A. Davis, A Comparison of Techniques for the Specification of External System Behavior, Communications of the ACM 31, No. 9 (September 1988).

[8] A. Hevner, S. Becker and L. Pedowitz, Integrated CASE for Cleanroom Development, IEEE Software 9, No. 2 (March 1992).

[9] A. Hevner, T. Vagoun and D. Lemmon, Identifying Quality Measurements in the Cleanroom Development Process, Second International Conference on Software Quality, Research Triangle, NC (October 1992).

[10] A. Hevner and H. Mills, Object-Oriented System Development with Box Structures, IBM Systems Journal 32, No. 2 (1993).

[11] R. Linger, H. Mills and B. Witt, Structured Programming: Theory and Practice (Addison-Wesley, Inc., 1979).

[12] H. Mills, R. Linger and A. Hevner, Principles of Information Systems Analysis and Design (Academic Press, Inc., 1986).

[13] H. Mills, M. Dyer and R. Linger, Cleanroom Software Engineering, IEEE Software 4, No. 5 (September 1987).

[14] H. Mills, Stepwise Refinement and Verification in Box-Structured Systems, IEEE Computer 21, No. 6 (June 1988).

[15] E. Von Hipple, The Sources of Innovation (Oxford University Press, 1988).

[16] E. Yourdon, Modern Structured Analysis (Yourdon Press, Inc., 1989).

[17] P. Zave, An Insider's Evaluation of PAISLey, IEEE Transactions on Software Engineering 17, No. 3 (March 1991).