standing personal contributions to computer performance evaluation. In 1974 he joined the Software Research and Technology Staff of TRW. His efforts were devoted to improving the development of software, both through research and application, in the environment of large, complex systems. Analysis of empirical data, development of new techniques, and design of a new software requirements statement language were emphasized. He is currently a Manager at Peat, Marwick, Mitchell & Copartners, in the Management Consulting Department. He is responsible for practice development and application in the area of productivity enhancement techniques for computers and the humans using and operating them.

Dr. Bell is a member of the Association for Computing Machinery, the IEEE Computer Society, the Institute for Management Sciences, and the Computer Measurement Group. He is the author of over 30 papers in computer science.

David C. Bixler was born in Youngstown, OH, on August 1, 1949. He received the B.S. degree in chemical engineering in 1971 and the M.S. degree in computer science in 1973, both from Michigan State University, East Lansing.

Since 1973 he has been associated with the Applied Software Laboratory of TRW Defense and Space Systems Group, Redondo Beach, CA, where he has worked in the areas of software requirements engineering, computer description languages, and computer emulations. His primary areas of interest are the semantics of programming languages and verification techniques for computer programs.

Mr. Bixler is a member of Tau Beta Pi and the Association for Computing Machinery.

Margaret E. Dyer received the B.S. and M.S. degrees in mathematics in 1964 and 1965, respectively, from the University of Michigan, Ann Arbor.

From 1965 to 1973 she was associated with Teledyne Brown Engineering where her primary responsibilities were the development of a conversational language processor, the development and application of simulations, and the comparative evaluation of fourth generation data processors. In 1973 she became a member of the Technical Staff of the MITRE Corporation. At MITRE and, subsequently, with TRW Defense and Space Systems Group, she has been involved with the development of advanced techniques for the development of software for large-scale, real-time control systems. Since joining TRW in 1974 she has concentrated on research and development of advanced software requirements engineering languages and automated tools. As Manager of the REVS Software and Language Development at the TRW Huntsville Facility, she is responsible for the design and development of the Requirements Engineering and Validation System.

A Requirements Engineering Methodology for Real-Time Processing Requirements

MACK W. ALFORD

Abstract—**This paper describes a methodology for the generation of software requirements for large, real-time unmanned weapons systems. It describes what needs to be done, how to evaluate the intermediate products, and how to use automated aids to improve the quality of the product. An example is provided to illustrate the methodology steps and their products and the benefits. The results of some experimental applications are summarized.**

Index Terms—**Ballistic missile defense, methodology, real-time software, requirements engineering, software engineering, software requirements, Software Requirements Engineering Methodology (SREM), Software Requirements Engineering Program (SREP).**

I. Introduction

THE problems of correctly specifying the requirements for large software systems (particularly real-time weapon systems) have been recently highlighted by Royce [1], Boehm [2], APL [3], and the AIAA Software Management Conferences [4]. Department of Defense Directive 5000.29 [5] emphasizes the need for early software visibility, risk reduction through software requirements analysis prior to the second Defense System Acquisition Review Council review of a weapon system (DSARC II) and greater "front end" development. This problem has been under study the past three years in the Software Requirements Engineering Program (SREP), performed by TRW Defense and Space Systems Group for the Ballistic Missile Defense Advanced Technology Center (BMDATC). SREP is one part of an overall BMDATC Software Development System described by Davis and Vick [6].

The objective of the research was to synthesize a methodology which addressed the technical and management aspects of generating software requirements. The technical aspects were to include the identification of the activities to be performed, the intermediate products (e.g., functional simulation of the processing), the form in which the requirements were to be specified, and any language or support software to be used to improve the requirements quality or speed up the requirements generation. The management aspects were to include techniques for scheduling and evaluating intermediate milestones, so as to improve the visibility of the status of the requirements.

Manuscript received June 21, 1976; revised September 16, 1976. This work was supported in part by DASG60-75-C-0022.

The author is with TRW Defense and Space Systems Group, Huntsville, AL 35805.

The goal was to develop procedures and tools to lower the life cycle cost and schedule for developing software by making the requirements generation phase more manageable, enabling the generation of software specifications with fewer errors, and making the requirements modification more manageable and error free.

The results of this research include the following: 1) a survey of the state of the art in software requirements engineering [7]; 2) analysis of the types of software requirements problems [8]; 3) an approach to specifying testable functional and performance requirements [9], [10]; 4) the definition and implementation of the Requirements Statement Language (RSL) [11], and the Requirements Engineering and Validation System (REVS) [12], which embody that approach; and 5) a set of steps for generating and evaluating the software requirements utilizing RSL and REVS, reported here.

The combination of the language, support software, and steps is called the Software Requirements Engineering Methodology (SREM). SREM treats the phase of the software development which starts when the system requirements and responsibilities are first allocated to the data processing subsystem. Its product, a specification of the processing to be performed, is independent of the architecture of the hardware and software which is to satisfy it. The processing requirements constrain all of the data processing which could be accomplished in software (even though it may ultimately be implemented in special purpose hardware, e.g., fast Fourier transforms). This provides the foundation for an informed selection of adequate data processing hardware, and a subsequent software design to meet the processing requirements.

The synthesis of the methodology was aided by frequent experimental application of preliminary results. Due to length limitations, this paper will present only an overview of the methodology concepts, steps, and experimental results. Section II discusses the problem characteristics identified in previous BMDATC research efforts. Section III describes the key concepts of the methodology. Section IV describes the methodology steps, products, and techniques for their evaluation. Section V presents an example application of SREM to illustrate its products and benefits. Section VI briefly describes the results of its experimental application, and Section VII presents a summary of the lessons learned.

II. Problem Definition

The problems of generating software requirements have been a topic of continuing research sponsored by BMDATC since 1970. For example, between 1970 and 1973, the Terminal Defense Program (TDP) was sponsored in part to demonstrate that software requirements for large unmanned weapon systems could be written in a computer independent fashion. A set of software requirements was written for a prototype real-time BMD system, and a process design methodology was utilized to design, develop, and test software for a vector processor. The TDP demonstrated the following principles.

1) A specification of the real-time software, containing highly complex logic and algorithms, can be written which states equations to be solved, not how to solve them. Although the resulting software was implemented on a vector processor, it was evaluated to be implementable on any large centralized processor, i.e., computer independent.

2) A computer-independent software specification provided more design freedom in the design of the operating system and applications software than had previously been obtained in a specification of that detail.

3) The development of an analytical simulation of the requirements by implementing example algorithms for each type of processing aided greatly in the identification of specification errors before software developments. Approximately 80 errors were reported in the software development stage from a specification of over 400 pages.

Areas identified as needing improvement included the following.

1) Traceability of the simulation to the requirements—without extreme discipline, they tended to drift away from each other.

2) Testability of the performance requirements—even when equations are provided, accuracies and response times for the processing must be constrained in testable terms.

3) Traceability of requirements—traceability was generally weak from a specification paragraph back to an originating specification document paragraph.

It was with this background that the SREP was initiated. During its first phase, the nature of the requirements generation problem was analyzed. The symptoms were obvious: in almost every software project which fails, the requirements are accused of being late, incomplete, over-constraining, and just plain wrong. The published literature (e.g., [2], [13]-[15]) discussed these symptoms. Fore example, the CCIP-85 study [2] indicated that errors generated during the requirements phase are the most expensive to fix—they are discovered late, and usually require new code to be added, the fundamental design of the software to be modified, and a large amount of retesting performed. This implies that the generation of more error-free requirements has a high cost leverage, and that even small improvements would be worthwhile. Recently, Bell and Thayer [8] analyzed existing software development programs to determine the nature of the requirements errors and their frequencies. Such information is necessary to assure that the right issues are addressed by the methodology. Ten desirable properties of a software specification were then summarized [10]: completeness, consistancy, correctness, testability, unambiguity, design freedom, traceability, communicability, modularity (or change-robustness), and automatability.

A survey was made of the state of the art techniques and methodologies for requirements generation [6], [7]. Its primary conclusion was that no overall methodology existed which addressed the generation of requirements for large, real-time software satisfying the above properties.

As a result of the foregoing analyses, three goals were then identified for an SREM: 1) a structured medium or language for the statement of requirements, addressing the properties of unambiguity, design freedom, testability, modularity, and communicability; 2) an integrated set of computer-aided tools to assure consistancy, completeness, automatability, correctness; and 3) a structured approach for developing the requirements in this language, and for validating them using the tools.

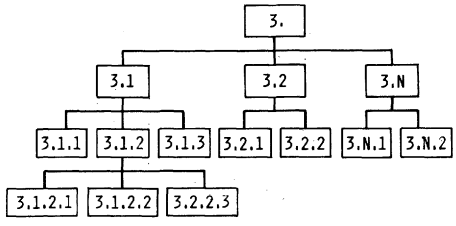

Fig. 1. Decomposition by functional hierarchy.

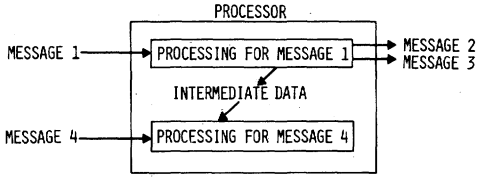

Fig. 2. Processing description.

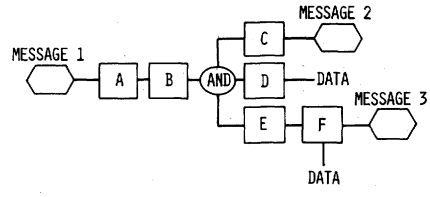

Fig. 3. Decomposition by PATHS.

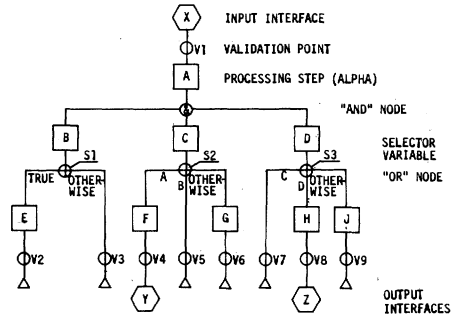

Fig. 4. R-Net nomenclature.

III. Key Concepts

The conventional way to describe software requirements is in terms of a hierarchy of functions (e.g., see MIL-STD-490 [16]). Fig. 1 illustrates this approach, in which all processing is divided into functions (e.g., communications control, device scheduling, surveillance processing, engagement control), and each of these are further subdivided into subfunctions and sub-subfunctions. A fundamental difficulty of this approach is that the processing for one input message might be described by a number of the sub-subfunctions; and the input required to exercise a specific sub-subfunction may be difficult to construct. Such a specification is hard to test, since most requirements are stated at the sub-subfunction level or below. Also, it implies a specific design, subroutines to implement sub-subfunctions.

The first key concept of SREM is based on the observation that real-time software is tested by inputting an interface message and extracting the results of its processing—output messages and the contents of memory. To illustrate this, consider Fig. 2. When MESSAGE 1 is input, three results are to occur: two messages are to be output by the processor, MESSAGE 2 and MESSAGE 3, and data resulting from processing the message are to be saved to process a subsequent message, MESSAGE 4. Thus, testable requirements must be specified in terms of data input and output.

The second key concept of SREM addresses these problems by defining the processing to be performed in terms of relationships of input messages, output messages, the processing steps, and data utilized and produced. Fig. 3 illustrates such a description of processing. When MESSAGE 1 is received, it will be processed by Steps *A* and *B*; then each of the processing Steps *C* (resulting in the output of MESSAGE 2), *D* (resulting in the storage of data), and *E* and *F* (resulting in the storage of data and the output of MESSAGE 3) takes place in any order, indicated by the "AND" node joining the processing steps to *B*. Note that the sequences of processing steps *ABC*, *ABD*, and *ABEF* describe the processing to be performed in a design-free manner, i.e., no subroutine hierarchy is implied. A sequence of processing steps connecting the arrival of a message at an input interface to the termination of processing of the message is called a PATH. (Note that this usage of PATH differs slightly from the concept of path used by some authors in that a PATH is a specified sequence of processing, and therefore does not contain "loops.") In Fig. 3, the sequence *ABC*, *ABD*, and *ABEF* are PATH's.

A third key concept addresses the statement of performance requirements: a test is defined in terms of variables measured on the PATHS. The places on the PATHS where data are measured are called validation points, and are analogous to test points in electrical circuits where voltages and currents are to be measurable. The definition of performance in terms of a test serves two purposes: it assures testability, and it communicates the requirement unambiguously.

The above approach was applied to a number of simple examples, and appeared to be sufficient for the statement of requirements. The approach was then applied to the preliminary translation of the TDP software specification. This was the most complex problem which has been addressed to date, and the complexity factor soon made its presence felt. The processing of one particular message was found to have a total of 137 distinct results depending on the previous history of processing. The specification of processing in terms of the stimulus-response combinations was found to be testable and unambiguous, but difficult to comprehend, and difficult to determine whether all combinations had been addressed. This demonstrated the need for a more compact notation.

To address this problem, the fourth key concept of SREM is that the PATHS processing a given type of stimulus should be integrated into a network called a requirements network (R-Net). If a PATH execution is conditioned upon the contents of the message or the database, this condition is summarized as a value of a selector variable; the PATH of processing is selected based on the value of this variable. An example R-Net is provided in Fig. 4. Note that 18 combinations of results are sum-

marized by combinations of eight PATHS and the values of three selector variables. The 137 distinct results discussed above were summarized by 21 PATHS.

Specification in this format maintains the advantages of testability and design freedom (all PATHS were in fact still explicit), while providing for visibility of the relationships between the PATHS.

These results were formalized using an extension of the graph model of computation [17] developed at UCLA to describe the operation of software. The details of this extension are reported in [9] and [10].

With this approach, a number of other experiments were carried out to write preliminary processing requirements for a proposed air traffic control system (which included some man-in-the-loop considerations), an underseas surveillance system, and a medical information system. Confidence in the approach was gained, and no further modifications to the approach were found to be necessary.

The fifth key concept of SREM is the use of a formal language, RSL, for the statement of requirements based on the above concepts. A formal language is needed to reduce ambiquity and serve as input to the support software (automatability).

The sixth key concept is the use of automated tools to speed up and validate the requirements. These tools are integrated into an REVS accepting RSL as input. These tools check the requirements for completeness and consistency, maintain traceability to originating requirements and simulations, and generate simulations to validate the correctness of the requirements. Modularity is enhanced by the maintenance of the requirements and their traceability in a centralized database; a flexible facility to extract such information from the database provides for the documentation of the requirements.

The capabilities of REVS and a preliminary version of RSL were defined. Both were refined by experimental application. The resulting RSL and REVS are described in [11] and [12].

The seventh key concept is that the methodology steps produce intermediate products which are evaluated for completeness. The production of intermediate products is necessary for planning and scheduling the steps. The ability to evaluate the products for their completion is necessary to assure that a step is in fact completed, and provide for management visibility and control. An overview of these steps is provided below.

IV. Methodology Steps and their Products

The SREM steps address the sequence of activities and usage of RSL and REVS to generate and validate the requirements. It assumes that system functions and performances have been allocated to the data processor, and have been collected into a Data Processing Subsystem Performance Requirement or DPSPR (see [6] for more details). Each step produces intermediate products which are evaluated for their completion. The description which follows is of necessity simplified to present the main ideas in limited space. An example is provided in the next section to illustrate the use of the methodology.

Fig. 5 provides an overview of the steps of the methodology, indicating the products of each step, and criteria for evaluation for its completion. The first step translates and interprets the DPSPR into a requirements baseline written in RSL. This provides the mechanism for an early review of the DPSPR for adequacy and for the planning of the remainder of the requirements generation activities. The second step addresses the traceability of performance requirements on the processing PATHS back to the DPSPR. In the third step, the sensitivity of the PATH performances to the satisfaction of the DPSPR is determined, the test for the performance requirements is established, and the Process Performance Requirements (PPR) are published. In the fourth step, an example algorithm is selected for each processing step, and the analytical feasibility of the algorithmic requirements is demonstrated via simulation. Each of these steps is discussed below in more detail.

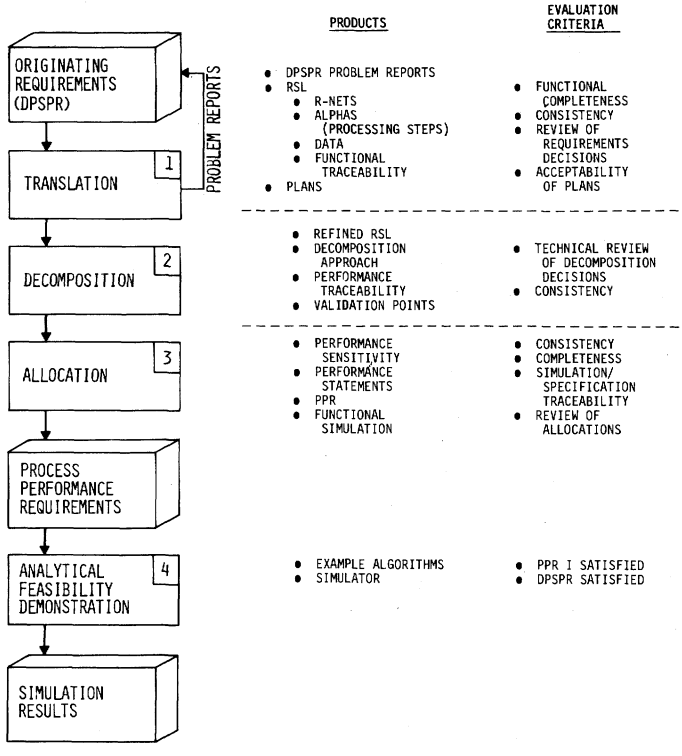

Fig. 5. Overview of the methodology steps.

A. Step 1—Translation

The issues addressed in Step 1 are: the adequacy of the DPSPR for the generation of the processing requirements; the early baselining of the functional requirements; and the budgeting and scheduling of the requirements generation activities. The activities of this step include the summary of the DPSPR requirements paragraphs as RSL ORIGINATING_REQUIREMENTS entered into the REVS data base; the generation of the R-Nets, data, and processing steps with traceability back to the DPSPR; the analysis of the consistency and completeness of those requirements; the generation of DPSPR problem reports for all problems identified; and the planning of the remainder of the requirements generation activities. The translation activity is complete when the source of every functional processing requirement and its associated performance requirement is either identified, or is covered by a DPSPR problem report; and every DPSPR requirement is mapped onto some processing.

Products and their evaluation: When this analysis is com-

plete, a formal review of the DPSPR is held at which the times for the answers to the DPSPR problem reports are set, thereby allowing the scheduling of the requirements generation activities. This ensures early visibility of when the requirements products will be available.

The conduct of the review is assisted by the use of the data extraction and analysis capabilities of REVS, e.g., the functional completeness and consistency can be verified by REVS, and the traceability of the DPSPR statements to the R-Nets can be displayed. The review of the DPSPR is complete when each problem previously identified has been withdrawn, fixed, or a date set for its resolution.

The review of the plan for subsequent activities is complete when the budgets, schedules and technical approaches for defining performance requirements for the processing PATHS are consistent with the resources allocated to the requirements engineering phase.

When these conditions are satisfied, the translation step is complete and the resulting requirements can be placed under configuration control.

Whenever modifications to the DPSPR are generated, the activities of Step 1 are repeated, and the requirements and schedules are updated accordingly. If portions of the DPSPR are modified, the traceability features can be used to identity the affected portions of the baselined requirements in the REVS data base, thereby allowing a quick response for the processing of such changes.

B. Step 2–Decomposition

The decomposition step addresses two issues: the incorporation of the processing to satisfy the subsystem performance constraints into the processing requirements, and the preliminary definition of the performance requirements. The step is finished when all requirements on the data processor identified in the interface specification are accounted for in the processing description, and the form of the performance requirements and the engineering trade studies to obtain the performance bounds are identified.

The design and development of the other subsystems of the overall system frequently place restrictions and additional requirements on the processing to be performed. For example, it is in the subsystem design phase that the detailed restrictions for scheduling radar pulses or communication messages are determined. Such information is usually recorded in the interface specifications as interface constraints. Such constraints are expressed in RSL and are entered into the REVS data base as additional ORIGINATING_REQUIREMENTS, and are traced to the appropriate PATHS of the modified R-Nets. This highlights the implications of a "to be determined" (TBD) in the originating requirements. If additional PATHS are added to an R-Net, some portions of Step 1 may need to be repeated, e.g., the modification of the requirements activities budgets and schedules.

The preliminary definition of the performance requirements involves three activities: the identification of the form of the requirements, specification of the variables of the software to be measured (i.e., the data to be collected at validation points), and the recording of the decisions made to relate these accuracy and timing requirements back to the satisfaction of the DPSPR performance requirements. These engineering decisions (their assumptions, the alternatives considered, and the rationale for the final selection) are necessary for understanding the requirements traceability; they are vital to an orderly assessment of the impact of requirements changes.

In order to allocate the DPSPR performance values to the values of accuracy and timing for the PATHS, a set of performance models linking the PATH performances to the DPSPR performance may be necessary. If so, the form of these models is identified, and the data and resources necessary to accomplish their development are estimated.

Products and their evaluation: The products of this step include a set of decisions which describe the manner and rationale for the allocation of performance; the traceability of the PATH performances to the DPSPR performance via the decisions; the revised R-Nets, including the validation points and the data to be recorded; and models linking the PATH performances to the DPSPR performance.

The decisions constituting the allocation approach should be evaluated by a formal design review for invalid assumptions, inadvertent overconstraints, etc. This review is assisted by the capability of REVS to extract the traceability of the engineering decisions to the DPSPR, the PATHS and other decisions. The adequacy of the projected performance models should also be reviewed to assure that all measures of performance have been accounted for. This should be a technical review conducted much like the preliminary design review for a software design. The key to having such a design review is the recording of the traceability, the decomposition decisions, and the performance studies. Because the R-Nets may have been refined, a set of completeness and consistency checks should again be done using the automated analyzers in REVS. In addition, the consistancy of the test for the requirement and the data available at the validation points should be determined. If requirements are being developed in phases, this review is repeated for each phase.

Finally, the plan for the third step is compared to the performance allocation approach to assure that the work can be accomplished with the allocated resources; if not, the modifications to the plan may be necessary.

C. Step 3–Allocation

During the third step, the sensitivities of the PATH performances to the DPSPR performances are determined. This is necessary to establish the tradeoff between timing and accuracies of the different PATHS which are to satisfy the requirements and to select an allocation which is not overly restrictive in any particular dimension. It is also desirable to use the least constraining form of the performance requirements (e.g., the volume of an ellipsoid might be constrained rather than the length of any particular axis).

For complicated systems like weapons systems, it is necessary to develop a functional simulator of the process which simulates the operation of the processing at a message by message level. This is used to check the expected behavior of the system, and it is frequently the only way that meaningful data processing loading information can be determined from the

definition of the system load (i.e., the system load may be in terms of the number of objects which it must deal with, while the data processing load is in terms of the number of messages to be handled). The simulator is generated using the REVS simulation generation facilities, thereby assuring the traceability of the simulation and the requirements.

When the performance for each processing PATH has been established, the requirement and its test are written in RSL. The procedure which tests the requirement uses the data at the validation points. The results of the engineering trade studies are recorded to maintain traceability. The data extraction facilities of REVS are then used in the preparation of the software requirements specification. This eliminated the introduction of additional errors in the publication activity.

Products and their evaluation: The products of this step include the identification of the sensitivity of the DPSPR performance to the PATH performances; the software requirements specifications, including the performance statements and their tests; and a functional simulator representing the processing.

The sensitivity analyses and the establishment of the performance values for the PATHS are subject to design review. The performance statements and tests must be consistent, and the final version of the R-Nets, the data, and processing test descriptions and their relationships must satisfy the properties of consistency and completeness. When all of these measures have been satisfied, the PPR can be given to a process designer for the design of software which implements the process on a specific selection of processing hardware.

D. Step 4—Analytical Feasibility Demonstration

In Section II, we noted that analytical simulation of requirements by implementing example algorithms for each type of processing greatly aids the requirements validation for highly complex systems. Such simulation is sometimes desirable to demonstrate that the critical processing requirements can, in fact, be met. For example, it may not be entirely obvious that a specific tracking accuracy can be achieved within a specific number of radar pulse returns. If so, analytical feasibility should be demonstrated before attempting the design of an algorithm for the real-time software. In addition, it provides for a direct check that algorithms which meet the PPR will, in fact, meet the originating DPSPR requirements.

This approach also aids in the communication of the requirements. Recently, Meseke, evaluating his experiences with the Safeguard software requirements, reported [13]:

> . . . experience suggests that it is probably best to state the performance requirement and then provide a recommended technique to be used at the designer's option.

The activities involved are those of building any large simulator: algorithm packages are established with requirements on input, output, and processing. Algorithms are selected for each processing step; if algorithms are not available which satisfy the requirements, they must be developed. The requirements engineer can then use the REVS facilities to combine algorithms into an analytic simulator. This simulator can be driven with realistic interface data produced by a simulation of the environment. If REVS is used to generate the simulation, the traceability of the simulator to the specification is guaranteed.

Products and their evaluation: The product of this step is a simulator which embodies sample algorithms and which demonstrates that the critical processing can be performed (although not necessarily on any specific machine within the specified response times).

The measure of whether this has been accomplished is simply the test for meeting the DPSPR requirements and the tests in the PPR for the processing accuracy. Note that the feasibility of accomplishing the desired processing within the required time responses for a specific processor cannot be determined without a preliminary software design. Thus this step demonstrates computational feasibility, not real-time feasibility.

E. Discussion

The above description identifies the sequence of steps in the development and validation of the software requirements. There is no implication that all requirements must be platooned through the steps in unison: rather, it is possible to develop a portion of the requirements through Step 4 before the requirements for others have finished Step 2. All of this scheduling and sequencing information is the proper subject of the plan for the requirements development. There is a considerable body of evidence and agreement that a software development should be specified in terms of a software development plan, with milestones for groups of capabilities. The discussion above indicates that the same type of discipline can and should be applied to the development of the requirements as well.

The identification of the reviewable products for the requirements generation activities is the key to the scheduling and control of those activities. Whenever changes to the DPSPR are forwarded to the SRE activity, a modified version of Step 1 is accomplished. An estimate of the technical, cost, and schedule impact of implementing such a change is tied to the PATHS of processing. Visibility of the generation of requirements is then achievable.

V. Example

In order to illustrate the steps of the methodology, their products, and the kinds of errors identified by those steps, consider the "patient-monitoring" problem used by Stevens, Myers, and Constantine [18]. Although simply stated, it contains many of the elements of actual real-time systems, and illustrates the need for a definitive statement of the requirements before software design is initiated. The problem is the following ([18, p. 135], sentence numbers added):

> 1) A patient monitoring program is required for a hospital. 2) Each patient is monitored by an analog device which measures factors such as pulse, temperature, blood-pressure, and skin resistance. 3) The program reads these factors on a periodic basis (specified for each patient) and stores these factors in a data base. 4) For each patient, safe ranges for each factor are specified (e.g., patient X's valid temperature range is 98 to 99.5 degrees Fahrenheit). 5) If a factor falls outside of the patient's safe range, or if an analog device fails, the nurse's station is notified.

TABLE I
EXAMPLE RSL DATA DESCRIPTIONS

```
ORIGINATING_REQUIREMENT:  SENTENCE_2.
     DESCRIPTION:  "DEFINES ANALOG DEVICE MEASUREMENTS".
     TRACES TO:  MESSAGE DEVICE_REPORT.

MESSAGE:  DEVICE_REPORT.
     PASSED THROUGH:  INPUT_INTERFACE FROM_DEVICE.
     MADE BY:  DATA DEVICE_NUMBER, DATA TYPE_MESSAGE,
               DATA DEVICE_DATA.
     TRACED FROM:  SENTENCE_2.

DATA:  DEVICE_DATA.
     INCLUDES:  DATA PULSE, DATA TEMPERATURE,
                DATA BLOOD_PRESSURE,
                DATA SKIN_RESISTANCE.

ENTITY_CLASS:  PATIENT.
     ASSOCIATES:  DATA PATIENT_NUMBER,
                  DATA SAFE_FACTOR_RANGE
                  FILE FACTOR_HISTORY.

DATA:  SAFE_FACTOR RANGE.
     INCLUDES:  DATA LOW_PRESSURE, DATA HI_PRESSURE,
                DATA LOW_TEMPERATURE, DATA HI_TEMPERATURE,
                DATA LOW_SKIN_RESISTANCE,
                DATA HI_SKIN_RESISTANCE.
     TRACED FROM:  SENTENCE_4.

FILE:  FACTOR_HISTORY.
     CONTAINS:  DATA MEASUREMENT_TIME, DATA HPULSE,
                DATA HTEMPERATURE, DATA HBLOOD_PRESSURE,
                DATA HSKIN_RESISTANCE.
     TRACED FROM:  SENTENCE_3.
```

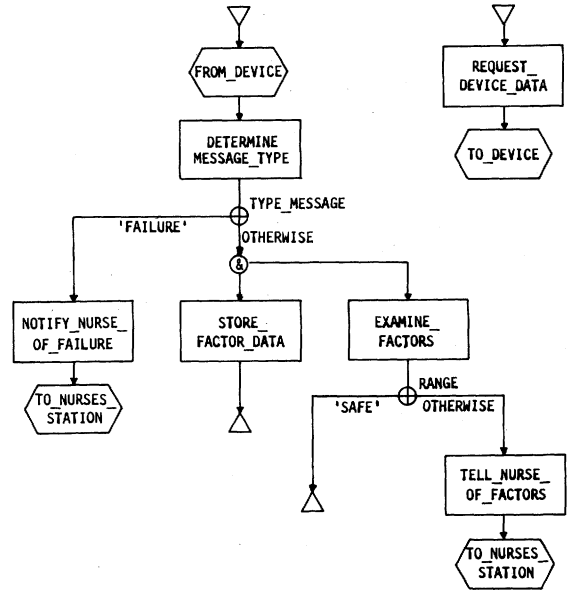

Fig. 6. Example R-Net.

In a real-life case, the problem statement would contain more detail. However, it is sufficient to initiate the development of the requirements.

A. Step 1—Translation

First, each sentence is written as an ORIGINATING_REQUIREMENT in RSL. Sentence 1 imposes no specific requirement. Sentence 5 identifies two output messages, i.e., factor out of range, and the device failure notification. Sentences 2 and 5 identify two input messages: device data, and a message indicating a device failure of some sort (this latter is implied, not explicit). Sentences 2 and 3 identify required contents of the processing data base, i.e., a factor history, and a set of safe factor ranges for each patient. These data are partially described in RSL in Table I. Note that patient data are associated with the patient; RSL describes this by defining an ENTITY_CLASS PATIENT, which has data ASSOCIATED with it. In this way, RSL describes data contents of the software without imposing a data base design. Note that the measurement and the file containing the history of the measurements have different names because they represent different data.

Sentences 3 and 5 identify processing to be performed. Four PATHS of processing are identified for device data to cover the cases of device failure, factors within safe ranges, and factors outside safe ranges. In addition, Sentence 3 identifies a scheduling function; these are represented in two R-Nets shown in Fig. 6. Note that "STORE_FACTOR_DATA" is parallel to checking the factors for safe ranges.

Each of the processing steps on the R-Net is described in RSL as an ALPHA, which includes a definition of the INPUT and OUTPUT DATA, a description, and other attributes. When attempting to define the ALPHA EXAMINE_FACTORS, the input data are found to be the device data and the safe ranges for the patient's factors: the output data consists of the variable RANGE which is used to determine which branch of the R-Net is next executed. In trying to describe the processing, it becomes clear that some correlation is necessary between patient number and device number; therefore, PATIENT_DEVICE_NUMBER was added to the patient description. The resulting RSL is contained in Table II.

TABLE II
ADDITIONAL RSL FOR EXAMPLE

```
ALPHA:  EXAMINE_FACTORS.
     INPUTS:  DATA DEVICE_DATA, DATA SAFE_FACTOR_RANGE.
     OUTPUTS:  RANGE.
     DESCRIPTION:  "THIS PROCESSING STEP FIRST RETRIEVES THE SAFE
                   FACTOR DATA ASSOCIATED WITH THE DEVICE, COMPARES
                   THE DEVICE DATA TO THE SAFE FACTOR RANGES FOR
                   THE PATIENT BEING MONITORED, AND DETERMINES
                   WHETHER THE FACTORS ARE IN BOUNDS (RANGE_SAFE)
                   OR OUT OF BOUNDS".

DATA:  PATIENT_DEVICE_NUMBER.
     INCLUDED IN:  DATA SAFE_FACTOR_RANGE.
```

The RSL is analyzed for completeness and consistency. Two things are identified with the assistance of REVS: the safe factor ranges data associated with the patient are never created, and the patient factor history is created but not used. The creation of the safe factor ranges requires the definition of another interface FROM_NURSES_STATION with a message to set the safe factors; and another message is created to retrieve the patient history data so that they are used. Both of these changes are traced to a decision documenting the assumption, and a DPSPR problem report is generated.

Further analysis reveals that the units of all the variables at the interfaces are not specified, and the contents of the device error message are not even known. These are also documented in DPSPR problem reports.

Finally, the sources of performance requirements for the processing PATHS are addressed. It is clear that some sort of time-response requirements should be imposed upon the processing which notifies the nurse of a device failure or a factor out of range. Accuracies for the processing should also be speci-

fied. Also, no limits are provided on the number of devices or the total required message arrival rates. No information is provided in the problem statement from which this information can be derived. Without this information, the performance requirements (response times and accuracy requirements) for the PATHS will TRACE TO no ORIGINATING_REQUIREMENTS. Thus the lack of the information is identified in a DPSPR problem report.

The products of Step 1 are the RSL and the DPSPR problem reports. Note that the problem reports were not prompted by some superior skill or experience in analyzing requirements, but by attemps to "fill in the blanks" provided by the RSL. Thus RSL provides not only a language in which to record the requirements, but prompts the requirements engineer to ask the right questions.

B. Step 2–Decomposition

To illustrate the impact of the detailed design of the monitoring device and communication hardware on the processing requirements, consider the following questions.

1) What is the exact manner in which the analog device can fail, and inform the data processor of the failure? Additional processing may be required to identify a failure, e.g., measurements unchanged for three messages.

2) What time delays are experienced from the query of the device until a message is sent to the data processor? Is a failure mode that it simply not respond? This may require a "time-out" feature.

3) Is communication between the devices and the processsor accomplished by time sharing one message net, or by one line per device? If time shared, additional scheduling constraints may be introduced.

4) Does calibration of the analog devices require data processing support? If so, what are the calibration procedures and what processing is required?

The answers to each of the above questions could require a refinement of the RSL data, ALPHA'S, and R-Nets.

To illustrate the performance decomposition, consider the response time requirements to notify the nurse of a device failure. If a response time constraint is not derivable from the originating requirements, it will be determined by a decision not traceable directly to the originating specification (e.g., within 10 seconds of reading the device). The fact that this was an "arbitrary" decision will be recorded as a requirements decision, and appear in the performance traceability.

Consider the scheduling of the patient monitoring. The problem statement identified that each patient should be monitored at the specified rate. The decomposition step would attempt to quantify the requirement by addressing what should be measured (e.g., the patient monitoring times, and the rate requested for each patient), the form of the requirement (e.g., time of measurement plus reciprocal of rate), and allowed tolerances (e.g., plus or minus N seconds). This will result in a set of validation points and associated data to be measured.

C. Step 3–Allocation

In Step 3, the values of the tolerances are selected, and a test is written in terms of the data at validation points. Thus, the tolerance for meeting the scheduling of the monitoring is established (e.g., plus or minus 60 seconds), and is recorded as a specific test. The maximum load is defined, and a PPR can be published.

The utility of a functional simulator can be seen for even this simple example. A simulator would project the message rates into and out of the processor, the length of the patient factor files as a function of time (or nursing shift), and the message rate to the nurse's station. Thus issues related to the dynamic behavior of a data processor satisfying the requirements and the nurse/machine interface can be surfaced. Note that these are functions of the requirements, not a specific implementation.

D. Step 4–Analytical Feasibility Demonstration

Because of the simplicity of this example, no analytical simulators appear to be necessary for this problem. For more complicated extensions of the problem, e.g., having the computer control injections based upon the patient's monitored factors, a "non-real-time" demonstration of feasibility might well be in order.

E. Summary

Even though this example is simple, it illustrates many of the problems of real-time software requirements: the initial problem statement does not specify all of the functions to be performed nor all of the interface data, and it lacks testable performance requirements. Software designed from these requirements will reflect these inadequacies. SREM systematically surfaces these problems early, and substantially aids the development of sound processing requirements.

VI. Experimentation

The goal of the SREP was the synthesis of a methodology which could be applied to large, complex problems, not just simple problems like that of Section V. To that end, experimental applications of the methodology were continuous. Some of the early development experiments were discussed in Section III; others are discussed below.

A. Real-Time Test Control

An early independent application of the preliminary concepts provided confidence in the correctness of the approach. When the software requirements specification for a real-time test control program needed extensive modification, it was rewritten in terms of stimulus/response relationships, with all conditions for execution of different PATHS identified?

This version of the specification was found to be an improvement in several dimensions. Because of its design-free nature, it did not need modification when several modifications to the software design occurred. The cost of maintaining the requirements during the next two years were less than half of the cost before the rewrite for the same level of change traffic. Response time to respond to requirements changes was similarly reduced. Traceability of the requirements to the software test plan was simple and direct due to the input/output nature of the requirements. Although qualitative in nature, these evalua-

tions indicated that the projected benefits of a specification written in terms of PATHS were achievable.

B. Track Loop Experiment

In the track loop experiment, the methodology was applied to the generation of the processing requirements for the tracking portion of the TDP software specification. The purpose of this experiment was to demonstrate that the contents of the software requirements could be written in RSL, to evaluate the products of the methodology steps, and to evaluate the steps of the methodology previously defined. The experiment products included an example of the DPSPR contents used to generate the requirements (about 20 pages), and a prototype software specification written in RSL (80 pages).

In the first phase of the experiment, an originating specification was written containing the previously specified DPSPR contents. This DPSPR was heavily reviewed before publication. In the second phase, the processing functional and performance requirements were developed in RSL following the steps of the methodology; REVS was not yet available, so the RSL was keypunched on cards and periodically listed.

At the end of Step 1, a review was held of the DPSPR contents. A surprising number of ambiguities (2-4 per page) were surfaced as a result of the Step 1 activities (e.g., did the elevation constraint apply to altitude or to the angle between horizontal and the radar line-of-sight to a target?). A number of "holes" in the specification of the performances were identified by attempting to trace the effect of the performance of a PATH of processing back to DPSPR required performances. An early conclusion of the experiment was that the Step 1 DPSPR review was explicit and thorough.

During Steps 2 and 3, the performance requirements approach was formulated, and a number of the requirements were written in RSL. The documentation of the design decisions and their use to describe the traceability of DPSPR to PPR performance requirements was met by enthusiasm; the communication of the requirements, as well as the understanding of the traceability, was enchanced. The specification of the performance requirements in terms of a test flushed out further ambiguities. As one of the participants explained, "If I could have written the requirements in English, I would have been done in three days; but you can't wave your hands in RSL."

When completed, the RSL was formatted into a PPR and evaluated. The general conclusion was that the SREM steps did produce an acceptable PPR, that RSL was sufficient to define the requirements, and that these requirements were more design-free, modular, traceable, and testable than a "standard" specification.

The methodology steps, RSL, and the projected REVS capabilities were reviewed against the experimental results, and all were modified where a weakness was detected. For example, the statement of design-free performance requirements in RSL was strengthened, and the capabilities of REVS to extract requirements data for presentation in a specification were upgraded.

During the next phase of the experiment, the requirements were written in the upgraded RSL, and a message-by-message functional simulator was written using RSL and executed using the REVS simulation generation and execution facilities. Several inconsistencies were discovered by REVS—some simple, some quite subtle. The resulting requirements were demonstrably free of many types of common errors (i.e., inconsistent units, data used before initialized, required processing which resulted in data not used, etc.). The resulting requirements were used as a "test case" to validate the REVS capabilities. These requirements also form the basis for the Methodology User's Manual, a "textbook" on software requirements generation.

C. Summary

The development of the methodology for generating software requirements required not only analysis and synthesis, but the experimental application of the concepts to realistic situations. The experiments proved necessary to define a set of steps which would generate software requirements for realistic cases.

VII. Conclusions

The SREP set out to identify and solve the problems of generating and validating software requirements for large real-time weapon systems. A set of key concepts was formulated to address all of the problems discussed in Section II. A set of steps, a language, and support software were developed using those concepts. The concepts, steps, language, and software were applied to "realistic" problems continuously to assure that they addressed the real technical and managerial problems, not just textbook simplifications.

The first benefit of the research is that all of the activities necessary for the generation of real-time software requirements have been identified, together with their required input and expected outputs. A second benefit is the demonstration of the clear distinction between requirements for processing, and design of processing; much of the activity in software development called "functional design" is really a detailed definition of the requirements for the software as a whole. A third benefit of the research is the development of the methodology steps, language, and support software itself, and the demonstration of their adequacy to address realistic problems. Although modifications and augmentations may be found to be desirable to address other problems (e.g., man-in-the-loop software), the current methodology provides a sound starting point for application to real projects.

Analysis of the experimental results to date indicates that the specification properties of testability, automatability, traceability, and certain kinds of consistency and completeness have been achieved; and that the remaining properties of design freedom, correctness, communicability, and unambiguity have been significantly improved. The identification of the methodology steps, and the techniques for evaluation of their completion, provide the foundation for visibility into the requirements generation activities. This visibility is crucial for planning and management control of the requirements generation and maintenance phases of software development.

An assessment of the quantitative benefits of using the methodology on the generation or modification of weapons system software is underway now that the RSL and REVS are opera-

tional. A further set of experiments are underway to address these questions.

References

[1] W. W. Royce, "Software requirements analysis, sizing, and costing," in *Practical Strategies for Developing Large Software Systems*, E. Horowitz, Ed. Reading, MA: Addison-Wesley, 1975.

[2] B. W. Boehm, "Software and its impact: A quantitative assessment," *Datamation*, vol. 19, pp. 48–59, May 1973.

[3] "DOD weapon systems software management study," Appl. Phys. Lab., Johns Hopkins Univ., Baltimore, MD, rep. SR95-3, June 1975.

[4] *Proc. Software Management Conf.*, 1st series, AIAA, 1976.

[5] "Management of computer resources in major defense systems," Dep. of Defense, directive 5000.29, Apr. 26, 1976.

[6] C. G. Davis and C. R. Vick, "The software development system," this issue, pp. 69–84.

[7] I. F. Burns *et al.*, "Current software requirements engineering technology," TRW Systems Group, Huntsville, AL, Aug. 1974.

[8] T. E. Bell and T. A. Thayer, "Software requirements: Are they really a problem?," in *Proc. 2nd Int. Software Engineering Conf.*, Oct. 1976.

[9] M. W. Alford and I. F. Burns, "An approach to stating real-time processing requirements," presented at Conf. Petri Nets and Related Methods, Massachusetts Inst. Technol., Cambridge, MA, July 1–3, 1975.

[10] M. W. Alford and I. F. Burns, "R-Nets: A graph model for real-time software requirements," presented at MRI Conf. Software Engineering, New York, NY, Apr. 1976.

[11] T. E. Bell and D. C. Bixler, "A flow-oriented requirements statement language," presented at MRI Conf. Software Engineering, New York, NY, Apr. 1976.

[12] T. E. Bell, D. C. Bixler, and M. E. Dyer, "An extendable approach to computer-aided software requirements engineering," this issue, pp. 49–60.

[13] D. W. Meseke, "Safeguard data processing system: The data processing system performance requirements in retrospect," *Bell Syst. Tech. J.* special suppl., 1975.

[14] F. J. Buckley, "Software testing–A report from the field," presented at IEEE Symp. Computer Software Reliability, New York, NY, Apr. 30–May 2, 1973.

[15] J. D. McGonagle, "A study of a software development project," J. P. Anderson and Co., Sept. 1971.

[16] Department of Defense, "Military standard specification practices," rep. MIL-STD-490, Oct. 1968.

[17] V. C. Cerf, "Mulitprocessors, semaphores, and a graph model of computation." Dept. of Comput. Sci., Univ. of California, Los Angeles, rep. UCLA-ENG-7223, Apr. 1972.

[18] W. P. Stevens, G. F. Myers, and L. C. Constantine, "Structured design," *IBM Sys. J.*, vol. 13, no. 2, pp. 115–139, 1974.

Mack W. Alford received the B.A. and M.A. degrees in mathematics from the University of California, Los Angeles, in 1962 and 1966, respectively. In 1971, he was named a TRW Fellow, and attended graduate school in computer science at UCLA.

In 1961 he was employed by Space Technology Laboratories (now TRW Defense and Space Systems Group), Redondo Beach, CA, where he was primarily responsible for writing requirements for computer programs to analyze rocket propulsion systems. Form 1970 to 1973 he worked on writing requirements for large, real-time software. In 1973 he moved to Huntsville, AL, to work on the Software Requirements Engineering Program, addressing the question of how real-time software requirements should be written.

The Software Development System

CARL G. DAVIS AND CHARLES R. VICK

Abstract–**This paper contains a discussion of the Software Development System (SDS), a methodology addressing the problems involved in the development of software for ballistic missile defense systems. These are large real-time, automated systems with a requirement for high reliability. The SDS is a broad approach attacking problems arising in requirements generation, software design, coding, and testing. The approach is highly requirements oriented and has resulted in the formulation of structuring concepts, a requirements statement language, process design language, and support software to be used throughout the development cycle. This methodology represents a significant advance in software technology for the development of software for a class of systems such as BMD. The support software has been implemented and is undergoing evaluation.**

Index Terms–**Requirements, software design, software development, software engineering, specifications, validation, verification.**

Manuscript received July 22, 1976; revised September 14, 1976.

The authors are with the U.S. Army Ballistic Missile Defense Advanced Technology Center, Huntsville, AL 35807.

I. Introduction

THE development of highly reliable software for large real-time weapons systems on schedule and minimizing life cycle costs has been recognized as one of the most significant problems confronting the defense community today [1], [2]. One of these weapons systems having many complexities, unique features, and highly stressing characteristics is a Ballistic Missile Defense (BMD) system.

BMD systems must be able to detect and defend against threatening objects, e.g., missile warheads, reentering the atmosphere at several thousand meters per second with a reentry phase of only several seconds. The BMD mission is of strategic importance with the cost of failure being the potential loss of offensive capability or large loss of life. BMD systems include large, powerful phased array sensors that must detect reentering objects and provide information for identification, classification, and guidance of the highly maneuverable