APPLICATION OF MODERN SOFTWARE TECHNIQUES TO MODELING AND SIMULATION

Ronald M. Huhn
Edward R. Comer
Harris Government Communications Systems Division
P.O. Box 37
Melbourne, Florida 32901

It is commonly agreed that software developments tend to be high risk activities; simulation is recognized as being even more "exciting". Great emphasis is being placed to develop methodologies which lower the risk of software development. Since a major portion of simulation activity is software oriented, it is natural to look to these modern software methodologies for solutions applicable to the modeling and simulation community. The total solution to current dilemmas is many years away. However, significant progress is being made to develop a comprehensive software development methodology that is readily extendable to simulation activities. The major additional burden of modeling and simulation is the handling of the system modeling part of the process. This paper introduces an Integrated Software Methodology (ISOMET) currently in use which has been successfully applied to simulation projects. Modifications to the methodology to accommodate specialized modeling considerations are discussed.

1. INTRODUCTION

The lack of a formal methodology for modeling and simulation is receiving considerable attention in the literature (see Reference Section for a sample). The problems are numerous and complex, the proposed solutions varied and sometimes conflicting. This paper adds to this volume of literature addressing the issue, while hopefully not adding to the present state of confusion. In fact, the methodologies successfully applied to date would indicate the "light at the end of the tunnel" has finally appeared.

In order to understand the origins of this methodology, it is important to have insight into the environment of its birth. While modeling and simulation is accomplished within numerous groups within the Harris Corporation, a single Modeling and Simulation Group serves as the focal point for such activities. The well established group has experience in numerous communications and information handling applications, primarily for government customers. As such, the Group must deal with a rising number of strict government standards. The Group, by practice, strives to deliver a user-oriented simulation package, thus minimizing future dependencies for analysis.

For years the Modeling and Simulation Group worked as an entity of Systems Engineering, as is typical with most such groups. However since 1979, the Group has operated as part of the Computer Systems Department which is primarily software oriented. The Department for the last two years has experienced an "explosion" in growth. This growth was manageable through a similarly drastic revolution in software methodology. Being key in the development of a software methodology, a similar approach naturally was developed for simulation. What was initially a mere reorganization alignment has resulted in a natural merging of simulation and software.

The approach presented is not purported to solve everyone's problems, nor be universally applicable. It has however been extremely successful where applied. While personal bias would indicate widespread potential to the community, only time and the judgement of many will tell.

The paper is organized by first presenting an evaluation of the current environment in both simulation and software, followed by a brief problem statement. The Integrated Software Methodology used as a basis is outlined and discussed. Its application to modeling and simulation is presented as related to the development life cycle. Finally, current status and assessments are summarized in Conclusions.

2. ENVIRONMENT

Simulations have a high potential for failure.

The modeling and simulation community is in a dilemma over a plethora of languages and numerous methodologies for simulation designed to overcome this problem. Clema (1980) concluded that "...we have a small number of artisans with proven track records in consulting, industry, academia, and government who are generally successful, and a second group of simulation practitioners, large in number, who perhaps have not acquired the necessary skills and experience to achieve the success desired by the customer." Simulation remains a "black magic" art form, not having yet developed into a "science".

Major languages are currently extending their capabilities and in effect "growing" closer in both appearance and features. The trend is for "richer" simulation languages. It is felt that this new generation of languages will solve the current problems. Yet there are numerous examples of simulation failures which cannot be attributed to a poor simulation language.

It is apparent that no one is worrying about the total simulation methodology.

Software, likewise, has a high potential for failure.

By contrast, the software problem is currently being investigated by numerous agencies and contracts: RADC is sponsoring enhancement of SREM (Software Requirements Engineering Methodology) to address the requirements definition problem; numerous Ada projects are aiming to contribute to implementation, test and O & M phases; the Army CENTACS and BMDATC are pursuing methodologies across the development spectrum. The list of potential contributors is so long that the DOD Management Steering Committee for Embedded Computer Resources is tasked with tracking and coordinating the numerous developments.

Software technologies frequently do not coordinate with practiced management philosophies, resulting in complex human interfaces, unplanned and unbudgeted activities and an untrackable software development. As a result, software produced by organizations with established, documented and sometimes marketed software methodologies, may not be developed in a sound and rigorous fashion.

The difficulty in integrating a complete methodology from existing techniques stems from the fact that each was developed from a different perspective. As a result, most software methodologies do not easily lend themselves to integration with other techniques without modification.

Classically the software engineer and the simulation analyst have had a lot in common: both dealt with complex and frequently ambiguous problems on a daily basis; both possessed "black magic" skills; both were frequently required to perform miracles; both "spoke" unusual languages; both worked in an environment of few rules; neither were understood by management nor were they manageable; both were considered somewhat "strange" by counterparts in other fields. It is no wonder that these two disciplines are somehow related.

The great hope is that the fields of management, software engineering and simulation will be able to produce an overall methodology for simulation.

3. PROBLEM STATEMENT

The basic problem is two-fold: first, there is no established methodology for characterizing a system as a model; secondly, there is no single proven methodology for reliably automating a model as simulation software.

The problem in modeling is the inability to adequately document the system model with traceability from the customers view of the system to the system model. A preoccupation exists in the selection of a simulation language before the system problem and its model are understood. The classic problem is in the attempt to use a simulation language to express the system model.

The production of a simulation from a system model is a software process. The tendency is to treat simulation language representations of a model as mere inputs to a simulation language processor. Even input data for a simulation language such as GPSS is really a high level language, and requires the same disciplines as that for software developments.

Software technology in general is deficient in that no single methodology covers the entire development process from beginning to end. As a result, the existing methodologies leave areas in the development process to chance and thus contribute to the problems encountered in software development. A recent report by the Software Acquisition and Development Working Group (Jones, et. al., 1980) highlighted major problems currently existing in the acquisition and development of military embedded computer systems, as summarized in Figure 1. Similar findings have been noted in numerous DOD and GAO reports. The fact is that many technically sound methodologies fail because they are either too difficult to use, too costly or too short lived in a project, providing no basis for following phases of the software life cycle.

With such problems in the modeling and software aspects, it is no wonder that simulation projects have such difficulty.

4. ISOMET - AN INTEGRATED SOFTWARE METHODOLOGY

While a number of valuable software techniques have been in existence for several years, only recent work has developed the concept of an "integrated approach". This philosophy of integration brings a large number of techniques together to be applied throughout the development cycle in a coherent manner. Such a methodology must not only address technical issues, but also establish management, organizational and accounting procedures which are best suited for the task.

To meet this need, an Integrated Software Methodology (ISOMET) was conceived to address the total software problem. The basic intent of ISOMET is to provide a collection of integrated policies, procedures, standard practices, and guidelines which provide increased productivity and management visibility of the software development process. Harris provides this integrated software development environment by coordinating the elements of Software Development Practices, Planning & Tracking, and Management & Controls. The relationships to software development are shown in Figure 2.

The Software Development Practices define the software methodology which is the "silent leader" of ISOMET. In addition to providing technical guidance to the project, the methodology provides the basis for Planning & Tracking and the development objectives to the Management and Control function. The following subsections provide additional details on the three facets of ISOMET.

"...all facets of the software acquisition and development process need varying degrees of improvement".

1. The multiplicity of government standards causes inefficiency and confusion.
2. Projects are often started with inadequate planning.
3. SW development projects are being conducted with a lack of good management practices.
4. The government inadvertently impacts cost and schedules by specifying inappropriate hardware.
5. Often there is a mismatch between contract type and complexity.
6. Security requirements impact software development costs.
7. Software is unique. The software industry is the only industry required to build useable products right the first time without prototyping.
8. No adequate means exist to estimate lifecycle costs.
9. Software development productivity must be increased. One method is through the use of automated tools.

Fig. 1
Conclusions of the Final Report of the Software Acquisition and Development Working Group

4.1 SOFTWARE DEVELOPMENT PRACTICES

The Software Development Practices provide an environment that produces a top-down structured design, provides a basis for Planning & Tracking, and visibility for Management & Control of the project for each phase of the software life cycle. The Software Development Practices consist of: Documentation, Top-Down Partitioning, Control and Data Flow Specification, Structured Process Description, Data Storage & Organization, Software Verification & Validation and Design Guidelines.

Documentation as the design progresses is the key ingredient of the methodology because the written word provides the only interim measure of successful software development. It provides good communications between the customer, management and software team members. In addition, review materials are easily derived from the documentation.

The internal standard for software documentation is the Progressive Project Document (PPD). The PPD provides an unfolding description of the software project, the software requirements, and the eventual design. The PPD is highly modular due to the technical writing approach called storyboards (Tracy, et. al., 1965). Simply, a storyboard is a 1 to 2 page discussion of a single thesis, and the PPD is a collection of storyboards arranged in a predetermined order. The arrangement is such that the requirements and design unfold in a top-down scenario which aids in the communication and presentation of the software work. The organization allows for quick and easy reference, and the modularity facilitates document updates, changes, and rewrites. The methodology and PPD work hand in hand in that guidelines specified by the methodology affect the content of the PPD, and the PPD reflects the design philosophy of the methodology which is, simply, top-down.

The PPD virtually encompasses all meaningful information regarding a software development. As such, virtually any deliverable document may be derived from the PPD by a technical writing staff. These documents may include requirements documents, design documents, interface specifications, user manuals or maintenance manuals. It also serves as an efficient means for dealing with the variety of government documentation standards (e.g. MIL-STD-483/490, DOD 7935.1S, MIL-STD-1679). Use of this approach allows the design team to become intimately familiar with a single documentation method, and most importantly, a standard communication medium.

Top-down partitioning is supported with guidelines and graphic techniques during the Definition and Design Phases. The Functional Partitioning Diagram is used during the Definition Phase as a graphic aid for the stepwise refinement process. The Software Partitioning Diagram provides a graphic aid during the Design Phase to illustrate the software decomposition.

The Control & Data Flow Diagram provides a graphic aid to illustrate relationships and interfaces between functions in the Definition Phase and modules in the Design Phase. It provides an early checkpoint in the top-down partitioning process by revealing a complex structure of interfaces if partitioning is not performed correctly and it is used in the documentation as additional supporting information.

Harris uses a standardized version of Program Design Language (PDL) for process specification to insure that the benefits of structured programming will occur regardless of the implementation language. It supports successive refinement from the highest level into code and has been found to

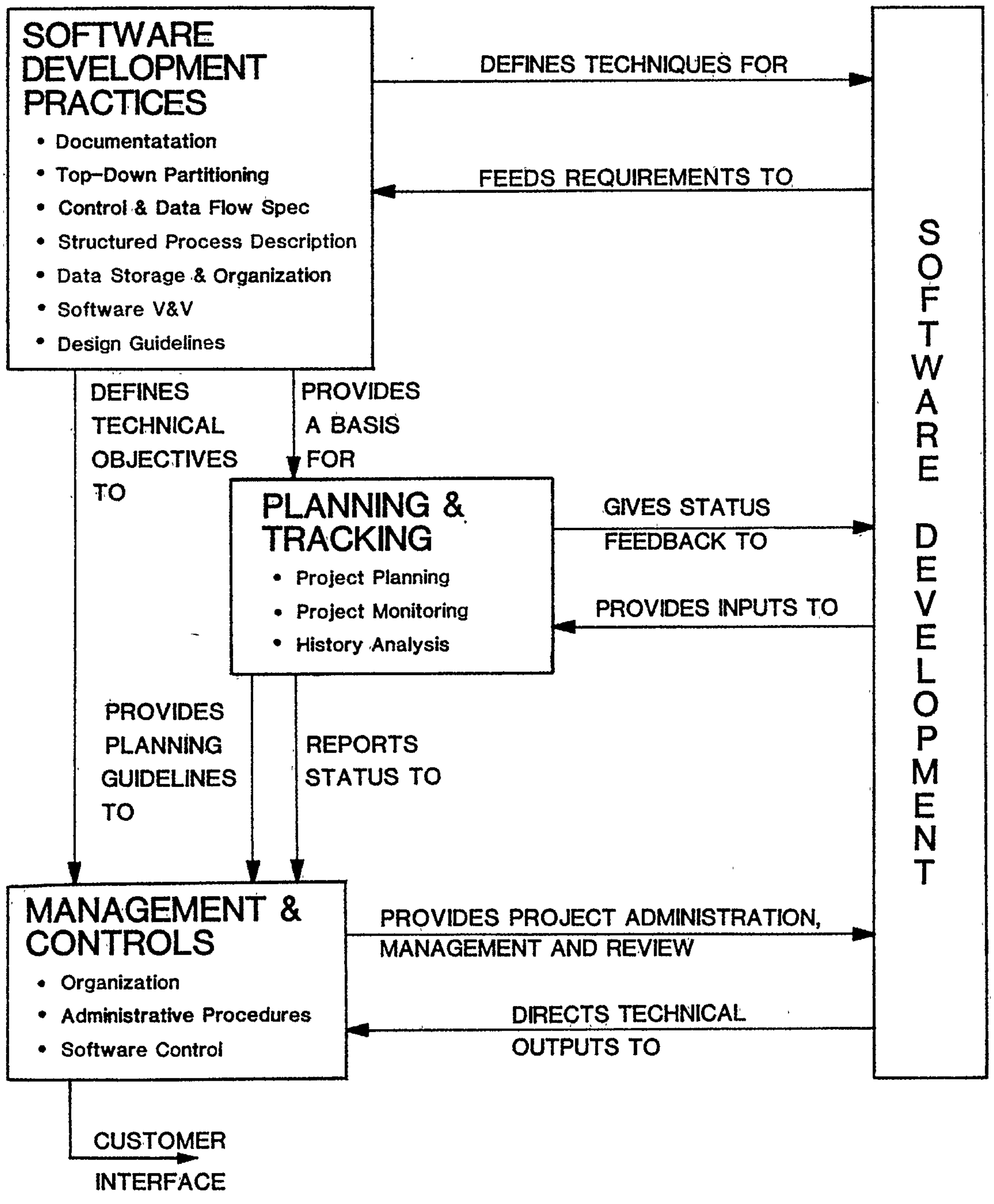

Fig. 2
Integrated Software Methodology (ISOMET)

provide excellent in-line documentation. The technique may be generally applied as an English replacement for process description applicable to both the Definition and Design Phases.

Data storage and organization details are imbedded within the documentation. A software subsystem verification and validation philosophy has been adopted to progressively verify the result of each phase of test. The testing is organized into distinct phases, progressing from design specification verification to requirements validation.

4.2 PLANNING AND TRACKING

The Software Development Practices provide a basis for the Planning & Tracking of a software project. Planning & Tracking consists of Project Planning, Project Monitoring and History Analysis. These three elements are essential to the success of a software project.

Without a project plan, progress cannot be measured. The "90% complete" syndrome is avoided with proper planning. Project planning establishes the basis for how a project will be managed, assesses the software development to be undertaken, estimates the required resources, defines what will be delivered to the customer, and defines the roles of software configuration and software quality assurance.

A management plan establishes management objectives and defines the project organization and how it will interface with the general organizational structure, supporting organizations, and subcontractors. Responsibilities for each organization must be defined to minimize communication problems.

An assessment of the software development provides a system overview to guide the project, determines potential risks, and defines the technical activities for each phase of the software life cycle. Schedules with detailed milestones are developed from this information.

Resource planning establishes manpower, software development facility and training requirements. The deliverables to the customer must be defined in terms of content and format. Guidelines must be prepared for generating these documents in terms of relating in-house documentation to deliverables and the procedures for preparing them.

The planning activity prepares schedule information with man-hour estimates from the software development team and support organizations that provide a basis for tracking progress on the project. Monthly project monitoring of Cost, Schedule and Status Reporting (CSSR) with an automated system provides the data needed to control a software project. The PPD supports project monitoring by providing an indication of what was accomplished during the month when a storyboard (or set of related storyboards) is completed, the tracking is updated and a milestone is reached. This earned value aspect of CSSR is measured by completion of documentation expected at each level of the Definition and Design Phases, by code produced during the Implementation Phase and by successful completion of phases of the Software Test Phase.

The history gained from previous projects is used to give credibility to the estimation process for a new project. Therefore, data gathering is an essential part of each project.

4.3 MANAGEMENT AND CONTROLS

Management in the context of the personnel insuring that control of the software is achieved includes general management and technical management. The Software Development Practices provide a set of directions for software development to produce a consistent technical package for management review. Planning & Tracking helps management understand the ground rules for the project and provides monthly status report information for control.

Overall software control is achieved through a reporting system that tracks progress informally on a weekly basis and on a formal CSSR basis monthly. The monthly CSSR meetings address cost, schedule and technical status based on information generated during the planning process. Normally the basis for cost and schedule determined during the planning phase is not updated unless changes of scope modify the work to be performed. The weekly meetings are brief status reviews which concentrate primarily on any problem areas that should receive management attention.

The technical integrity of the software product is controlled through a series of design reviews during each phase of the life cycle. The design review process is progressive, consisting of reviews within the software organization, reviews with interfacing organizations, formal project reviews and special customer reviews. The reviews during both the Definition and Design Phases start out with level by level reviews for the higher level design and culminate in structured walkthroughs for specific areas of detail. Code walkthroughs are utilized in the implementation phase prior to release for test. Test results are reviewed for each phase of testing.

5. SIMULATION LIFECYCLE

While the end product of a simulation task is software, it is by no means a simple software development. Simulation development has the important difference of requiring the production and maintenance of a model of the system being studied.

A model is merely a representation of a system as understood at the time. As the systems engineering process progresses, the model is refined. The technique for managing this refinement while preserving traceability is termed hierarchical modeling. (Rose, 1981; Kumar and Davidson, 1980; Browne, 1975; and others.) It is important to realize that the primary requirement of the simulation software is to automate the model. Hence the software development must accommodate a requirement (i.e. the model) which will change throughout the lifecycle!

Successful modeling and simulation requires a cohesive and highly controlled progression from model conceptualization to simulation operational test and evaluation. Figure 3 illustrates the simulation lifecycle. The close ties to the software lifecycle are driven by two key points:

1. Digital simulations, in any language, are software, and should be treated as such.

2. For lack of a better alternative, the government imposes software controls and deliverables on its simulation contractors.

A successful modeling and simulation methodology must integrate system modeling techniques with those software technologies which have been successfully applied in the industry. As with software, the backbone of the approach is the "document-as-you-go" philosophy, using the Progressive Project Document (PPD). This concept defines an organized modeling and simulation software development notebook which grows as the project progresses. The PPD provides the vehicle in which to document and to manage a model which is in a constant state of refinement. The software PPD is a "living document" which has been modified for simulation purposes consisting of the following volumes:

1. System Overview
2. Project Management
3. System Model
4. Simulation Operation
5. Simulation Requirements
6. Simulation Design
7. Simulation Implementation
8. Interfaces
9. Test/V&V

While maintaining the basic structure of the software PPD, the modeling aspect molds the documentation to suit the purpose. Since the PPD is adjusted to suit the modeling activity, government CDRL's likewise will take on new perspectives.

Planning the simulation project must occur early in the project and completely detail the activities to follow. Not only is the project plan based on key deliverable milestones but usually upon a parallel system concept and design activity. The two efforts are highly dependent, taking on a symbiotic relationship. The modeling and simulation activity derives it's information from the systems engineering efforts. This systems engineering however requires analytic information from the modeling and simulation task in order to evaluate tradeoffs key to determining a baseline design.

This close relationship must be carefully planned with numerous intermediate milestones. Information must be incrementally documented and released to the modeling team. Analytic feedback points must be clearly defined as to scope and level of detail. Care must be taken to insure that information necessary for later feedback is obtained on a timely basis. If this process becomes overly complex, PERT analysis is suggested.

The engineering methodology is likewise integrated with a management structure customized for model developments. Here the concepts of the systems engineering team, the simulation work package and the software chief programmer team are intertwined.

The model definition and simulation requirements are conducted by a single multi-disciplined team. This team contains expertise in the application system, modeling, mathematical analysis, software and simulation languages. As such, the team for a large project may include personnel from three organizations: systems engineering, modeling and simulation and software development.

This team will remain as the single organized entity through preliminary design. At this time, software work packages will be created which parallel the software partitioning. The initial team remains to continue the model refinement. This structure, as shown in Figure 4, will remain through verification of the simulation. Additional functional groups are added as needed. All teams throughout the life cycle are supported by the necessary secretary, librarian, administration, analysis and programming functions.

It should be noted that while the organization above is oriented toward large program management, it can be used intact for smaller ones: the difference being simply in the number of personnel for each function. A small project may not require a fully staffed management team, but participating roles should remain intact. In fact, it is not the organizational structure which is important, it is the clear definition of the various roles which must be performed which leads to project success.

Rigorous controls are enforced throughout the life cycle. Documented procedures for everyday operations are augmented by numerous team reviews at the various levels of model, functional and software decomposition and refinement. Internal systems engineering and external customer reviews add further controls, while ensuring concurrence on the model throughout the development.

Project cost and schedule are tracked with the CSSR system which not only reports on actual expenditures versus plan but also on earned value. The system provides the complete visibility necessary to keep the simulation development on track. The problem of a large number of small impacts, frequently detected late in the project, may be flagged early using this system. This is particularly important to the control of model changes, where the problem frequently occurs.

5.1 MODEL DEFINITION AND SIMULATION REQUIREMENTS PHASE

The initial definition phase for typical software developments accomplishes two primary functions: to plan in detail the management and engineering activities of the project and to determine the functional requirements for the software. For simulation software, the functional requirements may be summarized as to:

1. Accept the required user inputs
2. Automate the system model
3. Output the desired statistics

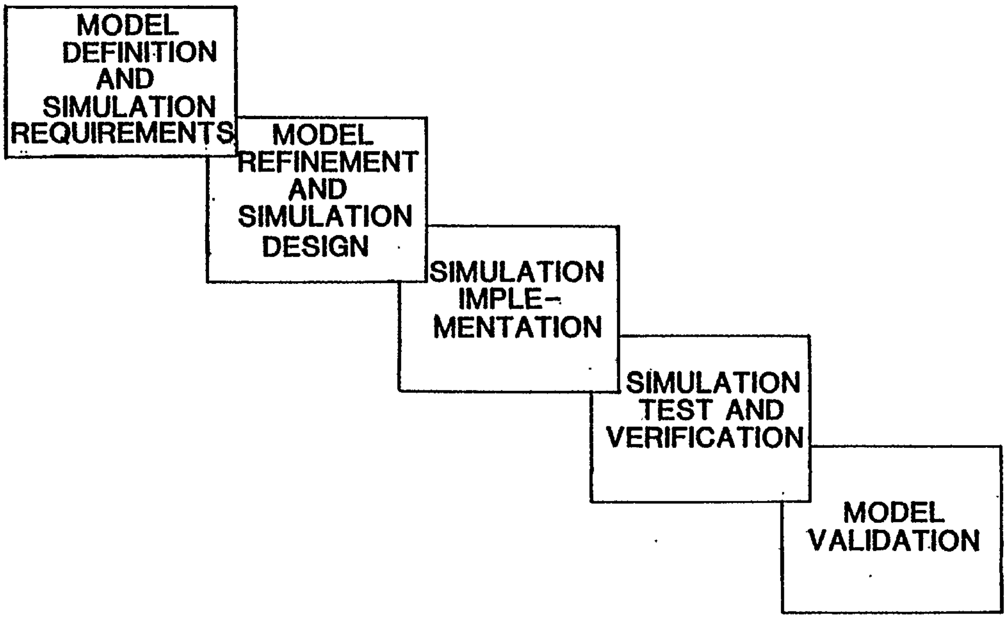
MODEL DEFINITION AND SIMULATION REQUIREMENTS
MODEL REFINEMENT AND SIMULATION DESIGN
SIMULATION IMPLE-MENTATION
SIMULATION TEST AND VERIFICATION
MODEL VALIDATION

Fig. 3
Modeling and Simulation Lifecycle

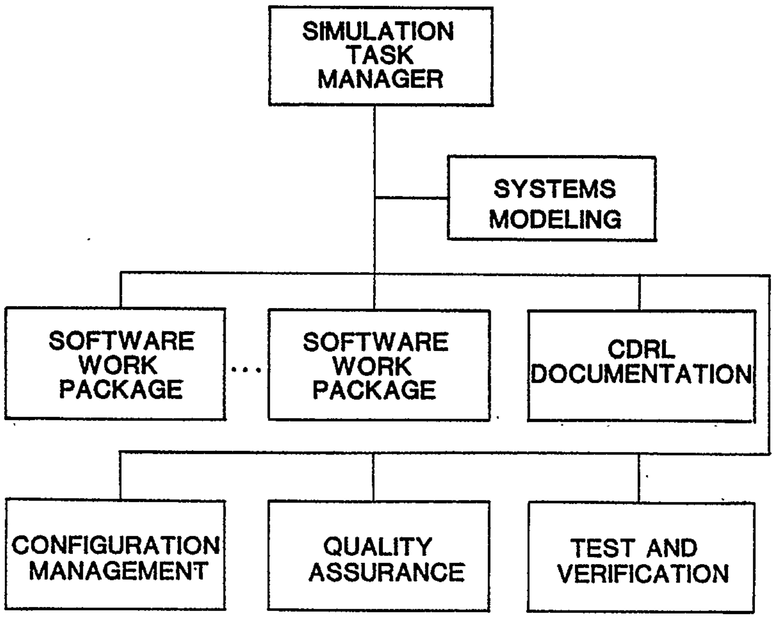

Fig. 4
Simulation Project Organization

The functional requirements for simulation software may therefore be specified by detailing required inputs and outputs and by defining the system model. This emphasis to the model specification is frequently lost in the production of government standard software requirements documents (e.g. B-5) for simulations.

Model specifications must serve as the media of communication between the systems engineers and the simulation team. Unless this specification is easily understood by all, important details may remain erroneous. Typically model specifications are heavily dependent on the simulation language. While this is a natural means for unambiguous specification for the simulation analyst, it is frequently beyond the level of total comprehension for the customer and the systems engineers.

The solution to this problem may be derived from software technology where the concepts of iterative decomposition (or partitioning) and a Program Design Language (PDL) are utilized to both structure the design process and to make it understandable to others. The partitioning process has an added benefit for modeling in that it provides a means for coping with varying and evolving levels of detail. This important concept is termed hierarchical modeling.

The modeling methodology is structured to first partition the model in a level-wise fashion until the desired level of detail is reached. Each level has traceability to the inputs, outputs and control flow of the higher level component. Informal reviews at each level provide an efficient means for checking the model definition. At the lowest level a "structured English" specification of the model element is provided. This specification is a derivative of the software PDL which has been augmented with keywords meaningful to both the application and the analytic structures. By doing so, a Model Specification Language (MSL) which is both unambiguous and readily understandable may be taylored for the application.

This method of structured model specification integrated with the software methodology results in this initial phase being highly organized and manageable. More importantly a documented model baseline is set and agreed upon as the basis for future phases.

This phase also typically results in the selection of a simulation language. With the model specification essentially being language independent, more careful consideration can be directed to the language choice. Issues such as flexibility, host-ability, cost effectiveness and applicability can now overcome personal prejudices in the selection process. It should be noted that in general, this methodology tends to de-emphasize the simulation language.

Model Definition and Simulation Requirements Phase Summary:

Project Planning
- o Project Management
- o Requirements Assessment and Risk Analysis
- o Project Schedule
- o Phase Activities
- o Review and Reporting
- o Manpower/Training
- o Development Facilities
- o Contract Data Requirements List (CDRL)
- o Software CM/QA Procedures
- o Computer Resource Estimates
- o Simulation Language Choice

Model Definition
- o Model Component Partitioning
- o Model Element Specification - MSL
- o Workload Model
- o Parameter Estimates
- o System Cross-Reference

Simulation Requirements
- o Input Definition
- o Output Requirements
- o Functional Decomposition
- o Interface Specification
- o Operational Scenarios
- o Information Storage Requirements
- o Performance Requirements

5.2 MODEL REFINEMENT AND SIMULATION DESIGN PHASE

Establishing a baseline model definition is key to managing a simulation effort. When the simulation is being used as a design tool, two complicating factors typically surface:

1. The model baseline will change and require further refinement in detail.

2. The systems engineers will require preliminary simulation results prior to completion of the end software package.

Unless strictly managed, such a situation can literally run out of control. The tendency frequently is to concentrate on the design changes and output requests, while losing sight of the overall simulation development.

Here again the solution to this problem may be derived from software engineering. Surviving an environment of rapidly changing requirements (in this case, the model) is achieved through a strictly configuration managed baseline. This approach, aside from providing stability, allows two important "thought processes" to occur:

1. On whether one really wants to include this change in the baseline, or is it still only a consideration.

2. Is this change in the scope of this development, or is it a new area not previously considered.

It should be noted that the earlier introduced project organization which is formed during this phase is well suited to handle this case. The Systems Modeling team maintains the model baseline, filtering only approved changes to the rest of the group.

The problem of producing interim results while proceeding with the development may be coped with through the science of software prototyping. Under the rules of prototyping, this interim software must not be utilized in the end product. It may however be used to investigate techniques and approaches which may be incorporated into the design. By producing interim results from software prototypes the software development process may proceed in parallel while also benefitting in the long run from the process.

The actual software design process proceeds along the lines of standard developments. The software is successively partitioned with informal reviews at each level. A Program Design Language (PDL) is utilized to specify processing details at the lowest level. Data bases are designed with heavy emphasis on data typing using a data dictionary approach. The user interface is completely designed. Test and integration plans are carefully formulated for future phases.

Model Refinement and Simulation Design Phase Summary:

Model Refinement
- o Baseline Maintenance
- o Refinement via Hierarchical Modeling
- o Simulation Prototyping

Simulation Design
- o Software Component Partitioning
- o Program Design Language Specifications
- o Level-by-level Reviews and Structured Walkthroughs
- o Data Base Specification
- o User Manual
- o Test and Verification Plan
- o Resource Estimation
- o Model/Requirements Traceability

5.3 SIMULATION IMPLEMENTATION PHASE

The two factors of a changing model baseline and interim result requirements usually continue for the remainder of the project. Hopefully however, model changes will decrease in frequency and impact as the system design becomes firm. It is usually possible to anticipate Implementation Phase refinements during Design by appropriate parameterization of the simulation. In any case, model changes must be managed stricter than ever in order to prevent wide spread design changes.

Prototyping for interim analysis may indeed proceed into this phase. However careful planning of the implementation may make subsets of the software available for early use. To accomplish this, unit testing is replaced by top-down testing which results in an integrated subset being assembled early in the phase. Organization of this combined test and integration into well defined "builds" allows early production of core simulation software while producing polished man-machine software concurrent with any interim analysis. The phase is marked by completion of this low level testing and the assembly of the integrated software subsystems.

The point should be made that even specialized simulation language "code" must be treated as other software. Good programming practices may be followed regardless of language. Any "code" may be structured and highly modularized. If no direct support is provided for this by the language, the intent may be satisfied through standardized conventions. These conventions must be developed and documented prior to entry into this phase. Any prototyping should be used to verify these conventions.

Simulation Implementation Phase Summary:

Model Refinement
- o Baseline Maintenance
- o Refinement via Parameterized Inputs

Simulation Implementation
- o Top-down Implementation and Testing
- o Build Scheduling
- o Structured Programming Practices
- o Code Walkthroughs
- o Code Configuration Management
- o In-Line UDF (Unit Development Folder)
- o Acceptance Test and Verification Procedures
- o Validation Plan
- o Product Baseline Established

5.4 SIMULATION TEST AND VERIFICATION PHASE

As implied by the organizational structure, this phase may be conducted by a separate team within the group. Using the model specification and simulation requirements as input, test procedures may be formulated. Experience in the software industry has in fact shown that best results are obtained when testing is accomplished by a team independent from the development group.

This approach lends additional formality to the procedure. Interfacing between test and development groups is accomplished via Software Trouble Reports (STR's) and configuration managed test libraries. Simulations are in fact more complex than most application software and as such require more thorough and formal testing. Verification of the simulation to the model is an added burden placed upon the testing team. Here too, an easily understood model definition is key.

Upon completion of this phase, the software simulation should be reliable and entirely useable. The final step - validation, must however be completed prior to final release of the package.

Simulation Test and Verification Phase Summary:

- o Software Functional Test to the Requirements Document
- o Verification of the Simulation to the Model Specification
- o Formal Trouble Reporting and Resolution
- o Configuration Management
- o Functional Configuration Audit (FCA)
- o Software Quality Assurance
- o Preliminary Qualification Testing (PQT)

5.5 MODEL VALIDATION PHASE

Division of verification and validation into separate phases is based upon their difference in both methods and organization. While verification may be done by an independent group, validation should be conducted by the Systems Modeling team. It is in this group that the in-depth systems knowledge lies. It is also from this group that earlier system and/or field studies were conducted. These studies are typically used to validate the simulation model.

The methods and extent of validation are deeply dependent on the circumstances of the individual development. More heuristic approaches must be used if neither the system nor a representative environment are available. If on the other hand, such are available, system and environment studies may be extensive. It is for this reason that complete planning of the Validation Phase must be done at the start of the project.

While differing in methods, the control of the validation process must follow along the lines used in earlier testing. Upon completion, a physical configuration audit verifies the consistency of the delivered software and its documentation.

Model Validation Phase Summary:

- o Validation of the Simulation Model to the System
- o Formal Trouble Reporting and Resolution
- o Configuration Management
- o Physical Configuration Audit (PCA)
- o Formal Qualification Testing (FQA)

6. CONCLUSIONS

Modern software methodologies applied to modeling and simulation have proven to be effective in our environment.

Harris' unique approach to a comprehensive software development environment integrating Software Development Practices, Planning & Tracking, and Management & Controls in conjunction with the Progressive Project Document (PPD) has increased productivity and lowered the risk for software development.

Simulation projects applying the methodology have been successful, resulting in:

- o Increased productivity on the order of 2:1
- o A highly visible development process
- o Superior documentation
- o High quality code
- o A well-received product

While the methodology has definitely shown its potential, it is neither perfect nor complete. Benefits to date have been achieved in three short years. While the key techniques and tools are already in place, further refinement and additional automation are planned for the next five years.

Modeling and simulation will remain a special skill which advanced software methodologies will enhance.

ACKNOWLEDGEMENTS

The authors wish to acknowledge Cas Mendez-Vigo, Director, Computer Systems Department for his continued guidance and support. Additional recognition is due to the numerous other individuals who, with patience and hard work, have helped us achieve the level of success currently enjoyed.

REFERENCES

Annino, J.S. and Russell, E.C. (1979), The Ten Most Frequent Causes of Simulation Analysis Failure - and How to Avoid Them, In: Simulation, June, pp 137-140

Automated Data Systems Documentation Standards, 7935.1S (1977) Department of Defense, 128 p

Beauchamp, J.N. and Field, R.C. (1979), Simulation Modelling by Stepwise Refinement, In: Proceedings of the Winter Simulation Conference, IEEE

Brooks, F.P., Jr. (1975) The Mythical Man-Month, Addison-Wesley

Browne, J.C., et. al. (1975), Hierarchical Techniques for the Development of Realistic Models of Complex Computer Systems, In: Proceedings of the IEEE, Vol. 63, June, pp 966-975

Clema, J.K. (1980), Managing Simulation Projects, In: Simulation With Discrete Models: A State-of-the-Art View, T. I. Oren, C. M. Shub, P. F. Roth (eds.), IEEE, pp 235-241

Computer Software Life Cycle Management Guide (1979), NALEXINST 5200.23, Naval Electronic Systems Command

Cutler, M.M. (1979), Proving Properties of Simulation Program for System Verification and Validation, In: Proceedings of the Summer Computer Simulation, pp 610-616

Dahl, O.J., Dijkstra, E.W. and Hoare, C.A.R (1972), Structured Programming, Academic Press, 220 p

Davies, N.R. (1976), A Modular Interactive System for Discrete Event Simulation Modelling, In: Proceedings of the 9th Hawaii International Conference on Systems Science, pp 296

Jones, V.E., et. al. (1980), Final Report of the Software Acquisition and Development Working Group, Department of Defense

Kay, I.M. (1972), An Over-the-Shoulder Look at Discrete Simulation Languages, In: Proceedings of the Spring Joint Computer Conference, IEEE, pp 791-798

Kiviat, P.J., Villanveva, R. and Markowitz, H.M. (1973), Simscript II.5 Programming Language, CACI, Inc.

Kumar, B. and Davidson, E.S. (1980), Computer Systems Design Using a Hierarchical Approach to Performance Evaluation, In: Communications of the ACM, September, pp 511-521

Management of Computer Resources in Systems (1975), AF 800-14, Department of the Air Force

Mathewson, S.C. and Allen, J.A. (1978), A Commentary on the Proposal for a Simulation Model Specification and Documentation Language, In: Proceedings of the UKSC Conference on Computer Simulation, April, pp 158-167

McLeod, J. (1978), Ways to Improve Management of Computerized Models: Theme and Variations, In: Simulation, November, pp vii-x

Metzger, P.W. (1981), Managing a Programming Project, 2nd ed., Prentice-Hall, 244 p

Military Standard Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs, MIL-STD-483 (1970), Department of Defense, 119 p

Military Standard Specification Practices, MIL-STD-490 (1968), Department of Defense, 77 p

Military Standard Technical Reviews and Audits for Systems, Equipments and Computer Programs, MIL-STD-1521A (1976), Department of Defense, 105 p

Military Standard Weapon System Software Development, MIL-STD-1679 (1978), Department of Defense, 28 p

Montgomery, J.D. (1981), Integrated Software Methodology: ISOMET, Harris Corporation, 10 p

Newton, O.L. and Weatherbee, J.E. (1980), Guidelines for Documenting Simulation Models: A Review and Procedures, In: Simulation with Discrete Models: A State-of-the-Art View, T.I. Oren, C.M. Shub, P.F. Roth (eds.), IEEE, pp 243-258

Ören, T.I. (1978), A Personal View on the Future of Simulation Languages, In: Proceedings of the 1978 UKSC Conference on Computer Simulation, IPC Science and Technology Press, pp 294-306

Ören, T.I. (1979), Concepts for Advanced Computer Assisted Modelling, In: Methodology in Systems Modelling and Simulation, B.P. Ziegler, et. al. (eds.), pp 29-54

Ören, T.I. (1980), Concepts and Criteria to Assess Acceptability of Simulation Studies: A Frame of Reference, TR 80.03, University of Ottawa, 46 p

Ören, T.I. and Zeigler, B.P. (1979), Concepts for Advanced Simulation Methodologies, In: Simulation, March, pp 69-82

Pritsker, A.B. and Pegden, C.D. (1979), Introduction to Simulation and SLAM, John Wiley & Sons, 588 p

Program Design Language Standard (1979), Harris Corporation

Progressive Project Document (1979), Harris Corporation, 58 p

Rose, L.L. (1981), Hierarchical Modeling in GASP, In: Record of Proceedings of the 14th Annual Simulation Symposium, R.M. Huhn, E.R. Comer, F.O. Simons, Jr. (eds.), Annual Simulation Symposium, pp 199-213

Roth, P.F., Gass, S.I. and Lemaine, A.J. (1978), Some Considerations for Improving Federal Modeling, In: Proceedings of the Winter Simulation Conference

Ryan, K.T. (1979), Software Engineering and Simulation, In: Proceedings of the Winter Simulation Conference

Schriber, T.J. (1974), Simulation Using GPSS, John Wiley & Sons, 533 p

Shannon, R.E. (1975), Systems Simulation: The Art and Science, Prentice-Hall, Inc., 387 p

Software Design Guide (1979), Harris Corporation, 74 p

Spiegel, M.G. (1980), Prototyping: An Approach to Information and Communication System Design, In: Simulation with Discrete Models: A State-of-the-Art View, T.I. Oren, C.M. Shub, P.F. Roth (eds.), IEEE, pp 219-232

Tausworthe, R.C. (1977), Standardized Development of Computer Software, Part 1: Methods, Prentice-Hall, 379 p

Top Down Design and Structured Programming (1974), Military Airlift Command

Tracey, J.R., Rugh, D.E. and Starkey, W.S. (1965), Sequential Thematic Organization of Publications (STOP), Hughes Aircraft Company, 40 p

Wagner, H.M. (1970), Principles of Management Science, Prentice-Hall, pp 541-565

Wider Use of Better Computer Software Technology Can Improve Management Control and Reduce Costs (1980), Report FGMSD-80-38 to the Congress of the United States, 57 p

Yourdon, E.N. (ed.) (1979a), Classics in Software Engineering, Yourdon Press, 424 p

Yourdon, E.N. (1979b), Managing the Structured Techniques, Prentice-Hall, 266 p

Zeigler, B.P. (1976), Theory of Modelling and Simulation, John Wiley, 435 p

Zeigler, B. P. (1980), Concepts and Software for Advanced Simulation Methodologies, In: Simulation with Discrete Models: A State-of-the-Art View, T.I. Ören, C.M. Shu , P.F. Roth (eds.), IEEE, pp 25-44