ROLANDS & ASSOCIATES Corporation

Approaches and Aspects of Implementing a Computer Wargame Simulation

A Historical Perspective

January 1989

Dr. R. J. Roland
Mrs. E. F. Roland
Mr. E. P. Kelleher

The work described herein was conducted by ROLANDS & ASSOCIATES Corporation for Research & Development Associates (RDA), Marina del Rey, California, under BOA/Subcontract No. 89-7788-0183, dated 9 May 1988. It was performed for RDA as part of its support for the Jet Propulsion Laboratory (JPL), Pasadena, California, in the development of Analysis and Training simulation systems. Permission for its release has been given by the JPL.

Preface

In mid 1988 ROLANDS & ASSOCIATES Corporation (R&A) was approached by NASA's Jet Propulsion Laboratory (JPL) to document R&A's work and experience in the development and implementation of combat models. In particular, JPL was interested in the efforts expended on the Joint Theater Level Simulation (JTLS) combat model and any insights learned that would be useful for future research in combat simulation development.

R&A was, and remains, well suited to provide such insight because of our business focus on the design, development and implementation of combat and conflict models and our technical background that includes advanced degrees in operations research, computer science and organizational behavior. Our engineers have been responsible for the design and development of a variety of models for both the public and private sectors including the conceptual design, specifications and software for JTLS.

This document reflects some of the experiences, positive and negative, of the authors' as related to our work in simulation and modeling. JTLS is used as the primary example because of JPL's interest, the wide distribution and use of JTLS, and because it illustrates many of the concepts used in current combat modeling techniques. No effort has been made to "legitimize" our views by including an extensive bibliography or to minimize our shortfalls. We take credit and criticism as are due. Written in a "straight from the shoulder", non-academic style the ***Approaches and Aspects of Implementing A Computer Wargame Simulation*** is presented as a historical perspective on the design, implementation, and continued maintenance of a major combat simulation.

The responsibility for the content of ***Approaches and Aspects of Implementing A Computer Wargame Simulation*** rests solely with the authors. Information concerning the current uses of JTLS may be obtained by contacting the Commander, Joint Warfighting Center, Fort Monroe, Virginia.

_________________________Date:

Ronald J. Roland, Ph.D.

President

1.0 INTRODUCTION

1.1 PURPOSE

In September of 1981, Professor S. H. Parry of the Naval Postgraduate School (NPS) was invited by the commander of the newly created U. S. Readiness Command (USREDCOM), General Starry, to visit the USREDCOM headquarters in Tampa, Florida. The primary purpose of the visit was to review USREDCOM's capability to conduct joint operation plan evaluations, and advise General Starry on his options to improve the staff's position in this necessary analysis area. Professor Parry's final conclusions were that there was no existing wargame or analysis tool, developed in enough detail, to model the complete spectrum of joint warfare requirements in a coordinated manner. Professor Parry noted that there were a few good starts at developing analysis tools which modeled the interactions of both air and land, but many of these existing models were only a few years old and were not robust enough to handle the complexities required to conduct a full operation plan evaluation. Furthermore, none of the existing models attempted to include the naval and amphibious aspects of joint operation plans.

As a result of Professor Parry's review, USREDCOM established the initial requirement for a joint wargame that could be used for operation plan analysis and evaluation. Since the fall of 1981, the project has gone from upgrading an existing air and land combat model, to the development, testing, fielding and coordinated improvement of two completely new models.

The Jet Propulsion Laboratory (JPL) has played a major role in the development of both new models, the Joint Theater Level Simulation (JTLS) and the Joint Exercise Support System (JESS). They are currently considering taking the next major step in the development of a follow on model that uses many new computational capabilities which have been developed over the past five years. Prior to undertaking that task, JPL has asked ROLANDS & ASSOCIATES Corporation (R&A), the contractor responsible for much of the design and combat model implementation of JTLS, to step back and reflect on the lessons learned over the past six year model development cycle. This report represents the views and conclusions of R&A senior analysts and management staff on this subject.

1.2 PROJECTS REVIEWED

Over the previous six year period, R&A has participated in the development of several wargames and simulation models. We have drawn our conclusions and recommendations in this report not only from the JTLS experience, but also from these other projects.

Many of these projects were started after JTLS and used the benefit, or what we thought at the time was the benefit, of some of the lessons learned from JTLS. Throughout this time period, it has become apparent that the grass is not always greener on the other side of the fence. In many cases, our subsequent attempts to solve perceived JTLS development problems have only proven that the perceived problems are not really problems. Therefore, the conclusions and recommendations presented in this report are based on experience gained through several iterations of attempting to develop, manage and field large military analysis tools.

The following is a description and brief history of the various projects which have influenced this report.

1.2.1 Joint Theatre level Simulation

After Professor Parry's initial visit to USREDCOM a contract was let to JPL and a subcontract for R&A to upgrade an existing joint air land combat model called the McClintic Theater Model (MTM). MTM was developed by a single person at the Army War College (AWC) and was written in FORTRAN. It was a fairly easy to follow, well organized program that had numerous simplifying assumptions. It had one major advantage in that it was a working model. Its major disadvantage was a rudimentary air module which, for example, allowed a squadron of aircraft to conduct only a single mission at a time.

The goal of this initial JPL contract was to upgrade the MTM air module to allow for numerous missions to fly from a single squadron. In the process of accomplishing this task, JPL was to analyze the overall MTM structure and determine whether it was robust enough to continue improving MTM to meet the analysis needs of the USREDCOM staff. The initial contract culminated in what has been referred to as the Summer Feasibility Demonstration (SFD). The SFD demonstrated the upgraded MTM program using a partial USREDCOM operational plan (OPLAN).

During the SFD, JPL presented to General Starry, the AWC, and the Army Concepts Analysis Agency (CAA) their conclusions on MTM. These conclusions primarily stressed that the FORTRAN language restrictions associated with MTM would cause problems in the future and that any further improvement to MTM was not advisable. The structure of MTM and the requirements associated with the model were appropriate for the needs of the USREDCOM staff, but its potential for the future was limited because of the host computer system and the non-dynamic memory capabilities associated with the FORTRAN language.

After the SFD, a follow-on contract was let to JPL to start the development of a new interactive joint combat model. The model was to be developed on the Digital Equipment Corporation (DEC) VAX computer system using DEC's virtual memory operating system, VMS. Although it was not decided at the SFD, SIMSCRIPT II.5 was strongly suggested as the language of choice because of its proven record in building combat models at NPS.

JPL and R&A conducted a complete requirements study and developed a conceptual design for what was to become known as JTLS. Once the requirements were completed, the time between project start and the first full user acceptance test was less than 1 year. JPL was responsible for the overall project management, and for direct management of the development of a program to help prepare the large database associated with JTLS and the program required to act as the interface between the players and the combat model. R&A was responsible for the development and coding of the combat model, a program that verified the logical consistency of the data, and a series of software tools that were needed to run and organize the operation of the entire wargame system on the VAX computer.

Since the first user acceptance test, improvement, testing, and fielding of JTLS has continued. In 1987, responsibility for the model was transferred from USREDCOM to the Analysis Directorate of the Joint Chiefs of Staff (JCS/J-8). JTLS is now part of the Modern Aids to Planning Program (MAPP) and is installed at all of the unified commands world-wide for use as an OPLAN analysis tool.

1.2.2 Planning Alternatives for Interdicting National Terrorism (PAINT)

In 1985 a Middle East country contracted with a private corporation in the Los Angeles area to develop a prototype tactical Command, Control and Communication Center. The purpose of this prototype command center was to provide decision makers with a real time capability to track Persian Gulf activity through the use of state-of-the-art automated hardware and graphical situation displays. The contract included a requirement to create an interactive wargame that could run on the command center hardware and provide realistic situations for use as training and exercise support for both decision makers and the command center staff.

R&A was a subcontractor on this project and was responsible for designing and implementing the interactive wargame. The hardware and software language were both chosen by the prime contractor without regard to the resources necessary for the development of an interactive wargame. The selection criteria was based solely on the operational needs of the command center.

The prime contractor chose to use a Charles River computer which is based on a Motorola 68000 chip and used a variant of the UNIX operating system called UNOS. Marketing literature claimed that a Pascal compiler was available on the Charles River, but within a few weeks of starting the project, the company admitted to numerous problems with the compiler. These problems could not be corrected within the project time limits. This mandated that the wargame be written in C, since no other high level language was available on the Charles River computer.

Graphics played an integral role in the command center; therefore, graphics had to play a major role in the wargame. The prime contractor decided not to purchase a commercial graphics package. Instead, they chose to develop (in house) a complete graphics package for the Charles River within the same time schedule as the wargame. Thus, the wargame was required to interface with graphics subroutines which did not exist, but were being defined, developed and tested simultaneously.

This project had many similarities with JTLS in that the system was being developed by numerous agencies over a wide geographic area. On the other hand, it differed from JTLS in that the combat modelers were not the programmers. Design and programming responsibility were divided between various R&A personnel. Furthermore, time was the only scarce resource on the project. Adequate and timely funding was available.

1.2.3 Airland Advanced Research Model (ALARM)

In 1985, NPS faculty researchers began a review of the current state of automated decision making in the Army's corps level combat models. There were three major problem areas noted by the NPS researchers.

a. Existing systemic, closed wargames based their decisions on either a series of decision tables or threshold values which used, as input data, the current state of the combat situation and resulted in a single deterministic decision. There were no attempts to represent the more typical human decision process in which the current situation is accepted as data, the data are projected forward to estimate a possible future situation and, if an unacceptable situation is predicted, various decision alternatives are analyzed to determine which possible solution most successfully alleviates the future unacceptable situation.
b. New battlefield technology has given the ground commander the ability to affect the battle outcome over a larger and deeper area around his forces. Not only does this mean that battlefield commanders need to consider more area,but they need to be more concerned about allocating available fighting resources against a larger and more diverse group of enemy force capabilities. Current combat decision models do not have an available methodology to compare the diverse group of militarily significant objects usingthe same metric. Without a consistent measurement system, decision models can not possibly allocate the limited fighting resources in an optimal manner.
c. The primary roadblock to improving the decision methodology in current systemic models is not the ability to model the process, but the computational limitations of the current hardware that is used for military analysis. The problems associated with developing algorithms and heuristics for robust decision process modeling are not solved, but in order to try any of the current hypotheses, computational capabilities of host computer systems must be increased drastically.

The goal of the ALARM project is to develop a corps level model that can simulate three to five days of battle without the need for human interaction and solve the decision model problems previously noted. During a year and a half period, NPS faculty and students developed several independent methodologies and hypotheses on ways to solve the problems.

NPS contracted with R&A to help develop a conceptual design for a new corps level combat modeling system which solved the identified problems and incorporated the various independent research ideas developed by the faculty and students at NPS. Once the conceptual design was complete, R&A continued to help with the research system by implementing various sub-system prototypes to obtain a better understanding of the computer processing requirements and modeling limitations associated with the design.

The conceptual design combines three modeling concepts that are the primary elements of the ALARM methodology.

a. ALARM decision modules are based on various network structures. Networks were chosen because of the availability of proven mathematical optimization constructs and the direct relationship of networks to numerous military operations. Cartesian space models, such as transportation networks, communication networks and logistic flow networks are used to optimally place forces in the battle area. Time domain networks, an idea obtained from open Soviet literature and similar to a Project Evaluation and Review Technique (PERT) network, are used to make mission and force allocation decisions.
b. A value system, called the Generalized Value System (GVS), was developed to allow ALARM to measure the military value of a diverse group of objects for limited resource allocation decisions. GVS bases the value of an object in terms of the military power it can deliver to the battle over a period of time. For example, the value of a bridge is not a function of the length or width of the bridge, but is measured in terms of the amount of combat power that can cross the bridge and be brought to bear in the main battle area. Once military units have crossed the bridge, the bridge is no longer of any value or at least has lost a majority of its value. Thus, the value of objects is continually changing with time which means that the computation of the value of an object must be tractable. The methodology developed drew its origins from the similarities between the future value of the military object and the future value of money. Exponential functions are used to represent the situation. Returning to the bridge example, the value of the bridge increases exponentially as a military unit approaches the bridge in an attempt to use the bridge as an obstacle crossing point. The closer the unit moves to the bridge, the more committed the unit is to using that specific bridge, and thus the more important it becomes to the enemy to alter the bridge's status.
c. A distributed computer architecture is required to successfully implement the ALARM methodology within acceptable analysis time constraints. As a minimum, the decision model and execution model are separate running processes which are independent enough to operate on different Central Processing Units (CPUs).

The decision model is designed to be decomposed further into separate decision task processes where each decision task represents an individual command level or staff task. For example, there are separate decision task processes for division decisions, brigade decisions, allocation of artillery assets and allocation of air assets. Some of the decisions must operate in a sequential manner, but many can operate in parallel. Finally, within each decision task individual decision alternatives can be analyzed in parallel, and then compared to select the best or most effective alternative.

Unlike JTLS and PAINT, ALARM was designated as a research project from its inception. There were no operational requirement time line or restrictions placed on the project. Although steady progress was required to continue the project's level of funding, there was never a production capable system promised to, or expected by, the funding organization.

1.2.4 Launcher Positioning Decision Task (LPDT)

In 1987, the Defense Nuclear Agency (DNA) became aware of the ALARM methodology. DNA requested that some of the ALARM concepts be incorporated into a model that could be used to test a hypothesis that Soviet high value weaponry could be rendered ineffective by simply increasing the capability to detect the location of the objects. The hypothesis was based on an assumption that Soviet commanders would move these high value targets if they perceived that the targets were detected at their current location and subject to an immediate enemy attack. If this perception was strong enough, the weapons would be dismantled, moved, and reassembled at a different location. Because of limited Soviet maintenance capability, re-detection at the new site could be accomplished prior to resumption of the operational status of the weapon.

In order to successfully test this hypothesis, the Soviet movement decision logic needed to be robust and realistic. Therefore, the movement plan decision module needed to make decisions based on a known future detection schedule, future maintenance schedule and projected movement schedule for other weapon sites. The decision could not be optimal, but needed to be near optimal to test the hypothesis under best Soviet conditions.

The developed model demonstrated the separate decision and execution model methodology of ALARM and within the decision model, used some optimal network algorithms (Kth shortest path) to construct a weapon movement plan. Although the resulting system was not implemented on a distributed hardware system, it was designed to operate within a loosely coupled distributed system that required only the capability to share common physical disk space.

Unlike the other models previously mentioned, the LPDT project had a very narrow scope and specific purpose. The specific data required to conduct the analysis were well defined prior to project initiation.

1.2.5 Vector in Commander - Generalized Value System Version

The U.S. Army Training and Doctrine Command Research Analysis Center (TRAC) has sponsored the majority of the ALARM research. During the continued research effort, the Deputy Undersecretary of the Army for Operations Research was kept informed of the research progress and felt that GVS was a valuable concept. He directed research sponsors to concentrate on developing a GVS based decision capability within the Army's currently accepted production corps level model. R&A was directed to alter the decision making process of the Vector in Commander (VIC) model to make use of the GVS future state exponential estimation procedures, and alternative selection criteria.

This project substantially benefited the preparation of this report because it revisited the problems of taking an existing model and modifying its capabilities within a very short period of time. The method used to achieve the goals of this project were very similar to those used when upgrading the air module for MTM and the SFD. It was dissimilar in that the resulting model was not tested within a major exercise environment as was done during the SFD, but was tested under laboratory conditions with very restrictive assumptions and model capabilities.

1.3 REPORT CONTRIBUTORS

The following R&A engineers contributed to the content and production of this report.

Dr. Ronald J. Roland has provided his thoughts and observations concerning the overall management of the various projects. He was responsible for the contract negotiations, sponsor coordination, budget, scope definition and timely delivery of all of the projects. His experience in developing project requirements, establishing configuration management procedures, implementing software quality assurance measures and guiding an entire combat modeling team were used to draw conclusions about the advantages and disadvantages of various organizational relationships.

Mrs. Ellen F. Roland has provided her thoughts and observations concerning the development of a combat model conceptual design and guiding the technical aspects of the design through coding, testing, fielding, operational evaluation and upgrade. Mrs. Roland was responsible for the conceptual design of all of the models and had technical responsibility during various stages of the JTLS development. She had full technical responsibility for the ALARM project, the LPDT project and the VIC-GVS project. Her insight into the benefits and flaws of the various project conceptual designs are the basis of many of the conclusions included in this report.

Mr. Edward P. Kelleher, Jr. has provided his thoughts and observations concerning the algorithms and existing data structures of the various models. He is the current R&A JTLS technical manager and was the technical manager for the PAINT project. He played a major role in the algorithm definition, programming implementation, and *database development for all of R&A's major software projects. His breadth of knowledge on the current capabilities and future trends within the combat modeling and analysis community were extensively drawn upon to develop the implications and lessons learned over the six year period represented by the report.

1.4 REPORT PURPOSE

This study was designed to have R&A provide a review and analysis of their combat modeling development cycle experience. The report is just that, simply our view and opinions of what to do and what not to do when starting a new, large and complex combat modeling effort. JTLS has been the center of our corporate life for the past six years and is the development effort that has most greatly influenced our professional conduct. We have taken a great deal of time and effort reviewing and discussing the past years' work in combat modeling and simulation, and trying to establish appropriate cause and effect relationships between situations and problem areas that were encountered. To accomplish this with an unbiased eye was impossible.

Many of the situations and conclusions that are drawn throughout this report are based on R&A's perception of the situation. Although we feel that our conclusions are valid, our perception of cause and effect relationships may not be the same as others who worked on JTLS or have differing opinions as to the reason why some problems occurred. Therefore at worst, this report should bring to light the differing perceptions of how JTLS and other combat models progress from problem to problem. At best, this report will help future combat modeling development teams avoid similar problems and stumble on a few new problems of their own. No large software development project is problem free, but with some prior warning the same problems do not need to be revisited.

We have tried to support each suggestion, recommendation and opinion with an example from more than one project. As we produced the report, we became aware that a majority of our opinions were qualified among various tradeoffs. There are only a few situations where our opinion can be explained in terms of "black and white" or "good and bad". Each situation encountered had some type of tradeoff where management, modelers and sponsors had to weigh the needs of the project against the cost and time implications of the decision.

It is much easier to explain some of the things that this study does not do.

1.4.1 Solutions

This study does not have all of the answers. By no means do R&A personnel have the solutions to all of the world's software development and combat modeling problems. There are numerous items from both categories in this report where we have presented conflicting evidence on how to handle the problem situation. Similarly, there are numerous situations discussed in this report for which we offer no solution, only the information that the problem can exist, in hopes that recognition of the problem is half of the battle.

1.4.2 A Unique Approach

This study does not mimic or reiterate the contents of a textbook on proper software development procedures. There are several similarities and differences between large combat model development projects and other large software projects such as database management systems. There are some ideas presented in this report that are contrary to accepted software development practices because we feel they do not apply to combat modeling projects. On the other hand, there are many items noted that can practically be quoted out of development project case studies. We have mentioned these ideas because their applicability was not readily apparent to us at the beginning of the project.

1.4.3 Disclaimer

It is not the intent of this study to lay blame on any one individual or any organization for the instigation of problems, or applaud any one individual or any organization for the discovery of a solution to existing problems. As much as possible, we have tried to remain objective and present the facts, situation and results without drawing attention to the personalities and inter-personal conflicts that occur within any team effort. This report discusses good decisions, poor decisions and those project decisions for which we have truly not decided whether they were good or bad; no matter who made the decision or who was responsible for their implementation.

2.0 PROJECT ORGANIZATION

2.1 PROJECT PURPOSE

In 1979, the Department of Navy formed a working group of experts in simulation and wargaming to determine why models and simulations were not shared among different organizations. After several meetings and numerous months of analysis, the working group broke up from a lack of interest and progress in defining why the situation was true or what could be done to solve the problem. In reviewing that working group's brief activity it appears that much time was spent discussing what could be done to inform organizations of existing models. No time was spent actually trying to identify examples of two or more independent modeling efforts that produced models that could suitably answer the same question. We don't believe there are many such examples.

If a given model is not suitable for use by numerous diverse organizations after it has been developed, a logical question is: Can a complex model, something more than a spreadsheet, be designed and implemented to satisfy two or more distinct organizations when the differences are identified and considered from the very beginning? We believe the answer is NO.

JTLS started out with three program sponsors, each with different objectives and reasons why they wanted a new theater level combat model. The USREDCOM wanted a model to assist in OPLAN analysis. The AWC wanted model that could be used by students enrolled in the senior level course and could present a realistic decision environment under which various strategies, tactics and employment alternatives could be tried. CAA wanted a model that could be used to analyze various force requirement alternatives for theater level conflict. All three agencies had differing views of the manpower resources, manpower familiarity and level of detail that were required to meet their objectives. The end result was that one model could not fulfill all three desired roles.

For this reason, JTLS developed into a model that did not meet CAA requirements and has never been (to our knowledge) seriously considered for use, by that organization. As far as CAA is concerned they wasted their money, and they are probably correct. Unfortunately, because of this problem, JTLS does not get any good press from CAA. In the long run this has been more harmful to the model than their monetary contribution ever helped the project. In addition, it is questionable whether either one of the other two organizations got exactly what they wanted.

They received a model close to what they needed, but JTLS does not perfectly match or fulfill all of their needs either.

Therefore, we strongly suggest that, as much as possible, multiple agency funding be avoided. This sounds expensive and probably is for one organization, but no one model can fulfill the different objectives of different organizations. If two organizations have the same problem that needs to be solved, then both organization have the same tasking, and the monetary savings can not be found in developing a single model for both agencies, but should be found in assigning the individual tasks to a single agency. To try to develop a dual-purpose model leads to conflicting goals, an ill defined purpose and compromises model development throughout the implementation cycle.

This does not mean that a single model can not eventually be used by more than one organization, but we believe a model should be developed with one organization in mind and to solve that organization's specific problems. This helps insure that the sponsor will get exactly what is desired from the modeling effort. After the model is built and operational, other organizations can take a look at the model and determine whether it can be of use to them. In almost all circumstances, the model will only be of use to another organization after modifications. We have never seen an organization select an existing model without qualifying the selection in terms of mandatory changes and future modifications.

Furthermore, none of the original three JTLS supporting agencies adequately specified a detailed or realistic project purpose. For example, the USREDCOM purpose, to conduct OPLAN analysis, is much too general and therefore quite meaningless. When in the long process of developing an OPLAN did the USREDCOM staff expect the model was to be used?

This question was never asked, nor was it ever answered. It was obvious that the logistics staff believed it was after the Time Phased Force Deployment Data (TPFDD) was established, but before force sustainment requirements were determined because they expected to obtain data from the model on that very issue. On the other hand, the air staff expected that supply availability would have been determined, and that there would be an identifiable cause and effect relationship between the targeting of second echelon enemy supply points and success or failure of the plan. This conflict should have been identified early in the development cycle, and would have been, if the expression of the model purpose had concentrated on a more explicit definition.

When defining the purpose of a new model the following types of questions should be asked: What question is the model to answer? When is the model to be used? Who is going to use the model? How is the model going to be used? Why are other available models inappropriate? What data are available? Finally and in as much detail as possible, what data and information are desired from the model?

At all cost, the situation that existed on the PAINT project should be avoided. The model developers were isolated from the client organization. R&A never met, talked to or knew for whom the PAINT model was being developed. There was no way to explore or understand what the end user problems and expectations were for the completed model. All of this information was filtered through the prime contractor who had no knowledge or experience in the development of wargames or simulations.

The requirements for the combat modeling system should be initiated by the sponsor. Taking an idea for a wargame to several agencies and trying to sell the concept is asking for trouble. You are trying to find a purpose for the model instead of building a model for a purpose. It was never clear on the PAINT project whether the end user wanted a wargame or even had a problem that could be solved by a wargame. From what we saw, we believe that the prime contractor sold the concept of a wargame to the customer, and the customer never really developed a concept of how the wargame was to be used. If the end user doesn't believe a model is necessary, the development effort will never be able to obtain a specific definition of the model purpose and at least partial failure is inevitable.

2.2 PROJECT REQUIREMENTS DEFINITION

Despite the fact that there were conflicting goals for JTLS, the pre-development requirements study was extremely well done. Great effort was taken to independently establish the requirements for each organization at separate meetings. We remember specifically informing each organization at these meetings that requirements were being discussed and no promise was being made as to which of the requirements would or would not be fulfilled in the final delivered design or model. The fact that JTLS did not meet CAA's goals did not come as a surprise to CAA. They started to realize this fairly early in the development cycle because of the requirement definition studies and the emphasis placed on formally written and agreed upon model requirements.

We are very comfortable with the strategy used to gather the project's specific requirements and would use the same system over again for our next major project. The JTLS project requirements procedure was broken down into the following four sub-tasks:

2.2.1 Initial Requirements Gathering

This sub-task consists of a meeting with the program sponsor in which the program sponsor outlines what the final model must do. The system developers should come prepared with a series of questions to help guide the discussions, but the majority of the talking should be done by the sponsor. Requirements concerning available input data, data preparation time, computer resources, personnel resources, required expertise, post-execution analysis time, desired cause and effect relationships, modeled level of detail, project cost and project development time should be discussed.

In the case of JTLS, the initial requirements gathering meetings consisted of three independent meetings with the three program sponsors. This helped because project management could concentrate on each sponsoring agency at its meeting without entering into debates about conflicting requirements or expectations. These debates were postponed until the design and requirements briefing. For multiple sponsor projects, such debates are probably inevitable. They can only be resolved by the sponsors, but technical input from the developers is mandatory.

The requirements gathered during this meeting must be put in written form by the developers, and sent to the sponsoring agency for review. It is important that the developers document the formal requirements so they are expressed in the developer's own words to insure that what the sponsoring agency expressed was properly conveyed to and understood by the developers.

The requirements document should be sent to the sponsoring agency for review. Any misunderstandings or mis-communication that took place at the initial requirements briefing can be cleared up during this review procedure.

When creating the requirements document, a formal numeric tracking system should be created. The introduction to the requirements document should very explicitly explain the tracking system and how it is to be interpreted. This was done for JTLS and was an invaluable tool until initial system delivery. At that time, configuration management should be implemented and its system for tracking requested changes takes over.

The only flaw in the JTLS requirements tracking system was that management insisted that the development team continue to use the system during the coding stage of the project. Management's goal was to have each subroutine list the specific requirement that was being fulfilled by the subroutine. This may be a good idea for other types of automated systems in which a subroutine is normally defined as a module of code that accomplishes one and only one task given one and only one type of data record or entity. Combat models are not developed in that manner.

A subroutine in a combat model should fulfill one and only one function but usually requires access to more than one entity because combat modeling is concerned with the interactions between entities and the coordination or conflict between tasks. Thus there was usually no one subroutine in JTLS that could be designated as the subroutine that implemented a given user requirement. Conversely, some low level functions, such as distance computations, contributed to many requirements, but no requirement specified: "be able to calculate distance between two points". The capability is implicitly required for many other functions. Specifying the requirements for both of these situations was troublesome.

The tracking system should be maintained throughout the model development cycle, and is most useful for design reviews, test purposes, and user functional validations. We believe that maintaining the system in any more detail through the coding and model implementation phase of the project is useless, time consuming and does not increase the probability of success for the project.

2.2.2 Conceptual Design Development

After the requirements have been formally written and approved, the design team can create the model's conceptual design. The conceptual design should address every user requirement, but should not attempt to develop a design so integrated that the model will not run until all of the requirements are fulfilled. The design must include all of the requirements, but should allow for modular implementation to satisfy the various requirements. This is probably the most important and strongest suggestion that R&A has for any combat model development team.

Time and time again over the past 15 years, we have witnessed modeling projects that expend a great deal of effort gathering requirements and that promise that all of the requirements will be fulfilled in the initial model release. Every single one of these projects, in our opinion, is either in trouble or has already been abandoned. Projects such as the Air University's CRES project, the Naval War College's Naval Warfare Gaming System (NWGS), the Joint chiefs of Staff Joint Analytic Warfare System (JAWS) project are good examples.

Combat models that are built as follow-on models to already existing models are especially vulnerable to this problem. Usually, the only way to convince sponsoring agencies to fund a follow-on project is by promising them that the new model will do everything the old model does in addition to a whole lot more. Since the new development is a complete redesign of the system (if it wasn't the old model could simply be upgraded), a major coding effort is required to get the new model to perform all of the functions and capabilities of the old system and establish an acceptable initial release.

An initial delivery of anything more than one year, especially when working with military agencies, is bound to cause trouble because of the lack of continuity within the sponsoring agency. Thus the relative lack of time and the enormity of the initial release project leads to failure or at least insurmountable delays. We believe that JESS II can very easily find itself in this dilemma if JPL management is not extremely careful concerning the modular implementation of its conceptual design.

One approach to solving this problem is to severely limit the scope of an initial release. It might provide only the capabilities of the old model, but using the new concepts, structures and language. This version would be specifically designed and implemented to permit the addition of the new model capabilities. The difficulty in selling this approach lies in the fact that, on the surface, at the point of the first release, there is no new capability, although there is new potential. A more palatable approach may be to deliver a very limited subset of the new capabilities, with the "hooks" for the other capabilities in place and not implement unneeded or unused capabilities from the old model. As discussed in the following paragraphs, this should be specifically identified in the schedule for fulfilling the requirements.

Naturally, some of the sponsor's requirements may be contradictory. These need to be identified and addressed by either providing design alternatives or asking for clarification prior to completing the conceptual design. The conceptual design should not discuss specific algorithms or details of model logic. It should address the modeling system as a whole, concentrate on how data flows through the system, discuss how the various program interfaces will work, and describe the expected type of output that will be available.

Finally, the conceptual design should include a matrix of the model requirements that outlines either when or how each requirement will be fulfilled. If the requirement is not fulfilled as part of the initial delivery, the design should address how the design allows for easy upgrade to include the required capability at some later date. A requirements implementation schedule should be included with the conceptual design to insure that the sponsors understand which of their requirements are going to be fulfilled with the initial system delivery and subsequent follow on deliveries.

Requirements concerning detailed model algorithms or logic only need to be listed in terms of whether the model will have that capability as part of the initial delivery or not. Requirements concerning system operation need to be explicitly addressed.

In summary, get something up fast and plan for upgrades. Do not try to create the perfect model from the very beginning. In our experience this advice is the most important difference between the success and failure of a combat modeling effort.

2.2.3 Conceptual Design and Requirements Briefing

The conceptual design should be briefed and discussed with the sponsoring agency. This is not a design review where algorithms and specific model logic are discussed, but a review of the concept of the model and the implementation plan. If, as is usually true, not all requirements can be met in the first release, the sponsoring agency should establish a priority scheme for the requirements in order to redirect which requirements should be included in the initial delivery. Project management must be capable of assessing the implications and tradeoffs of altering the suggested project development plan. Suggestions that the initial delivery be postponed until all requirements can be included should be firmly resisted.

The meeting should end with a mutually agreed upon schedule and a list of initial delivery requirements to be fulfilled.

2.2.4 Finalization of Conceptual Design

Given the changes and agreements established at the conceptual design briefing, a final conceptual design and requirements document can be created. This document should become the basis for the test plan and user functional validations of the future. After this is complete, the design team can begin the detailed design process, which is discussed in more detail under the data structures and algorithms section of this report.

2.3 PROJECT TEAM ORGANIZATION

Building a combat model requires a diverse set of talents seldom found in any one individual, group or organization. Therefore under almost all circumstances, a combat model building project team requires that a lot of management time be spent in putting together a team, organizing the team, and insuring that the team interacts appropriately. In reality a large modeling project should not be a single team, but should consist of a group of teams each with its own specific area of responsibility and expertise.

The following represents R&A's observations and conclusions about the organizational aspects of a modeling group.

2.3.1 Project Management

There are two extremes of manager found in most organizations. One extreme is the manager who is totally devoted to the technological aspects of a project and would not recognize a people problem if confronted with one. The other extreme, is the behaviorally oriented manager who is too busy to be bothered with the detailed technical issues. Both are unsatisfactory for the position of managing the highly technical individuals usually associated with the design and development of combat models and mathematically oriented military simulations.

The project manager (PM) is a critical position to staff. It should be done with the greatest of care and oversight. The PM should have a solid background working "in the trenches" with combat analysis, should understand the design issues and limitations, and should be able to recognize and differentiate among software problems, hardware problems, personnel problems, and system issues. The PM should also have a few years experience within his or her organization in order to understand organization goals, formal and informal communication structures, and responsibilities to the organization. This is of particular importance if the PM is to operate at a location remote from his/her parent organization. The PM must have had direct military experience from which to draw the ability to comfortably interface with the sponsoring agency. Finally, good inter-personal skills and a solid work ethic are mandatory.

This is easy to say, but hard to provide. In retrospect, some difficulty with the development of JTLS was the assignment of an individual with excellent credentials who had no experience or understanding of JPL's management, contracting or technical procedures. He also had had no experience with JPL's project management styles nor the organizational structure. The result was that project personnel were misled concerning their goals and the goals of top level management. Upper management must keep close control and oversight of newly assigned managers to insure they understand their assigned goals, objectives and responsibilities.

2.3.2 Team Definition

A combat modeling project should be divided into well defined sub-functions which can be developed by independent teams. It is easiest to define these sub-functions by evaluating the difficulty of defining and explicitly documenting the interface and interaction requirements between the sub-functions. If the interface can not be precisely defined, then the sub-functions are not independent and should not be given to different teams.

This method was used with JTLS. In the original design, there were four programs, the Scenario Preparation Program (SPP), to build the database for JTLS; the Scenario Verification Program (SVP), to verify that the prepared database was internally consistent; the Combat Events Program (CEP), the actual combat model program; and the Model Interface Program (MIP), to interface between the human players and the CEP.

The SPP was created by one team because the interface between the SPP and the combat model, the Combat Events Program (CEP), was well defined in terms of the scenario initialization file. The same was true for the Model Interface Program (MIP) and the CEP. On the other hand, there was no well defined interface between the CEP and the Scenario Verification Program (SVP). The information required to create the SVP was closely related to the detailed assumptions associated with the CEP. These assumptions could only be defined as the model was built. Thus the same team that built the CEP should have been, and was, responsible for development of the SVP.

This method of organization leads to some potentially risky situations. In particular, the project may become "single point vulnerable", especially if a team consists of only one person or a team is made up of personnel from a single organization. Unfortunately, we believe other methods of organization have side effects that are even worse.

We have observed another organization divide their area of responsibility not by program sub-function, but by model requirement. In this organization a team responsible for an air module change, was responsible for incorporating the data change in the SPP, the orders and order verification change in the MIP, the design and coding of the change in the CEP, the data verification change in the SVP, and the related summary statistics change in the Post-Processor.

This organization felt that by organizing themselves in this manner, they were no longer single point vulnerable. Every team was familiar with, and could change, the SPP, MIP, CEP, SVP and Post-Processor. This was true, but they didn't develop any area experts. No single person or team was responsible for a coordinated upgrade plan for any of the functional areas or individual programs. Each upgrade was developed and implemented independently across all model capabilities. For example, scenario preparation data screens were inconsistent and player order entry procedures were incompatible. In some areas, the lack of wide familiarity led to multiple versions of code with the same function within a single module. During testing and integration it led to patching and excessive "bullet proofing" of the code.

2.3.3 Team Size

R&A is a strong believer in small teams. We can not envision working with a team that has more than five or six people. It is a simple management problem of span of control. Realistically, a manager can not understand what is being done by his or her subordinates if there are more than that number of people accomplishing a task. Likewise, the amount of communication and coordination that must be done within the team grows exponentially as the team size grows. Given that a team is a group of individuals working on a sub-function within which no interface is easily definable, it is imperative that communication be easy to accomplish. As the size of a team increases, the fraction of the total team effort that must be used to maintain effective communication among the members increases. If the team gets too large, the effectiveness of the communication usually decreases, coordination becomes harder to maintain and the quality of the sub-function for which the team is responsible suffers.

If more people are needed to complete a project within the time constraints of the sponsoring agency, the project needs to be divided into smaller sub-functions. This means that more well defined interfaces between the sub-functions are required. If this can not be accomplished, we believe that either the scope of the project should be reduced, or the project should be extended. Adding more people to a project team does not usually help the problem.

2.3.4 Geographic Separation of Teams

Individual team members can not be separated geographically. The members must work together in the same place and at the same time. The PAINT project suffered because the development team was required to work in shifts on the project. This made close coordination impossible. Given that the inherent definition of a sub-function and a sub-function team is that no easily definable interface exists, the inability to conduct close, continuous coordination was disastrous.

Until 1988, the JTLS CEP was developed by a single team. In 1988 the decision was made that the CEP could be divided into two sub-functions that were fairly independent of one another. One team was assigned responsibility for naval and amphibious functions, the other for the air, ground, logistics, intelligence and main modules. This has not worked well. It could have been improved by either better defining the interface between the two sub-functions or leaving the model under the control of one team. Either solution would have alleviated the problems that have been encountered.

For example, the existing model capability to represent intra-theater airlift was used as the basic structure with which to represent the heliborne amphibious lift capability. The two operations are inherently dissimilar. The resulting code is at best clumsy because of the incompatibility of the two operations, and we consider it a high risk area for model stability.

It is possible to develop a model when there is geographic separation between teams, but there are numerous hazards. It takes strong, knowledgeable, and dedicated management control to insure that these hazards do not get out of hand. The JTLS project did not score very high in this area, although the situation did improve with time. The CEP, SVP and several of the support tools were developed by R&A in Monterey while the remainder of the system programs were developed at JPL in Pasadena.

Project delays are a fact of life, as are programming errors. No team wants to be blamed for the delays or errors which can easily degenerate into what we refer to as the "finger pointing syndrome". The reason that Team Z is late is because Team X changed the interface specification or Team Y did not deliver the code required to test an interface.

When this starts to happen two things occur, both of which are extremely detrimental to the project. First, teams will go to any length not to be late because they refuse to allow any other team to lay blame on them. This means that they may deliver code before it is thoroughly tested, that they may broaden simplifying assumptions, or that they may cut corners and ignore problems altogether.

Second, any hope of a close working association of the individual teams is lost. The project becomes a project of confrontation instead of a project of cooperation. The teams become individual entities instead of a coordinated group working to complete a single project.

This problem may be manageable when contained within the overall development team. When the "finger pointing" is permitted to extend outside the development team, to the point where the sponsor becomes aware of it, it is destructive. The concern of the sub-function teams is exacerbated, the push to avoid lateness at all costs increases, and internal tensions grow worse. From the point of view of the sponsor, the problem is an internal one, and is, by definition, the responsibility of project management to alleviate.

Once finger pointing starts, the individual teams can not stop the snowball effect. Project management needs to insure that it never gets off the ground. Some simple rules can insure that "finger pointing" does not occur. Teams should not be allowed to report delays on or after due dates. Team management should realize well before their due date that a delay of more than a day or two is going to occur. When there is a delay, either anticipated or actual, project management should ensure that any affected teams are notified. Those teams should be required to assess the impact of the delay on their delivery schedule, and to evaluate the possibility that a rearrangement of their internal tasking or resources can be made, so as to minimize the overall effect of the delay.

Project management can not be biased in its approval of the other team's reevaluated delivery schedule. Reported schedule impacts should almost never be longer than the delay that caused them. Except in unusual cases, such assessments are indicative of problems in the area of the team reporting the large impact. For example, a reevaluated delivery schedule that predicts a delivery delay of two weeks because of a two day delay in another team's delivery should be unacceptable to project management. If project management allows this situation to occur, finger pointing and team strife are sure to follow.

2.3.5 ORGANIZATIONAL SEPARATION OF TEAMS

It is hard to say whether the cause of the finger pointing syndrome is a geographic separation or organizational separation of the teams since both were true for R&A. Our perception of the situation is that the geographic separation was more to blame, because we saw very little finger pointing between the SPP development team and the MIP development team. They were from different organizations, but were geographically collocated. Therefore, our conclusion is that the communication breakdown caused by the geographic separation is the greater contributor to the finger pointing syndrome than organizational chauvinism or pride.

We have no examples that can be used to analyze what problems may occur when several organizations place personnel on the same sub-function team. In our experience this seldom happens because management of most organizations is reluctant to allow other organizations to obtain the knowledge and expertise associated with the tasking for which they have responsibility.

Other than the possibility that organizational management may not allow different organizations to work on the same team, there are many possible benefits to such a situation. One benefit has already been mentioned in that no one organization can make or break the project by being the single point of expertise on a sub-function. Second, different organizations have different areas of expertise and there are many modeling sub-functions that require a wide range of expertise. It is easier to get this expertise from a variety of organizations that to find a single organization that can provide all of expertise required. Again we are not aware of any such arrangements, and can only hypothesize as to the benefits that would result. We do not know whether these benefits would be outweighed by unknown problems.

2.3.5 Team Composition

Team composition depends largely on the sub-function of the team.

a. Conceptual Design Team

The conceptual design team needs at least one military expert that has detailed knowledge of the current military system that is being modeled. For example, the LPDT model needed an expert in how the Soviets deployed and planned on using the high value targets for which the simulation was built. JTLS really needed a military expert in OPLAN evaluation, and ALARM needed a military expert in the corps level planning and staff operation. In all three of these examples, the military sponsors were responsible for providing this expertise, but were not collocated with the remainder of the team. Although this arrangement was acceptable, communication would have been easier between the design team and the sponsors if a full time expert was part of the design team.

The design team needs at least two combat modelers with military experience. Two modelers are necessary to insure that there is a built in check and balance system in the design concept. Developing a combat model is an art as well as a science. The two modelers are needed to exchange ideas, question each others opinions, and build upon each others concept of operations. The military experience is required to help translate the desires of the sponsor into a workable model. The modelers need to know how to ask questions on what simplifying assumptions are feasible and acceptable. This can not be done effectively if they do not have some military experience on which to base suggestions or ask questions.

The design team needs access to a computer scientist who has extensive knowledge of the current state and capability of available computing resources. Combat modelers usually understand the capabilities of the systems on which they have recently worked, but they tend not to follow or keep abreast of the new technology, especially in terms of hardware capability.

A computer scientist is needed to insure that the conceptual design includes the most advanced computing techniques and hardware configuration that are feasible within the cost and time constraints of the project.

The design team needs a database management or database design expert. The key to a successful model today appears to be the design and distribution of the data required by the model. Data organization, data availability, data access, data creation, data storage and methods to share or pass data between and among processes are all important aspects of model development that must be considered in the initial conceptual design.

JTLS was designed to break the computational requirements into separate processes in hopes that several CPUs could eventually be used and over which the processing could be distributed. Since the conceptual design team did not have a database expert, many of the database access questions associated with this design were never properly investigated. Today JTLS is not CPU bound, but it is limited by the amount of input and output (I/O) required to pass information between the individual processes. A resident database expert might have avoided this problem from the beginning of the project.

b. Combat Modeling Team

R&A does not believe in independent model design and implementation teams. We feel that the design of a combat model is too closely tied to the coding for that to work. Practically every line of code in a combat model adds an assumption to the model. It may be as simple as an assumption that the constant Pi is accurate to six decimal places, or as far reaching as to assert that combat can only occur between units in adjacent hexes.

It is the assumptions between these two extremes that non-modeling programmers unwittingly make and which in turn cause problems. An experienced modeler, who is also programming, has much greater ability to recognize and assess these built-in assumptions, whereas a "coder" may not be aware of the implications of what they believe to be unimportant decisions.

We believe that a perfect combat modeling team would include two modeler/analyst/coders, two medium level programmers and one junior level programmer. One of the two medium level programmers should have extensive computer hardware and software system experience to insure that the modelers fully understood how to efficiently use the development system to accomplish the modeling goal.

The CEP portion of the JTLS project had a fairly good modeling team, but the team wasted a lot of time because of the absence of a good system programmer. The system level programming information was learned by the analysts on the team, but at a cost of time and money that could have been used to better advantage.

The team needs at least two experienced modelers in order that ideas can be easily exchanged. We can no longer count the number of times that an R&A modeler has had a problem in developing an approach to solve a modeling problem, and a second team member has opened the door with a different approach or unique idea. One person can not develop a good model in isolation.

As with the design team, including two modelers on the development team provides another system check and balance which is invaluable to the success of a project. The phrase "I wouldn't have done it that way" is heard often around R&A and whether it results in a methodological change to the problem's solution is immaterial. It has caused both experienced combat modelers to view the problem from a different angle and consider different sets of decision criteria.

c. Support Software Team

In almost all cases, support software teams need a human factors specialist to help define a consistent, user friendly interface. They also require at least one senior level programmer and depending on the size of the support software task, several mid and junior level programmers.

We believe that at least one senior level analyst/programmer is required on every support software team so the team is not intimidated by other interfacing teams. R&A is a strong proponent of working managers. Any individual in charge of a programming team should be spending some time designing, coding and testing the software for which the team is responsible.

d. Independent Test Team

An independent test team is required from the very moment testing can start. Model developers need to test their own code, but once that is done an independent test team is a mandatory requirement to complete the process. JTLS did not have an independent test team soon enough, and it showed at the first functional validation and user acceptance test. Coders can only test what they know they have programmed. The problems arise when a user tries to implement a strategy that the designer and developer did not consider.

The test team needs to have some people that are fairly knowledgeable about military strategy, doctrine, tactics, and the type of combat that is being modeled. In addition, individuals familiar with testing are an extremely valuable asset as are testers that grow along with the model. An experienced tester can speed up the development of an upgrade by the very fact that they understand the system, database and have developed a working relationship with the model coders.

2.3.6 Project Team Communication

There are three types of project team coordination.

a. Intra-Team Communication

We have already stated that a team should work side by side because communication among the team members must be constant. If this does not happen, the team can not accomplish its task in a coordinated or efficient manner. Intra-team communication can not be limited to meetings, but must occur on an ad hoc basis when and where it is needed. Each team member must have the ability to quickly access and coordinate with other team members on any problem, decision or possible conflict. The items that need to be discussed and the decisions that must be made can not all be foreseen prior to a daily or weekly meeting and usually can not wait for the next team meeting.

b. Inter-Team Communication

If the teams belong to different organizational entities, much effort should be taken in insuring that informal communication channels exist. Early in the JTLS development cycle, inter-team communication went from the CEP development team through local management to headquarters management and then down to system level programmers. Answers came back the same way. This kept management informed of interface problems, but it slowed the progress in finding solutions to those problems. As the JTLS project continued, and the teams got to know each other, this changed. The direct communication among the teams helped project continuity, reduce the amount of finger pointing and led to more of a feeling of project camaraderie.

c. Team and Sponsor Functional Representatives

Whether to allow direct communication between development team members and area experts from the sponsoring organization is the hardest communication decision that a combat modeling effort must make. Hopefully, the sponsoring agency does not unilaterally make the decision by refusing to name a functional area expert or by providing an on-site dedicated functional area expert. Either extreme can spell disaster.

It is very important that modeling team members have quick, easy access to functional area experts at the sponsoring agency. We truly believe that practically every line of code written for a combat model contains a modeling assumption. When a modeler recognizes that a major assumption is about to be made, or has a choice of two or more modeling approaches, he or she needs to have a sponsor representative available to discuss the pros, cons and implications of the decision. The modeler is the best person to explain the situation and can enter into a knowledgeable exchange with the functional area expert. If this type of exchange is conducted through channels, there is bound to be information lost as part of the transmission.

On the other hand, direct communication between modelers and area experts decreases management's control of the project. Modelers tend to want to make their models as realistic and robust as possible. Functional area experts tend to overlook or reject the concept of simplification of reality for analysis purposes. Thus, allowing a modeler to talk directly to a functional area expert can create a situation in which increased detail, costing time and money, may be promised to the sponsor without the approval or knowledge of project management. There is a fine line between obtaining the benefit of a functional areas expertise and changing modeling complexity to match the desires of the sponsor's functional expert.

JTLS is a perfect example of this problem. There was no functional area expert for the Intelligence module of JTLS. Although the sponsor assigned one, the expert made it clear that he did not want to be bothered or consulted. The Intelligence module is now in its third rewrite and still does not closely represent the functions, capabilities or output of a theater level intelligence staff.

For example, the Logistics module had an extremely interested functional area expert. He was continually available and took an active interest in all of the assumptions and model capabilities. The modeler responsible for the logistics capability was frequently on the phone consulting the functional area representative.

The model ended up doing exactly what the functional expert wanted. Unfortunately, this is a two edged sword. Much of the logistics logic is more complicated than it needs to be. For example, the logistics expert insisted on extremely detailed modeling of the intra-theater airlift and airdrop capability. This was not part of the original model requirements. Whenever approached for an opinion on how to handle a situation, the expert always chose the most detailed and expensive solution. The modeler was happy for the guidance and took the word of the functional area expert that the indicated level of detail was required. The functional area expert was happy because the model was being developed exactly as desired.

This went on for some time before management discovered the problem and insisted the modeling detail for Airlift and Airdrop mission be cut back. Project management and sponsor management were far from happy because the increased level of detail started to cause delays and not simply because of extra coding. The more detailed model required more data, more documentation and more testing.

The continuous availability of a functional area expert can easily lead to problems that are as bad as or worse than the Airlift/drop problem. The ready accessability of the expertise makes the interface problem simpler. Unless a great deal of care is exercised, it may make it too simple, to the point that the functional area expert drives one particular area of the model to a level of detail that is incommensurate with the rest of the model. On the other hand, the simplification of the interface can improve a functional area, provided judicious restraint is used by the functional area expert and the modeling team.

Many management books and some Government regulations prescribe solving the dilemma by proposing that all such modeling choices be explicitly presented and discussed during design reviews. That is a commendable goal, but is simply impossible. The number of decisions to be made and options from which to choose when developing a major combat model can not all be planned for. Even if they could, there would not be enough time to document, explain, create alternatives, and discuss all of the assumptions and various options at a design review. We do not have a solution to this problem. We are convinced that easily accessible functional area experts are essential. We do not know how to identify and preclude the modeler and area expert from crossing over the line from guidance to redefinition of capabilities.

This is an area of delicate balance and one that presents the greatest challenge to the professional manager. It represents the two ultimate management choices. One is to proceed and deliver the product to the user in a short period of time by not imposing additional communication and documentation constraints and accepting the calculated risk of some "seat of the pants" changes being made to the original requirements. The other is to impose a mechanism by which all design discussions be documented and discussed formally prior to considering them for inclusion into the system and assume the risk that the project will be over cost and schedule. The successful manager navigates carefully in these waters and must weigh factors such as the sponsor's sensitivities, schedule constraints, cost and resource availability before making a final decision.

2.4 PROJECT RESOURCES

For the majority of our modeling projects the availability of computer resources has been close to unlimited. None of the computers that we have used for development were CPU bound during the development cycle. We did witness this problem at JPL during the initial phases of JTLS, and were happy to get back to Monterey where we could get some work done. We consider a resource constrained CPU as unacceptable during model development. Each developer needs as least a terminal with windowing capability or two terminals available on which to do development. We find that we are not efficient without at least this minimum arrangement.

We have already mentioned that it is very important that team members have constant and direct communication. To allow this to happen, the idea of separate individual office space is unworkable. On the other hand, we found that requiring personnel to pickup all of their working papers when they leave a common terminal area is disruptive. We believe that larger office spaces in which individual work areas can be accommodated provides the needed environment for constant communication, and allows developers to organize a work area to meet their needs.

Whether the separate teams are geographically separated or not, a simple means of informal communication among the teams is mandatory. Face to face or telephone communication are desirable, but frequently difficult. With the current availability and relatively low cost of electronic mail systems, we believe that they offer an excellent opportunity to improve this type of interface. Because of the required proximity in time and space of the members of a single team, availability of an E-mail system within the team, while still desirable, is less necessary.

We have developed combat models on systems other than their intended host system. The first was the upgrade to MTM which was done in preparation for the SFD. The model upgrade was developed on a DEC VAX/VMS and hosted on a Honeywell 6000 just two weeks prior to the SFD. Although standard FORTRAN was used, there were so many file transfer problems that only a small miracle saved the project from being a total disaster. If it is necessary to develop the software on a machine other than the target machine, one of the first tasks should be to practice transferring files and develop a quick prototype to determine whether there are any unexpected programming language problems.

2.5 CONTRACTING

We have built combat models under three different types of contracts. Although we prefer labor hour contracts, that is not necessarily the best type of contract for building combat models.

2.5.1 Fixed Price Contract

We have had both good and bad luck with fixed priced contracts. The common thread in the acceptable situations is the daily interaction of the contract monitor with the work that is being done. If you have a contract monitor and a contractor that know each other, trust each other and work together daily, it appears as if fixed price contracts can be very workable. The problem is that when dealing with large organizations, project personnel, on both sides, can change. If this happens and the working relationship is altered, there is apt to be trouble.

This is our perception of what happened in our second JTLS contract. The first two contracts issued on the JTLS project were fixed price contracts. In both circumstances, the tasks that needed to be done were defined in general terms. In both cases, details concerning the model requirements and design were not specified as part of the contract.

The first contract was for the upgrade of MTM and support for its demonstration at the SFD. In a ninety day period, more than twenty thousand lines of FORTRAN code were added to the existing MTM model by three modeler/programmers. This represented an average of more than one hundred lines of code per workday per programmer. A 1988 article on simulation in a modern programming language speaks glowingly of programmers attaining high productivity levels of as much as twenty-five lines of code per day. This 1982 effort produced an order of magnitude more than industry standards at the time. Even though the effort resulted in a system that was not fully tested and operational, the magnitude of the effort involved was recognised. All the team participants were happy and there was no question that the sponsors were well satisfied with the results.

The second contract was not a good experience. One quarter of the way through the contract, the contract monitor was changed. The project requirements grew. The fixed price contract became a problem for R&A, and the inability to produce the model for the contracted amount became a problem for JPL. Clearly, some of these problems were caused by extensions of model capabilities, as discussed above, both with and without approval from project management.

The ALARM, LPDT and VIC-GVS contracts were all performed under a fixed price contract without any trouble. We believe that this is possible because of the extremely close working relationship with the contract monitor, even though these models and requirements were not well defined at the beginning of the projects.

2.5.2 Labor Hour

The remainder of the JTLS contracts and the PAINT project were labor hour contracts. From a contractor's point of view we had no problem with this type of contract, but having been on the contracting agency side of the table prior to starting our own business, it is fairly risky. A contract monitor must insure that there are some well defined deliverables. Unlike a fixed price contract this does not need to be decided on prior to contract award, but it does need to be decided as the project progresses. Without such deliverables, the project can continue to eat money and never deliver a product which can be considered as a baseline.

The problem with this type of contract arises because of the need to keep a tight control on the schedule. A combat model can be hurt by requiring a large number of closely scheduled deliveries. This happened on the JTLS 1.5 delivery, for which model deliveries were required every 3 weeks. There were so many management required deliveries that developers spent more time getting the deliveries out then they spent in upgrading the model.

In several cases, one developer could not start a major upgrade because the model needed to be fully functional for an upcoming delivery. The major upgrade would be placed on hold until the delivery was complete. After the delivery was made, the project would require long hours to complete the major upgrade within the shortened time period. The short delivery schedule resulted in every single aspect of the project being on the critical path. There was no slack, no room for miscalculation, no room for the unexpected, and no room for development teams to take a breath or rearrange their schedule given an unexpected problem.

When a labor hour contract is used, the contract monitor must find an acceptable median between too few and too many deliveries.

2.5.3 Cost Plus Fixed Fee

The current JTLS contract is a Cost Plus Fixed Fee contract. Although an intermediate delivery schedule has been established for the JTLS upgrades, it appears it is not as important as it was for the labor hour contract. The same problems can arise with this type of contract as with the labor hour contract and the same warnings apply.

3.0 DATA STRUCTURES AND ALGORITHMS

3.1 SCOPE OF DISCUSSION

When discussing the lessons learned about data structures and algorithms there are only a few generalizations that can be drawn across several models, but numerous conclusions on the advantages and disadvantages of various alternatives tied to a specific modeled system. Since the primary purpose of this report is to present ideas and concepts that JPL designers may find useful while planning and designing JESS II, the majority of the presentation in this section relates directly to the JTLS model.

JTLS is more closely related to the purpose and scope of JESS II than any of the other models and hopefully there is a better chance of finding applicability behind the lessons learned. We have assumed that there is little interest on the part of the reader concerning our conclusions about the Kth shortest path algorithm chosen for the LPDT project, or tree search algorithm implemented for testing alternative mission plans as part of the ALARM project. Furthermore, JTLS is our longest term project, and the one for which we have had the most opportunity to observe the results of the decisions that we and others have made. We have watched the evolution of the model for over five years, sometimes with satisfaction, sometimes with amusement, and sometimes with disbelief. The following discussions include some of each.

3.2 DATA STRUCTURES

The importance of creating a proper data structure for a wargame can not be over emphasized. It is decided at the very beginning of the model design process, and must be lived with and worked with throughout the remaining life of the project. We know of no rules or regulations on how to select the best and proper data structure since there is no single correct answer. Data structures are not measured in terms of right or wrong, but are measured in degrees of flexibility and ease of use. The following data structure elements were areas over which we struggled the most during the design process or for which we have felt the most restrictive when upgrading and improving the model.

3.2.1 Hardwired Constraints

Early in the process of defining the system that was to become JTLS, a decision was made that there would be no data constants imbedded in the code, at least the code of the CEP. While this rule was not followed to the point of absurdity (the value of pi is a constant), it was followed quite faithfully in most respects. This decision is a minor annoyance to the modeler in the design and coding phases, because of the requirement to have, for example, all loop indices be variables. In the broader view, it frees the modeler from the effort of trying to determine what are reasonable maximum values for such items as ending values for loop indices, and permits the wise use of the capability of SIMSCRIPT to dynamically allocate memory.

These are relatively minor matters, however. We believe that decision to have been as good a decision as any that was made on the JTLS project. Almost without exception, model users express satisfaction with the capability of the model to represent any reasonable number of aircraft types, combat systems, categories of supply, and so on. Whether we are presenting training sessions or technical papers, questions such as "How does the presence of chemical contamination affect the rate of movement?" arise. The response that it may or may not affect the rate, and may either decrease or increase the rate, all depending on the data always satisfies the questioner. Equally important, it helps to bring home to the users of the model, the importance of the validity of the entries in the database.

3.2.2 Terrain Representation

Any combat model must have some representation of terrain, even if it is only implied. During the JTLS design phase, at least two forms of terrain representation were considered. In MTM, and some other large scale models, terrain was represented as a series of hexagons, with homogeneous terrain within each hex. At NPS, a new method of representing terrain had been devised in the late 1970's, and used with considerable success in high resolution combat models. This method was called "functional terrain" and was the only serious competitor to the well established hex method.

Functional terrain represents the surface of the area of interest by a group of mathematical functions in the three dimensional plane. Each instance of the function represents a hill, described by defining the northing and easting (X and Y) of the hill center, and the parameters of the function that described the height of the hill at any point in X,Y. The form of the function was a bi-variate gaussian curve, centered at the hill center, with (possibly) different scale parameters for the two variates, and rotated through some angle from north.

While the representation of the terrain was in absolutely no way stochastic, the resulting curve was immediately recognizable as a picture of a bi-variate normal probability density function. The altitude at any point was computed as the maximum height of any of the functions at that point. The computation was speeded by the maintenance of a linked list of all the hills that had any significant effect in an area. In the high resolution models, the area was usually a one kilometer grid.

There was no automated method of producing functional terrain parameters from any input data. The parameters were produced manually, using standard military maps of 1:50,000 and 1:25,000 scales. During the three years from 1979 to 1981, in support of combat modeling research at NPS, two US Army enlisted soldiers had been taught to produce the data parameters using map sheets, a series of elliptical templates, and a program that produced and plotted contour maps from the input data. After training, each of the soldiers could complete a section of terrain ten kilometers by ten kilometers in about five work days. These representations, judging from ten meter contour interval plots, were as close to representing the map terrain as one hundred meter interval digitized data.

Functional terrain is very good at representing the altitude of terrain, a vital consideration in high resolution models, where line of sight computations are a significant consumer of computation resources. It permitted time savings in computing line of sight between two entities. It makes no contribution to the problem of what the other characteristics of the terrain are at some spot. Efficient, fast algorithms for mapping X,Y coordinates into detailed "patch" data were, however, available from the high resolution models.

The competition between the two methods was brief but intense. In the end, the hexagonal method was selected. While one of the R&A modelers had been a user of the functional terrain model, we believe that the decision was correct.

The decision to represent the terrain within a hex as homogeneous has not caused any significant problems, especially because the size of hexes and thus, the level to which the terrain is resolved, depend on the input data. As part of the hex representation decision, it was decided that barriers would be represented as occurring only at the edges of the hexes. Rivers would follow hex edges, wadis (dry washes or ravines) would follow hex edges, and bridges and tunnels would affect only barriers at hex edges. Road interdiction points, on the other hand, would affect the interior of a hex, because conceptually, the presence or absence of a road affected the whole hex. The bridges, tunnels and interdiction points were to be represented as target entities for purposes of intelligence reporting and damage assessment.

These decisions have not caused any more problems than any other abstraction and simplification of reality, with two exceptions. The first was to include in the terrain database, the fact that a "bridge" or "tunnel" crossed a barrier at a hex edge. The second was that a single bridge or tunnel target would provide the full effect of modifying the terrain or barrier from e.g. RIVER to BRIDGE OVER RIVER.

Both of these simplifications were very reasonable in light of the area of the world in which the intended first use of the model was to take place. That area was Southwest Asia. The intended hex size was nominally sixteen kilometers, with edges about eleven kilometers long. Bridges, tunnels and road interdiction points are sparse in that part of the world, and the assumption caused little or no trouble in that theater.

In Europe, on the other hand, it is not unusual to have five or six militarily significant bridges in an eleven kilometer stretch of a major river. The fact that killing a single bridge target removes the benefit of all bridges across the barrier has been a point of serious (and appropriate) concern for users in that theater. The concern is only slightly mitigated by the fact that if you repair a single bridge or tunnel, the entire capability is restored, no matter how many are destroyed. The inclusion of the bridge data in the terrain database was little problem in Southwest Asia, where essentially all bridges are militarily significant, and there are few enough of them that representing them all explicitly as targets was not a database size problem. In Europe there are so many bridges that including them all in the database (as targets) leads to an enormous database.

Leaving the targets out of the database is tempting, because the representation is already in the terrain. Unfortunately, doing so results in a bridge over a barrier that is invulnerable. It is there for either side to use, but neither side can damage it. In fact, once the model is started, there is NO way to change such a barrier. In some cases, the database preparer has yielded to the temptation. In some of those cases, the problem of invulnerable bridges has caused player problems.

We now believe that only natural features should be included in the terrain database, with the possible exception of roads. Any man made feature, that can be damaged and have an impact on the game, such as bridges, tunnels, or interdictable roads, should be explicitly represented as a damageable entity in the database. To the extent that such entities modify the battlefield, the combat model should make the modifications to the data parameters in whatever form they are being held.

3.2.3 Hex Structure

We have very few qualms about the actual partitioning of the battlefield into hexes. In order for the model to function in any sort of timely manner, the battlefield must be partitioned into sections, and some actions must be limited to the local section and its neighbors. For example, in a large database there may be a few thousand unit entities and several thousand target entities, and corresponding number of truck convoys, and air missions. When a moving combat unit changes its position, its possible interaction with other entities in the vicinity changes. Checking all other entities in the battle would be extremely time consuming. For air missions, which move a hundred times as fast as ground units, such a global check is clearly infeasible. The hex representation provides a reasonable method of partitioning the battlefield into local "areas" for quick determination of which entities are in the vicinity.

3.2.4 Using the Hex Partition

When algorithms other than battlefield partitioning are based on the hex structure, the results can be less than desirable. For example, in JTLS, the effects of all artillery fire are strictly limited to the hex in which the rounds impact. Two entities can be equidistant from the impact point of an artillery mission and have drastically different effects applied.

A unit in the same hex may have combat systems, supplies, aircraft and/or trucks damaged by the artillery. An identical unit, in an adjacent hex will suffer no casualties at all, even if it were closer to the impact point. Nuclear and chemical effects of artillery fire, both casualties and contamination, are limited to the impact hex. These are examples of a quick simplification without sufficient assessment of the consequences. It would not have been significantly more difficult to extend the effects to an adjacent hex if it were appropriate to do so.

The original computation of the probability of detection of entities by flying air missions, or detection of flying air missions by radar, were based on the hex. Each time an air mission moved into a new hex, it was permitted to try to detect objects, and was subject to detection. Thus, an RF-4 and an SR-71, both carrying the same sensors, had exactly the same probability of detecting an object if they followed the same path. Because the RF-4 would spend significantly more time on the path, for most sensors, its probability of detecting some enemy object probably should be higher. Similarly, each aircraft would have an equal number of opportunities to be engaged by enemy air defense during the flight, even though the SR-71 might spend only one third as much time in the air defense system's envelope.

The prohibition of units of opposite color in the same hex was a conscious decision. The rationale was that this was a reasonable first approximation, that for the first delivery it was acceptable, and that the limitation would be removed as soon as resources permitted. The decision was made and approved in July 1983. The change, permitting units of opposite color in the same hex was finally made in March 1988, and is due for release in September 1988. Implementing the change took about forty five analyst/programmer workdays. Implementing it in the first version would have caused a delay at least that long. In retrospect, we believe that the trade-off was a good one.

Two simplifying assumptions concerning limiting interactions to a small group of hexes were made in the logistics section. The first was that if a unit had to discard supplies, it would first attempt to distribute them to my friendly units in the same hex, and the surrounding six hexes. No units outside that region would be considered. The second assumption was that if an air unit (squadron) needed fuel, ammunition or other supplies to begin execution of a mission, it could draw those supplies (without delay) from any "collocated" unit. A collocated unit was one that was within the COLLOCATED DISTANCE input parameter, but in no case outside the same and surrounding six hex area. Neither of these simplifications has ever occasioned even the mildest comment from users.

Because the representation of the battlefield will almost without exception require partitioning, there does not appear to us to be a complete solution to the problem. Clearly, any limitation of interactions or effects to a single partition or set of partitions must be critically examined to ascertain whether all the implicit assumptions are valid.

3.2.5 Hex Data Structures

One of the considerations in the selection of the hex representation of the terrain was that JPL had the capability to produce hex terrain files from digitized data. These data were intended for use in graphics applications. A complete set of transformations from hex coordinates to latitude/longitude were available. The representation of the data used a coordinate system in which the coordinates for each hex were either both even or both odd. That is, the first column of data was indexed as (1,1); (1,3); (1,5) and so on, while the second column was indexed (2,2); (2;4), (2;6). The intended first terrain database was to be on the order of two thousand by two thousand kilometers, for a total of about fifty three thousand hexes.

The most efficient way of storing the terrain data appeared to be in arrays, although two dimensional permanent entities were briefly considered. (They were discarded as being merely hidden arrays, with extra overhead.) For each hex, a terrain value, an altitude, and six barrier values were to be stored. Storing only three barrier values for each hex was suggested late enough in the process to make it painfully obvious what a good idea it probably was, but too late to implement.

There was a design goal that the system should run in a two megabyte memory partition, because that was all the real memory the CAA machine had. Terrain value and altitude were to be (eight byte) real valued, the barrier values were to be (four byte) integer indexes. A quick computation indicated a requirement for forty bytes per hex. There was an explicit prohibition against bit packing data. The forty bytes for fifty three thousand hexes would use all of the allocated two megabytes, plus. The thought of having the terrain matrixes half empty, and using four megabytes just for the terrain data was rejected out of hand, as completely unacceptable. As a result, the terrain data were stored in matrices that were fully used. The JPL coordinate (1,3) was converted to JTLS coordinates (1,2); (2,2) was converted to (2,1); and so on.

The result was a set of data structures that were very dense with data, and wasted very little memory. The cost was an extra step in the process of converting from hex coordinates to latitude/ longitude or vice versa.

One of the authors frequently states that he has never traded off model speed to save memory without regretting it in the long run. This may be the exception that proves the rule. The conversion between coordinate systems need only be done to provide output to the players. The size of the program image is always a factor. However, when the memory tradeoff reaches two megabytes, even on modern machines, with virtual memory and caching, the memory tradeoff has speed implications because of swapping issues.

3.2.6 Surrogates

Five major types of military units were to be represented in JTLS. They were Airbase Units; Ground Combat Units; Air Squadron Units; Support Units; and Naval Units. While the majority of the data required to represent a unit was common to all five types, there were some unique data for each type. For example, Support Units have cargo trucks, tanker trucks, and own three sets (linked lists) that other units do not have. Squadrons have counts of aircraft, capabilities regarding sortie generation, and sets for air missions, Airbase Units have runways and runway repair capability.

We decided that each of these unit types would have a surrogate entity that would be used to store the type peculiar data. The amount of memory used to store all the data for one surrogate of each type (as of 1986) was 328 bytes. Forty bytes were used to point back to the parent unit. Each unit used four bytes to point to the surrogate. There were duplicate sets. The saving was clearly less than 300 bytes per unit, and probably less than 255 bytes. The goal database was three hundred units. Very few databases have been executed with more than five hundred units. A thousand unit database is considered a VERY large database. For a VERY large database, the saving is less than three hundred thousand bytes.

The cost of having surrogates is one (or occasionally more) extra level of indirection in accessing the unit attributes that are type peculiar. For Units, that does not happen very often. For Air Squadrons, Airbases, Naval Units, and Support Units those attributes are accessed almost every time the unit does anything mission related. A 1985 assessment of JTLS CEP performance during a large exercise in Europe indicated that the CEP spent between thirty-five and forty percent of its available time in the routines that perform subscript checking. Extra levels of indirection contribute directly to that time cost.

We believe that using surrogates for the purpose of saving memory, at the cost of the extra level of indirection, was an error. The limitations imposed by the fact that only units of major type Y can perform action X has been a disguised blessing that came as a direct result of the surrogate structure. On balance, we believe the decision had negative impact on the system. We firmly believe that, within reason, the designers concern should be for speed as opposed to memory. The impact of increasing size of the executable image and its effect on model speed cannot be ignored, as noted in a preceding paragraph.

3.2.7 Arrays Versus Permanent Entities

SIMSCRIPT offers two different types of entities: temporary and permanent. Single temporary entities may be created and destroyed at any point during the simulation, while permanent entities may only be created or destroyed as an entire class. A permanent entity class is really represented in memory exactly as an array is represented, with one difference.

In SIMSCRIPT, all the entries in an array are of the same mode, such as real or text. A permanent entity, like a structure or record in other languages, permits parts of the structure to be real, other parts to be integer, others to be text, and still others to be array pointers. We failed to recognize the inherent advantage in using permanent entities to model such things as types of aircraft, types of combat systems, categories of supply and types of air weapons. A single combat system, such as T-62 tanks, has entries in a text array, a numeric array, and several multi-dimensional probability of kill arrays. The numeric array is a real array, but several of the entries are conceptually integers, such as the supply category (an index) from which the system is replaced. Every time that entry is accessed, a real to integer conversion takes place. This takes CPU time.

Using permanent entities would have permitted the use of a single structure to completely describe the entity. Additionally, the use of multiple dimensioned arrays makes the source code more obscure. CS.FRACTION.RECOVERED(COMBAT.SYSTEM) is at least marginally more intelligible than COMBAT SYSTEM CHARACTERISTICS (BLUE, 2, COMBAT.SYSTEM).

As noted in a preceding paragraph, much of the processing time in JTLS is spent range checking subscripts. The single level subscript is checked more quickly than the triple level subscript, even when very clever code is used. We believe that we should have used permanent entities at least for the four examples given, and have converted some of the original arrays to entities.

3.2.8 Temporary Entities

Unlike our decision to use arrays instead of permanent entities, we believe that the decision to use temporary entities (as described in a preceding paragraph) for military units and targets to have been correct. As it turned out, for targets, if any other decision had been made, it would have had to be altered to permit the Game Controller to create new targets as the game progressed.

In the original concept, neither units nor targets were to be destroyed, but the decision to make them temporary entities was never in question. As the model developed, an original requirement to have units arrive incrementally was redefined to require, in essence, the potential addition of increments to all (or potentially all) of a unit's capabilities. The fact that the unit entity was a temporary entity made it quite simple to use it as the incremental arriving entity, simply to hold the data until arrival, and then to destroy the entity, getting it out of the way, and releasing the memory back to the dynamic allocator. Other entities that are temporary in JTLS are SUPPLY RUNs (convoys) and AIR MISSIONS. Because it is impossible to predict how many of those entities may be required, there is no choice but to make them temporary entities.

The lesson to be learned from these entities is not in the choice of entity type, but in the management and cleanup of entities. In SIMSCRIPT, it is common practice to pass entity pointers (base addresses) as the arguments to events and routines. The zero-th word of the entity has a bit flagged to indicate whether the entity has been "destroyed", i.e., had its memory returned to the "heap". As part of the subscript range checking discussed in a preceding paragraph, SIMSCRIPT checks that bit. Attempting to access a "destroyed entity" is an immediately fatal run time error, or "Catastrophic Software Anomaly". If a temporary entity is destroyed, and its pointer is referenced on some future event, or in some array list of pointers, when that pointer is de-referenced, the model will crash. This makes it imperative that the model never attempt to access a destroyed entity.

For convoys, the problem is handled by never destroying a convoy that has a future event scheduled. The convoy is left as an entity with zero trucks left, and eventually destroyed in the future event. Because the modeling of convoys is quite simple, they seldom have their pointer referenced on more than one future event, and never in arrays.

Air missions are modeled at a much greater level of detail, and frequently have their entity pointers referenced in half a dozen future events, and in the sets of several different other missions. No matter when an air mission entity is destroyed, there is substantial cleanup to be performed. All future events that reference the entity must be found, and canceled, all surrogate type entities must be found, removed from whatever sets they are in and destroyed, and the mission itself must be removed from all sets before being destroyed. (Destroying an entity that is in a set is also a fatal error.) This requires a lot of code, and a great deal of care in the processing of the mission destruction.

Another approach that we have seen used involves target entities, which are also temporary entities. As discussed earlier, in the initial design, targets were never to be destroyed, although new ones could be created. A later change to the model added minefields as a target type, created the target entity when the field was laid, and destroyed the entity when the minefield was cleared.

To represent the fact that helicopters do not crash when they run out of fuel, another type of target, the DOWNED HELO target was introduced. A target entity was created, the air mission entity was destroyed, and a refuel of the downed helos was scheduled. When the refuel occurred, the target entity was destroyed, a new air mission entity was created, and the mission returned home.

In both cases, the decision was made (either implicitly or explicitly) to protect all references to targets from the possibility of referencing a destroyed entity, rather than seeking out and expunging all references to the entity before destroying it. References to targets can occur in Player Orders, in detected object lists, and on future event notices. As a result of the change, any time any of these items were processed, before any attempt to access any attribute of the target entity was made, every target in the game was accessed, and its pointer compared to the pointer about to be dereferenced. If the pointer was found as an active target the processing continued, otherwise, it was terminated. Large databases contain thousands of targets. This procedure is followed even if no targets have ever been destroyed!

R&A analysts are convinced that this is not optimal design. We feel rather strongly that there is no need to destroy targets, once created, provided the creation is treated judiciously. If it does become necessary to destroy a temporary entity, we know it is smarter to pay the price to find and eliminate all the occurrences of its pointer before destroying the entity. We are well aware of the method of tricking SIMSCRIPT into letting you check whether the entity is destroyed, without inducing a crash, but feel rather strongly that that step need not be taken, at least not in the CEP.

3.2.9 EVENTS VERSUS PROCESSES

Simulations usually reflect one of two world views, Process/Resource or Event. Some simulations, and some simulation languages, reflect a Process/Resource view of the world. This view explicitly represents the processes that occur as the modeling paradigm. For example, in a bank simulation, a customer needing service might be a demand, the tellers might be resources, and the provision of service, a process.

In the Event world view, events are moments in time where the state of an entity explicitly changes. In that world view, the customer and teller would be entities, and the things modeled would be the changes in status of the entities, such as the start of service, the completion of service, and leaving the bank. We are convinced that these two paradigms differ primarily in the way that they look at the world, and that the difference is about ninety degrees.

Our two modeler/analysts have both written both types of simulation, and are convinced that any system that can be modeled in the Event world view can be modeled in the Process/Resource manner, and vice versa. Unlike some languages, SIMSCRIPT permits either world view to be explicitly used, although this is a relatively recent change, occurring in the very early 1980s we believe. Regardless, we were more familiar with and more comfortable with the Event world view, and so we selected it. We have no regrets on that score at all.

3.3 ALGORITHMIC DECISIONS

Before discussing various problems and conclusions we have reached over the last few years about algorithm design, we want to reiterate our feeling of getting a new model up and running as quickly as possible. We do not want the reader to leave this discussion with the impression that a modeling project should not be released without some of the detailed modeling ideas that were eventually put in the JTLS model. You can not develop a new model that has everything. It must grow and develop over time. With this is in mind, the following discussion presents various major algorithm development concepts and specific algorithm ideas that have either operated satisfactorily or have been troublesome throughout the five year growth of JTLS.

3.3.1 Algorithms That Use Multipliers

In many parts of the model, there was a requirement to model a difference in some parameter based on environmental conditions, current activity of an entity, light conditions or other factors. This requirement applies to some sensor probabilities, and air to ground weapons effects, but most visibly to ground movement. Each unit entity has an AVERAGE SPEED attribute, that represents the speed that the unit can achieve, cross country, through open terrain, unimpeded by barriers, unmolested by enemy forces, performing an administrative move.

When the unit actually moves, it usually does not do so under those circumstances. It may move on a road, the terrain may be mountainous, there may be river to cross, minefields to negotiate, the unit may be harassed by artillery fire or air strikes, or it may be in a tactical formation that changes its speed capability. Each of these factors modifies the speed that the unit can attain. In an attempt to capture those effects, a series of multipliers was required to specify the size of the effect. For example, moving sixteen kilometers in the mountains might take twice as long as in open terrain; crossing a river barrier might take three times as long as if the river were not there; or a unit moving in the attack might take one and a half times as long as one performing an administrative move. In general, the users have not had much trouble coming up with reasonable numbers for the multipliers, although, at last check, the original terrain and barrier multipliers from the SFD were still in some terrain databases.

The problem arises, for the case of unit movement, and other cases, when there are multiple mutiplicative factors. Does a unit moving across a river in mountainous terrain, in an attack posture really take (3 * 2 * 1.5) nine times as long as the base case unit? Suppose the unit is also in a minefield, in a chemically contaminated area, and being fired upon by artillery? The problem is that there is no simple way to get at the interactions. The unit move time, admittedly the worst case, potentially has at least eight potential multipliers. If each were two, the net result would be a 256 fold penalty. That seems rather extreme.

We are convinced that the multiplicative approach is better than an additive approach. To say that it takes all units an extra hour to traverse a mountain hex, regardless of their base capability, is not appealing.

We are aware of the possible solution of selecting the multiplier that causes the largest penalty, and only applying that one. We do not believe that solves the problem, either. Once the player has applied the highest penalty solution to a location or an enemy unit, there is no sense in applying any other interdictive effects. The law of diminishing returns applies with a vengeance. In the real world, we think that it does take longer to move through a minefield if the area is also contaminated or under fire.

If the number of interactions is small enough, it is possible to create an array of multipliers. For example, when determining the degradation of an aircraft to deliver weapons during poor weather night conditions, instead of using the poor weather multiplier and the night multiplier, an array would be entered to obtain the single poor weather, night condition multiplier. Although the interaction problems are taken care of in this circumstance, this approach creates several other problems. First and foremost, is that the amount of data grows drastically when considering processes that are affected by several multipliers. More importantly there is a problem with obtaining data. Multiplier data are "soft" data and represent an expert's opinion of what would happen. It is fairly easy to obtain a consensus among a group of experts for one multiplier, but this gets harder when the group is required to consider a large variety of interactions when determining multiplier data.

We have no solution to this dilemma. Full application of the multipliers may be overcompensating, applying only one seems undercompensating, and additive penalties just don't seem appropriate for starting capabilities that are significantly different in size.

3.3.2 Expected Value Versus Stochastic Algorithms

A decision that must be made quite early in the development of a combat model is the method that will be used to represent the fact that combat is not a deterministic process. Usually, the choice is limited to either an explicit representation of the stochasticity of the processes via a Monte Carlo methodology, or a simple representation of outcomes as the expected value. A simple example may be found in the representation of the impact of a volley of artillery fire.

The realized impact points of individual projectiles are remarkably close to a truly bivariate normal distribution, independent in down range and cross range errors. The mean point of impact is the aim point (for good crews). The artillery fire only affects the entities in the vicinity of the impact point. The choice as to how to determine the impact point must be made from a wide spectrum. This spectrum ranges from modeling the aim point as the center of the effects, with the individual impact points distributed evenly around it; through drawing pseudo-random numbers to determine the actual center of impact; to drawing random numbers for the range and deflection errors of each round. After the decision is made as to how to model the impact, the question of how to model the casualty effects must be made. Again, the decision between drawing random numbers and assessing an expected value result must be made. One fully stochastic representation might involve drawing random numbers and assessing results for:

- The number of cannons for which the igniter functions;
- The number of cannons for which the powder ignites;
- The variance of the weather from that expected;
- The mean point of impact for the volley;
- The actual impact points of each of the rounds that impact;
- The functioning of the fuse and explosive charge in each projectile;
- The casualty causing effect of each of the projectiles that detonate on each entity in the area.

The distribution and parameters of each of these functions are at least moderately well known. Alternatively, one might convolve all the distributions into a single distribution that gave a probability of damage as a function of the range or range and bearing of an entity from the expected mean impact point.

An expected value approach might specify that all of those entities within an ellipse of some size, centered at the mean impact point, will suffer x percent damage, and assessing that much damage to each entity.

It is very important to note that the first approach is not MORE stochastic than the second approach. Stochasticity is like some other states, in that there is no such thing as being "a little bit" stochastic. A process is either stochastic, or it is not.

The first approach has all the parameters explicitly represented, and may be appropriate for very high resolution models used to investigate the importance of small parts of the whole process. The second and third approaches are more appropriate for measuring the effect of the system.

In those cases where the effects of each occurrence are (usually) applied to a reasonably large number of possible victims, the expected value approach must be equivalent to the aggregated stochastic approach. For other cases, this equivalence is not possible.

For example, suppose in a model, that an air mission is directed to attack and destroy a bridge, subject to the limitation that if it loses more than one half of its total aircraft, it is to return to base, regardless of whether it has attacked the bridge or not. Further suppose that the bridge is so well protected by air to air and surface to air resources, that the expected losses to an inbound air mission are sixty percent. If an expected value representation is being used, the bridge will never be destroyed. No matter how many attacks are tried, the missions will always be forced to turn back. If a stochastic representation were to be used, and enough attempts were made, eventually a mission would get through, and if enough missions got through, eventually the bridge would be destroyed.

In a theater level model, we believe, some processes are adequately represented by expected value models. Lanchestrian attrition, artillery fire against military units, and consumption of supplies are examples. In these examples, the process affects a large number of entities in the same way or represents a large number of small processes occurring. If the process is applied to a single entity, or to a group of entities, that are distinguishable, one from the other, we believe it should be represented by a stochastic model.

It is important to recall that the model in question here, JTLS, is an interactive model. It is, ipso facto, a stochastic model. Human players can not replicate a battle without changes, nor should they try, in general.

3.3.3 Land Combat Attrition Algorithms

In combat models, attrition of forces in land combat is usually represented in one of three ways. They are Firepower scores, Monte Carlo, and differential equations, usually referred to as Lanchestrian attrition. The Lanchestrian formulation was selected for JTLS.

The basic paradigm is a linked set of differential equations that involve the number of firers, a kill rate (Attrition Coefficient), and, for area fire, the number of victims. The kill rate can be input, computed from input based on the circumstances, or input and modified based on the circumstances. The original design of JTLS permitted a large number of sets of attrition coefficients, and the selection of one set to use based on the color of the attriting unit, the posture of the attritor, the posture of the victim unit, the light condition (day or night) and the weather condition (Good, Fair or Poor). Two colors, ten postures, two light conditions, and three weather conditions resulted in a five dimensional, twelve hundred entry, array of indexes (COMBAT INDEX) to the actual attrition coefficient array.

Two additional effects were omitted, both consciously. They were the effect of terrain on the attrition rate, and the effect of "distribution of fire". They were omitted for different reasons. The terrain was omitted as an unnecessary data complication. With an original database of fifteen terrain types, either multipliers would have been required, or the size of the COMBAT INDEX array would have increased significantly. The availability of attrition coefficients was a matter of minor concern at this point, but a single multiplier for all systems in the same terrain seemed overly simple, while another array of multipliers indexed by killing system, victim system, and terrain type, seemed to be overkill.

The other effect, distribution of fire, was omitted solely in the interest of model speed. Kill rate data have a strong implied assumption about the composition of the force that the killing side is facing, and the way in which the commander or individual killer divides his or her attention among the potentially different type of fighting systems. For example, a kill rate that specifies that each M-60 tank will kill 0.098 BMP vehicles and 0.12 T64 tanks per hour, might assume that the victim force was thirty percent T64 tanks and seventy percent BMPs. The problem occurs when the killing unit faces a victim unit that has for example, one hundred percent tanks. By assumption, some fraction of the M-60 tank fire was dedicated to BMPs to attain the .098 kill rate. In the basic design, there was no way to reallocate that fraction to kill the extra T64s.

As long as the forces that were faced were relatively homogeneous, the representation was more than adequate. The computations to reallocate the available killing power involve a fair amount of arithmetic, are moderately complicated, and must be made each time attrition is assessed.

Both of the omission decisions were made by a single analyst/programmer, with minimum discussion with R&A management, JPL management, or sponsor representatives. We believe that the decision reflects an example of the potential results of "benign neglect" on an area of the model. There was no real specified point of contact for the issues, no one ever really questioned the decisions, and the selected simplification worked. Both effects have now been included in the Ground module of JTLS.

The terrain modifier is a multiplicative factor, based on the color of the killer system, the type system of the victim, and the terrain index. It is applied if the killer is not in the attack posture. The allocation of fire correction is also multiplicative, and is based on the ratio of the expected number of each type combat system facing the unit, compared to the actual number of each type of system, summed over all the enemy units in the same hex as the victim unit.

3.3.4 Level and Consistency of Detail in the Major Models

Some critics of JTLS allege that there is an unacceptable disparity in the level of detail to which the model resolves the battle between the major modules. Most frequently, the concern is between air and ground.

The air module resolves units down to the squadron level, where a squadron is a unit that has, by definition, a single type of aircraft. The operating entity that executes the air war is the air mission, a group of aircraft, all from the same squadron, attempting to perform some mission, such as an Air to Ground Attack against a bridge. The mission moves from its squadron location, along a player specified (optional) route to its destination, performs the mission it was sent to perform, and returns to its home base via an (optional) egress route. It moves from hex to hex, using the same hex representation as the ground based units, and is subject to detection, attack, and attrition as it moves along.

The air mission always occupies the center of the hex that it is in, if it is flying, and is invisible and invulnerable while it is on the ground, refueling or on strip alert. Within the mission the current count of surviving aircraft, remaining fuel, and weapons are tracked. There is no tracking of individual aircraft, nor is their geographic distribution about the mission center ever represented explicitly.

The ground module resolves units down to whatever level is built into the database. Typically divisions, brigades and a few special battalions are chosen. Units move along a user specified path to their destination, but do not usually return. While they are moving, they are subject to attrition, detection, and attack. Within the unit, the current count of combat systems operational and supplies available are tracked, but individual combat systems cannot be distinguished, nor are their individual locations ever represented explicitly.

When a player wants to initiate a ground attack, or to fire explicit artillery, or to lay a minefield, he or she creates an order to a unit to perform the task, and sends it to the unit. The unit as a whole performs the action. When a player wants to provide airborne air defense, or attack a deep target or suppress air defense, the player creates an order that tasks a squadron unit to create an air mission, and send the air mission to perform the task. The air mission is the entity that performs the action.

It seems apparent that the mission and the unit are the same class of entity, an action implementor. The question of disparate level of resolution then comes down to whether the air mission and the unit are the appropriate levels of resolution. Because the unit can be resolved down to whatever level the players can manage, and for which the analysts are willing to accept Lanchestrian attrition as a reasonable representation, it is seldom suggested that the unit is either too high or too low a level of resolution.

One of the principal objections to the pre-SFD version of MTM was that a squadron could only perform a single mission (not a single type mission) at one time. If a squadron was flying Close Air Support for unit A, it could not provide any support for Unit B. There are two steps to a better representation of detail: a flight of aircraft (we call it a mission) or single aircraft. The idea of tracking tail numbers was examined but was rejected by the analysts and the sponsors as too much detail. That left the mission as the reasonable compromise.

It is sometimes suggested that the mission as the action entity is not the problem, but that it should be possible to get the effects, whether damage, protection, intelligence or loss of own aircraft, from the general availability and allocation of resources, without actually flying the missions. It is true that air mission movements and interaction constitute a large fraction of the events that take place in the model. Air missions move and interact much more quickly than ground units in the model, as in the real world. We are aware of other models that do model the effects of air resource allocation without actually modeling the flying of the mission.

None-the-less, we believe that the explicit audit trail of cause and effect that is permitted by the explicit representation of the flight of the mission, and its interaction with other missions, Surface to Air weapons, and its target is needed. As long as there is an intelligent opponent who is attempting to prevent the air resources from accomplishing their mission, and that opponent can make decisions and commit resources that effect the mission while the mission is flying, we believe that representing the mission profile explicitly in time is necessary.

We believe that the levels of resolution are as close to being commensurate as possible, given the intended use of the tool. There do seem to be two schools of thought in Air Force circles concerning the Air Mission in JTLS. They profess respectively, that there is not enough detail, and that tracking tail numbers is required; and that the model is too detailed, that all that is needed is the allocation of resources for each of the mission areas.

One lesson to be learned from this is that the level of detail will never satisfy everyone. A serious effort must be made by both the analyst/modelers, and management to resist the urgings of sponsors to add inappropriate levels of detail. Once one module has a higher level than some other module, the problem is started. The phrase "This is supposed to be a theater level model." stood us in good stead more than once.

3.3.5 Latitude/Longitude Versus the UTM Controversy

The method used to report locations to the player caused some problems. The original specification required that, if a hex based terrain was used, that structure be invisible to the players. The JPL based terrain algorithms converted hex coordinates to latitude/longitude and vice versa. The original release of JTLS accepted location input and provided location output only in latitude/longitude form. The Air Force sponsors were content with that method, because they were used to working in that format. Very early in the functional validation process, the Army sponsors' representatives expressed a strong desire to have their input and output in Military Grid format.

Military Grid is a modification of the Universal Transverse Mercator (UTM) map projection, that breaks each UTM segment into lettered 100,000 meter grid squares, and reports coordinates as a Northing and Easting from the southwest corner of the grid square. Within the military, the Military Grid system is usually referred to as UTM. The requested change was from latitude/longitude to UTM, but what was wanted was a conversion to Military Grid. The conversion is relatively straightforward, although it involves the computation of two or three transcendental functions, which are a little slow. There are three required inputs to the computation: the latitude; the longitude; and the radius of the earth spheroid used in making the maps for which the Military Grid was to be used. The US Army Technical Bulletin covering the topic listed four different earth spheroid models as used in the area covered by the Southwest Asia terrain database.

The areas within which the different models were used were extremely irregular. Examination of the military maps being used for the functional validation revealed that a spheroid model was used in an area allegedly being covered by a different spheroid model. Offline tests of the algorithm established that using the wrong spheroid model caused errors as large as three kilometers. In our judgement, confirmed by the sponsors, that size error was unacceptable. The users were unwilling to either input the spheroid for the area with each order, or accept model speed penalties of the size we estimated it would take to do an irregular polygon search to determine the area within which a point lay.

At the time of the controversy (1984-1985), JTLS did not have a graphics capability. Any location input from a player had to be typed in using the latitude/longitude format. Since then, a relatively robust graphics capability, directly descended from the JESS graphics has been added. It permits the player to enter locations into directives using the graphics puck, thus substantially easing the typing load.

At the last JTLS Configuration Management Board, the Engineering Change Proposal (ECP) to add UTM output was on the agenda again. The principal proponent of the change during the early phases, was present as an advisory attendee to the Board. He strongly advised that the ECP be cancelled, as no longer necessary, due to the addition of graphics, and not worth the cost to achieve the desired level of accuracy. The Board concurred and cancelled the ECP.

The lesson that we draw from this is two-fold. First, when two groups of users come from different technical cultures, it is important that the cultural preferences for output of both the groups be considered. Second, if the analysts and developers hold firm to their demands for technical correctness, they can usually prevail.

3.3.6 Explicit Artillery Fire

The original specifications required that the player be able to explicitly direct artillery units to fire their artillery. This requirement was implemented. Because the model was a theater level model, and the lethality of artillery fire is a strong function of its density, as well as other factors, the internal representation of the effects was in terms of the fire density per unit area. Unfortunately, the units chosen were in terms of tons of ammunition per hectare covered. A ton of ammunition is easy to measure, and a hectare is a standard metric unit of terrain area, and is used by some non-US forces as a standard against which to measure ammunition expenditure. The response to a query to the Army Materiel Systems Analysis Agency indicated that lethality data against combat systems in terms of tons of munitions delivered per hectare was easily available. The player was required to specify a radius of fire, and the amount of ammunition to be expended. The implementation required the input of the ammunition expenditure in terms of tons.

As it turned out, most Ground players were not used to thinking in terms of how much ammunition to expend against a target; those that were, thought in terms of volleys or rounds, and neither had a very clear idea of how many of either were in a ton. Most of them had no idea that a hectare was ten thousand square meters (100 x 100). The current version accepts input in terms of rounds, which is converted to tons inside the CEP. The input expected kill values are still in terms of PK per ton per hectare.

We believe this particular algorithm to be one of the causes of the allegation that JTLS "requires arcane data transformations". The lesson to be drawn, again, is to make sure the input structures reflect the cultural prejudices of the user.

3.3.7 Aircraft Maintenance

There was a strong desire on the part of both the modelers and the sponsors to be able to model the capability of air units (squadrons) to "surge", that is, to fly more sorties in a high demand environment than they can sustain over a longer period. In effect, this amounts to borrowing tomorrow's resources in order to survive until tomorrow. The cost is a reduced capability to generate sorties for some period in the future. During discussions with the Air Force sponsors, the consensus was that the penalty for the surge did not last longer than about three days. The algorithm designed required as input data the length of the surge adjustment period, a mean number of sorties per period that each squadron could generate, and a weighting factor for sorties flown three periods ago, two periods ago, and the previous period.

The model kept (and keeps) track of the number of sorties that the squadron generates each period, weights them according to the weighting factors, and uses the number of aircraft in the returning mission and the unit capability to generate sorties to compute the expected maintenance time. The accumulated equivalent sorties are then used to lengthen the maintenance time, by multiplying it by 1.0 plus the ratio of accumulated sorties to sorties per day capability. Longer maintenance times lead to fewer aircraft available to fly more sorties. This algorithm leads to exactly the type of surge penalty that was desired, given the correct three weighting parameters and sortie generation rates. Unfortunately, there are no real world data to use to generate the weighting factors. In trying to explain the meaning of the parameters to operational users, the confusion equals that caused by the tons per hectare algorithm. This is another source of the "arcane transformation" statement.

The lesson here seems to be that the fact that the analyst/modeler can derive a function that generates the desired effect is not as important as whether there are readily available, understandable data to support the function. No matter how badly a function is needed in the model, if no data are available to support the algorithm, that part of the model will not be acceptable.

3.3.8 Quality of Solutions

On occasion, it has been suggested (by non-original team members) that R&A is known for providing twenty dollar solutions to twenty cent problems. We have been known to reply that that is better than providing five cent solutions to twenty dollar problems. The real point of discussion here is the question of how good the first cut at modeling some phenomenon should be. How much detail should be included? What level of fidelity should the modeler try to maintain? How much abstraction of the real world is acceptable?

In a 1967 paper, W.T. Morriss suggests that modeling is an iterative process. That the optimal procedure is to start quite simply, and to gradually enrich the parts of the model until you can no longer solve it (or afford to solve it).

R&A earnestly supports at least the first step of this paradigm. We believe the modeling team, in its effort to get a product up and running, must model each phenomenon at the minimum acceptable level as a first cut. We refer to that minimum level as the twenty cent solution. Determining the minimum acceptable level is not a precise science, as some of the preceding discussion has indicated.

When it becomes apparent that the twenty cent solution first implemented is not good enough, a better solution must be found. Usually, the source of the dissatisfaction is the end user, and if the problem is important enough to fix, it is important enough to fix right. The fifty cent patch simply won't do. We go directly to the five dollar solution. Some examples may be instructive.

d. Twenty cent solutions that proved acceptable.

For long distance haul of supplies, the truck convoys that transport the supplies are explicitly represented. An early decision was that it was not necessary to model the location of convoys within a hex. Conceptually, they were somewhere in the hex, precisely where was not important. This is a moderately large simplification, given a sixteen kilometer hex. Still, it has caused no problems, and never been even a small bone of contention.

Similarly, when a convoy arrives at its destination and has lost some trucks to attrition of one sort or another along the way, some assessment of the supplies lost must be made. While experience might indicate that the supplies lost will be precisely those that are most needed, the decision was made that the loss of supplies would be exactly proportional to the loss of the type of truck that could carry the supplies. If ten percent of the tanker trucks were lost, ten percent of all wet supplies were lost as a result. This simplification has never occasioned any negative comments.

e. Upgraded twenty cent solutions.

In the original delivery, all units performed their computations of routine daily consumption of supplies, decided whether to requisition more supplies, and forwarded the requisition at the same time, typically at midnight each day. As a result, supporting units could not take action on the unfulfilled requirements of their supported units until the next day. This resulted in unacceptable delays in providing support. The solution adopted was to give each unit its own periodicity of consuming, and requisitioning.

It would have been somewhat easier and perhaps equally satisfactory to map from a level of command to a period, e.g. all brigade level units might requisition at eight hour intervals, division level units at twelve hour intervals and so on. On the other hand, it might not have been satisfactory. Once the model gets into the configuration managed mode, it is worth a little extra effort to avoid having to re-address a problem, if only for the sake of efficiency and avoiding administrative overhead.

A second example involves the repair of damaged runways. Originally, when a runway was damaged, if there was an owning airbase, and it had a repair capability, a repair of the damage was scheduled. If the damage was a twenty percent reduction in capability, when the repair was completed, the capability was increased by twenty percent. While the runway was being repaired, that twenty percent was not subject to any further damage. This proved to be unsatisfactory. Once a runway was reduced below twenty or thirty percent capability, there was no advantage in attacking it. A forty percent improvement might be about to come on line, but that section of the runway was inaccessible to any type attack. The current model has a small internal simulation of the process of selecting a cut to repair and repairing it.

f. A twenty dollar solution that was overkill.

Two kinds of trucks are modeled in JTLS, cargo trucks and tanker trucks. Cargo trucks can only transport supplies that are classified as DRY; tanker trucks can only transport those classified as WET. Clearly, in the real world, it is quite simple to transport different types of DRY supplies on the same truck. Spare parts, rations, and medical supplies can easily share a vehicle, and so we modeled DRY transport as infinitely shareable among cargo trucks.

WET cargo is a different matter. The thought of using a tanker truck to ship a half load of jet fuel and a half load of drinking water at the same time was repugnant, and aesthetically unacceptable. A few hundred lines of code are devoted to ensuring that WET supplies are segregated by truck, that no mixing occurs, and to topping up the trucks, if there is excess capacity. No user has ever expressed the slightest enthusiasm for this modeling nicety. Of course, as noted in a preceding paragraph, if there are ten tanker trucks in a convoy, carrying three different WET categories of supply, and one truck is killed, each of the three categories is assessed a ten percent capability, even though, by explicit assumption, the truck could only have been carrying one category of supply. This is indeed a twenty dollar solution to a twenty cent problem.

g. The fifty cent patch.

In the original model, when an air mission ran out of fuel, it crashed and was destroyed. The model went to great lengths to keep a mission from running out of fuel, refueling at airbases and orbiting tankers as required, and going for fuel whenever it decided that it could not achieve its mission because it couldn't get to a destination with the fuel it had. Still sometimes, the tanker was gone when the mission got there, or the mission got involved in air to air combat and used excessive fuel, and the mission ran out of fuel.

For helicopters, it was decided that the destruction of the out of fuel mission was unacceptable. As a result, the DOWNED HELO target category described in the entity pointer discussion in a preceding paragraph was added to the model. There was no capability to access missions that were on the ground for any reason, and it was desired that downed helicopter missions be susceptible to damage. To solve this problem, the DOWNED HELO category target was created, a few of the attributes of the mission were transferred to the target, and the mission was destroyed.

The DOWNED HELO target was, like all targets, susceptible to damage from air missions, artillery fire, and encroaching enemy units. If the target survived long enough to get refueled, a new air mission was created, the remaining known attributes transferred to the mission, the target entity destroyed, and the mission flown back to its home unit. For the rather rare case of the helicopter mission that ran out of fuel in an area with no friendly units around, this added to the model capability.

The missed opportunity was the failure to recognize that the fix could easily have been extended to improve several other simplifications, at very little cost. Given the method actually implemented, a net saving would probably have resulted. Missions on the ground awaiting completion of the loading process, refueling, waiting for the arrival of a unit for airlift or drop, or standing strip alert at a runway or airbase, are not accessible for damage by any mechanism. If, instead of converting the mission into a target, the missions had been made accessible when they were on the ground, the fidelity of the model could have been improved for, we believe, essentially no additional cost, and in a less complicated manner. As a bonus, none of the mission attributes would have been lost in the double transformation.

The moral here is not to get away with as much as you can. Rather it is that in combat modeling, as elsewhere, the simplest acceptable solution is the best solution, and unnecessary complication is a waste of resources. The best way to find out whether the simple model is acceptable is to try it. If it is not acceptable, don't patch it - FIX IT COMPLETELY!

3.3.9 Color Indexing of Data

JTLS contains many data entries that are indexed by color, and some that are not. Examples of those that are not are the number of tons a cargo truck can carry, the maximum support distance between units, the weighting of the preceding days air sorties, the speed at which HUMINT teams move while deploying, and the category of supply that represents artillery ammunition. None of these has caused really large problems, but in the development of a database they cause troublesome compromises to be made. Soviet trucks may be on the average larger than US trucks, or vice versa. One side's maintenance capability may be so superior that there are significant differences in the effects of surging sorties, and so on. The point is that every data entry, when at all practical, and even if you are sure it will never be needed that way, should be indexed by color.

3.3.10 Controllable Fidelity

One of the capabilities of JTLS is to establish the database so that logistics is unconstrained, or consumption is not modeled. Frequently, when the model is being used, one or the other of those options is desired for at least one side. The analyst may want to establish a worst case if the enemy is completely logistically unconstrained, or a best case if the friendly side has no constraints. In order to do this the database must be modified extensively. Units must be given "unlimited" supplies, in one case and the supply consumption data for each unit must be modified in the other. A smarter design would have included a global (color indexed) switch for each of those functions. The consumption computations are only made in a few routines, and the check would have been very simple and quick. The unlimited supplies switch would have been equally simple. As it is, a separate database must be established for each different case.

Part of the design considerations should be a consideration of which features should be switch controllable.

3.3.11 In Game Access to the Data

In JTLS, the Controller player has the capability to change almost any data parameter. We believe that this requirement was essential, although implementing it completely was a great deal of trouble. We have acted as Technical Coordinator, Controller, or technical advisors to a large number of functional validations, exercises, wargames and analyses using JTLS. We do not know of a single one for which the Controller capability was not essential. The capability should be a requirement for any interactive model, in our opinion.

As it has turned out in the long run, even the capability to change any single parameter was not really enough. In order to achieve some effects, the Controller had to create a modification order and try to send it at a precise time. For example, the Controller might want to represent the effect of unconventional forces on a supply unit, by killing some of its trucks, or some of its supplies. The Controller could do so, but the player would not know about it.

In the current release of JTLS, a concept called an External Event has been introduced. This is not the SIMSCRIPT provided external event, which is totally scripted and rather inflexible. It is a data generated or controller generated event that changes either the course of the battle, or the players' knowledge of the battle.

There are ten possible events. There are three intelligence events: the area report of all units and targets in an area; the target report of the current status of all targets on a list; and the unit report of the current status of all units on a list. Three events cause a specified amount of damage to a specified unit. They are: damage combat systems, which kills a given number of the specified combat system at the unit; damage supplies, which destroys a given amount of the specified category of supply; and damage other capabilities, which destroys a specified number of cargo trucks, tanker trucks or aircraft at the unit. Other external events are the impact of a specified number of rounds of artillery at a location, the impact of a naval missile on a naval vessel, and the creation or movement of a target entity.

The extent to which this capability will be used remains to be seen. There was vehement user support for it. We believe it is a valuable concept and was a good change.

One thing the Controller could not originally do, was change the location of a unit. Several more or less clumsy workarounds were developed between 1984 and 1987. One solution was to set the unit speed to some very high value, set all movement related consumption to zero, and give the unit a withdraw or move order to the new location, waiting until the unit arrived, and then changing everything back. Another awkward solution was to cause the unit to be removed from the game and then resurrect it at the new location. Both of those methods were arduous, time consuming, and known to cause serious model problems.

The current release includes a true Magic Move capability for the Controller. Subject to some very loose reasonability constraints, the Controller can move a unit anywhere on the battlefield. There is no cost in time or other resources. The unit disappears from one location and appears in another. We have conducted two "beta test" exercises using the new version of JTLS. We are convinced that the Magic Move should have been added earlier.

4.0 OUTPUT AND REPORTS

4.1 OUTPUT AND MODEL PURPOSE

Although the format of output and reports is important for any combat modeling tool to insure that the required data are easily available for analysis, this topic is of utmost importance when discussing an interactive combat model. A combat simulation can be developed as an interactive model, if it is to be used for training or, in the case of an analytical tool, if it is too difficult to implement the human decision process in the detail required for the purpose of the analysis. In either case, interactive users are required to make the decisions needed to manipulate the entities and objects represented in the model.

The purpose of the combat model dictates when and how much data should be provided to the user. If the purpose of the model falls into the training category, the data provided should represent the type of data and information that would be received by decision makers under real world conditions. The data need to be presented in a realistic form, level of aggregation, and degree of completeness. On the other hand, an interactive model developed strictly for analysis purposes should concentrate on presenting the data in a manner that can be easily assimilated by the user so decisions can be made in a timely manner.

R&A feels that false data should not be provided under any circumstance. In the modeling environment, the player does not have all the other information that, in the real world, helps assess the validity of a questionable report. This makes a realistic assessment impossible.

4.2 INTERACTIVE MODELS

The following represents R&A's observations and conclusions about providing information and output in an interactive model.

4.2.1 Inforamtion Management Terminal (IMT)

In any interactive model there needs to be a capability to easily access the current situation of all forces and objects represented in the model. JTLS did not have this as part of the initial conceptual design. All information is presented to users in the form of individual messages. These messages are viewed from a player's input terminal or passed to the printer for a hard copy listing of the information.

One problem with this system is that the player can not easily call up the current situation, but must organize the available message traffic into an organized filing system. This does not match the data access capability of the decision maker represented by the JTLS user. The real world decision maker may not have automated access to situational information, but the decision maker does have a complete staff to organize the information, at the level represented so that it can be presented immediately on an as needed basis.

When designing JTLS, R&A extensively analyzed both MTM and an existing naval interactive model, the Warfare Environment Simulator (WES), to determine what their major problems were and to design solutions into JTLS. One major problem in MTM was its message handling capability. If the model started to make a large computation, an interface program could become grid locked. All messages had to be read prior to entering new orders. The game generated messages so fast, that the controller could not take control of the input terminal to stop or slow down the game. Without much difficulty R&A engineers realized that this was a problem that needed to be solved, and a design was developed in which the message read function was independent of the order entry function.

We never realized the reason the problem did not exist in WES was because it did not use a message capability. All information presented to players was contained in an online database that automatically updated a series of status boards which were continually available to the user. The status board capability allowed the commander to view the perceived status, location and available of assigned forces at any given time. We recognized the problem of viewing and obtaining information from MTM because it was a problem, but did not analyze why there was no problem in WES.

To correct this oversight, JTLS now has a full design of a new capability, designed by the R&A engineers, called the Information Management Terminal (IMT). All text messages to the players except for warning messages and messages concerning imminent problems will be eliminated from JTLS, and replaced by an online database capability that has a function similar to the WES capability. This example has implications beyond the information access conclusions for which it was presented. Another valuable lesson is that when reviewing previous modeling efforts, don't simply ask what they did wrong, also ask what they did right. That is too often overlooked because when something is done correctly it is all too easy to ignore.

4.2.2 Graphics

Graphics should have been part of the design from the beginning. There is no easy way to present location information to a user in other than a graphical format. We can not envision attempting to create an interactive combat model without a graphics capability in the future. Not only does it provide the user with essential information, but it helps tremendously with the testing and debugging of the system logic.

4.2.3 Model Versus Reporting Requirements

When developing model requirements, model designers and end users need to concentrate on insuring that the modeling requirements match the reporting requirements. For example, during requirements definition, the air functional experts explicitly decided that there was no need to provide a post strike damage assessment capability in JTLS. Their view was that pilot reports were not accurate, and the designed reconnaissance mission could provide unit and target status information as needed. As a consequence of this decision, the air staff member had no way to explicitly determine the specific damage resulting from an air strike.

Both the air and ground commander needed detailed damage information to get a better understanding of what was attacked, hit and killed in an air strike. The only information available to the decision makers was the capability or strength of the object. There was no method for them to know that an enemy unit that was recently attacked by an air strike and had lost 2% of its strength had in fact lost 3 tanks and 4 APCs.

This detailed information was and is important to the commander when appropriating future resources and planning the next day's air tasking order. For the commander to receive the information, the reconnaissance resources must be allocated. Therefore, the solution was not as simple as providing the required information. The solution to this problem was to upgrade the model to provide the required player information, but only if the resources to acquire the data had been allocated.

4.2.4 Units of Measure

The units of measure that are to be reported should be considered as data, and thus should be part of the database definition as would any other data item. All of the models that we have developed in the last six years have had as a design requirement that no data be included in the source code. We believe that this should be a mandatory requirement of any model. Unfortunately, it is not always easy to recognize data. A JTLS database can be created using any measurement system desired, but all reports list dry supplies in tons and wet supplies in gallons. If all data were entered in terms of kilograms and liters, the model would work properly, but the reports would indicate the wrong unit of measure.

4.2.5 Summary Information

An interactive combat model requires, as a minimum, that some rudimentary summary or statistical information be available to the player. R&A strongly suggested during the design phase that the JTLS post-processor be abandoned for an in-game processor that was capable of providing the user with any type of statistical information on a real time basis as the game was progressing. Since this suggestion was never implemented, we can not assess the advisability of the idea. We realize there is a tradeoff between the amount of summary data available to the player and the amount of time and other computing resources required to make this data available to the player. The question is: "How much summary data should be available?"

The answer may well be as simple as saying provide as much data as the computing resources can support. The problem with this answer is that the model can probably support extensive summary statistics during the initial stages of the development cycle, but that computing resource availability decreases as the model grows. In this case as much time will be spent in removing capabilities as adding capabilities. Although each model has different requirements, we believe that in the majority of large combat models, summary information of what happened during the last defined time period data is important. Summary of events that have happened since the beginning of the analysis time do not appear to be as important, and can wait until the analysis is complete and post processing activity is started.

An interactive user is much more interested in the summary of the situation over the last period, and a comparison of this data with the previous one or two periods to obtain trend information. Summary data any further back usually do not directly relate to the user's decision making process, and thus are not necessary as the game is executing.

4.2.6 Reporting

There are different reporting requirements at different times during the model development and gaming cycle. Different reporting capability is needed to debug the code, to conduct a functional validation of the model, to train users on the model operation, and to operate the model for its specified purpose. Because of this, various flags or switches are required, to determine what messages and reports should be generated.

This is nothing new and exciting for debug files. Almost all models have a debug printing capability that can be turned on and off to assist in the testing and debugging of the computer code. What is not usually recognized is that there are different reporting requirements depending on the familiarity of the users with the model.

There are numerous examples in JTLS in which messages were added or taken out depending on the familiarity of the sponsoring agency with the model. For example, as part of the first JTLS functional validation, the sponsoring agency insisted that every order received by the combat model be explicitly acknowledged. There was a "fear" that some of the orders were not being properly received by the model. The acknowledgement was programmed. Within several months, the sponsors realized that all orders were received by the models, and the acknowledge messages were a nuisance. Four years later, the messages are still in the system, but a capability to turn them off has been added.

4.2.7 Naming Conventions

Naming conventions should have consistent lengths for the named items that are going to be reported to players. JTLS permits various different length names for different objects. For example, units can have names nine characters long, target names can be fifteen, air missions thirteen, and air defense classes can have up to fifteen character names. This is an unnecessary programming problem. It is easy for a programmer to forget the maximum number of characters in an object name, and it is very important to insure that correct names are sent to players when creating information messages. Consistent name lengths also help in the process of constructing reports or displays that are easy to read.

3.0 DEVELOPMENT AND MAINTENANCE OF THE DATABASE

3.1 DATABASE MANAGEMENT

New hardware and software technologies over the last few years have pushed the importance of database design and management to the forefront of any large modeling effort. The days of a single process combat model for which all required data are held and accessed from memory are gone. The concepts of multiple process models, distributed computing systems, and parallel processing constructs hold the key to the future and expansion of current modeling capabilities. These types of systems require that the conceptual design concentrate on the definition of the underlying database organization. Without a well defined and well organized database structure, the system is bound to limit the model's future enhancement capability and its ability to use advanced system architectures effectively.

JTLS has such a database definition problem. It was one of the first large combat models that was divided into individual and independent processes and had, as part of its initial conceptual design, plans for distributed processing. It was the first such model that employed inter-process communications capabilities instead of disk files to pass data from one program to another. Unfortunately, the database implications behind this advancement were not realized early enough in the development cycle.

JTLS is currently not CPU bound but is restricted by the I/O capability of the host system. This is mainly due to the duplication of data in several databases which are required by the different processes within the JTLS system. There are no fewer than seven databases currently within the JTLS architecture that overlap and contain much of the same data.

3.1.1 SPP Database

This is a random access file that contains all of the data required by the CEP in a form that can be accessed by the SPP and changed by several different database developers at the same time.

3.1.2 Initilization Database

This is an ASCII file that contains the same information as the SPP database, but in an easy to access and human readable format.

3.1.3 Internal Memory Database

The initialization database is read by the CEP and held in memory throughout the execution of the model. The internal memory database grows as objects are created, due to player orders, and tracked by the model.

3.1.4 Graphics Database

This database contains object location and status information that is accessible by the graphics processors. All of the data in this database are also held in the internal memory database and it must therefore be maintained in both locations.

3.1.5 IMT Database

This database contains object location and status information that are to be eventually accessible by the IMT. All of the data in this database are also held in the internal memory database. The data in the graphics database are duplicated in this database. Thus the IMT database is a super set of the graphics database.

3.1.6 Post Processor Database

This database contains history information about the capabilities, status, and location of modeled objects. Originally this database was a single sequential ASCII file created by the CEP which was divided into another database of 50 separate sequential files by a program called the Pre-Post Processor. The single sequential database has been eliminated and the CEP now writes directly to the 50 individual sequential ASCII files.

3.1.7 Ingres Database

This database is a duplicate of the Post Processor database, but the data exist in a relational database format. The data in this form can be accessed simultaneously by different users.

The problem is obvious. The requirement to maintain, on a real time basis, the several system databases is causing an I/O bottle neck. Although we are not database experts, we believe that one of the underlying factors that initiated the entire problem is the inability of SIMSCRIPT to integrate or alter its memory management structure to match other defined database structures.

It is possible that other languages would afford a system designer more flexibility in developing an appropriate database structure. Unfortunately, we also believe that the only reason JTLS was up and running long before other models that started around the same time was because of the SIMSCRIPT language. Although we can not knowledgeably include suggestions on how to solve the database dilemma, we have outlined the characteristics that we feel are mandatory from an applications developer's point of view.

3.1.8 Accessible Files

Easy to access ASCII files should be used to interface two separate processes, if the processes are not run simultaneously. Thus we feel that the initialization database is most appropriate to take the output from the SPP and pass it to the CEP. We are strong supporters of editable, readable and accessible interface data files. There are too many times that we have found ourselves in a position which required that we directly access an initialization database. There are bound to be times when a database developer or an analyst needs to reach out and touch the data, or needs to quickly view the data in its raw form. If the database is held in a packed, binary or random access format, or the file contains large records, this can not be done.

3.1.9 ASCII File Size

ASCII interface data files should not be extremely large. The reason for using an ASCII data file is to insure that the data are easy to access and are readable. If the file becomes too large, these objectives are not met. Dividing the data into smaller, naturally related data sets is better.

For example, the JTLS initialization data file should be segmented into two or more files. The terrain data are already in a separate file, but the remainder of the data are in one large file. This file could easily be broken into a modeling parameter file, a unit file, a target file, and an operational plan file which would include the Time Phased Force Deployment Data (TPFDD) and the strategic logistics data. In addition, there is the added advantage that the modeling parameter file could be used for a variety of different scenarios, and the operational plan data, that is, at times, only accessible to government employees, can be segregated. This would leave the remainder of the data accessible to contractor personnel.

3.1.10 Independent Database Functions

As much as possible, the combat model should not be required to accomplish database functions. One of the concepts behind the JTLS design, which we feel is still valid, is that the combat model should concentrate on modeling. Any requirement for ancillary computations will reduce the computational resources needed to accomplish more detailed modeling constructs.

Thus, the decision to have the CEP create the individual 50 Post Processor files is reasonable because it does not require much more time to divide the data into the 50 individual files than to write the data to a single data file that would need to be broken up at a later time. Another alternative would have been to send the data from the CEP directly to the Ingres database. We feel that accessing the relational database system and placing individual records into Ingres would require an unacceptable computational burden on the CEP. It is also contrary to our first observation.

As a side note, the 50 file access capability that is now in JTLS does have a negative aspect. All 50 files must be open which greatly limits any future growth in debug files or other types of output planned for JTLS.

3.2 DATABASE DESIGN

Since we have said that little time was spent on database design there is not much that we can conclude except that the lack of design was an error. A database expert and a system software expert should have been involved much more in the design to insure some significant questions were asked and answered. One of these questions was, should a commercial database management system be used within JTLS? The answer to that question obviously changed during the first three years, since a specifically designed database management system was written for the SPP, but a commercial system was chosen for the Post-Processor.

R&A was opposed to the use of the commercial database system for the Post-processor because it added a substantial amount of overhead to the system that we believed was not needed or required for the Post-processor. We have wavered in that conviction because of some of the positive aspects of the commercial system, but are not thoroughly convinced that the decision to use the commercial system was correct. The following observations have been made concerning the use of the Ingres database management system for the Post-processor. We have left it to the reader to weigh the individual advantages and disadvantages and draw their own conclusions because we, as a team, have not been able to do so.

3.2.1 Maintenance

The commercial system does not require the personnel maintenance resources that a specifically designed system would require. The code which reads in the data and produces the menus from which users access the various queries needs to be maintained, but the upgrade and debugging of the commercial system does not require project team support, just money.

3.2.2 Return on Investment

It is not clear, given the wide distribution of the JTLS system, that there is any savings as far as monetary or maintenance resources are concerned. All JTLS users must pay for maintenance support of the Ingres system. If all of the yearly maintenance money were added together we expect that it could easily pay for the additional personnel required to upgrade, test and document a JTLS specific Post Processor database management system.

3.2.3 Implementation

The commercial system was without a doubt easier to get up and running given the requirement that multiple people needed to access the Post Processor data simultaneously.

3.2.4 Utilization

Unfortunately, the simultaneous access mode is not used extensively because it takes too long to get the history data added to the relational database system. Since this can not be done in a reasonable amount of time, individual players can not query the system at the end of each day to help plan for the next day. To our knowledge, it is run in a true post processor mode in which all analysis is done after the scenario is complete and plenty of time is available to query the system as required. Simultaneous user access appears not to be a mandatory requirement.

3.2.5 An Alternative

We believe that a JTLS specific post processor program could read in the history data and place it in their proper table structure much faster than Ingres.

3.2.6 Resident Data

Post processor queries would be able to execute faster if the data were resident in memory. Ingres can not make use of this capability, but a specifically designed program could. If it did, the program would need to reread the data each time it was executed. This time could be reduced, if a checkpointing capability were implemented, but some delay would occur each time the post processor was executed.

3.2.7 Required Expertise

We believe there is no difference in the level of expertise required to create a new query using the relational database system and what would have been required for a specifically developed system.

3.2.8 Efficiency

If the relational database system is used for the post processing function, it is available and can be used for the scenario preparation function. This does reduce the number of software systems that a JTLS manager, developer, and user are required to learn.

The current Post Processor is not very useful. The reasons for this are not easy to grasp, but the required data initialization time and query access time are major contributing factors. The third contributing factor that we can identify is that the Post Processor does not answer the specific questions that users want. We don't believe that the users have yet identified what they want, they simply know that it is not all contained within the system's current set of queries.

This was not the fault of the Post Processor . As hard as the team tried, they could not get the users to specifically define what type of information was required, or what type of analysis summary statistics were desired. The development team had individuals with past military expertise, but they did not have functional area experts who understood the entire OPLAN development and evaluation cycle. Since the direct experience was not available and exact specifications for reports were not developed, the did not want to be caught short at some later date without the data. This meant all data that could ever be used for analysis purposes was saved. This has added to the computational processing requirements.

One of the first areas that we would dedicate resources to speed up the Post Processor would be to closely evaluate the usefulness of each data item that is being saved. This would be a major undertaking and would require several analysts with extensive knowledge of how OPLAN evaluations are conducted. Doing so would help alleviate the impact of the contributing factors simultaneously.

3.3 DATABASE DEVELOPMENT

R&A has helped populate several major exercise databases for JTLS, and has developed all of the test and demonstration databases for the other models we have helped create. The following represents R&A's observations and conclusions about the creation of databases for various combat models.

3.3.1 Data Availability

One of the very first things any modeler learns in a simulation class is to use data which exists and can be obtained for the model. This seems like a very simple lesson, but the error is made time and time again. It is not always easy to recognize that the data are not available, or that there are modeling assumptions placed on the data used by the model that invalidate the data available from other sources.

For example, at the time of JTLS development the entire series of Vector models used Lanchestrian data. R&A engineers assumed that the one hour difference equation coefficient data could easily be obtained or calculated from the existing data used by the Vector models. This was not true. There are no models available to take 24 hour difference equations coefficients and break them into 1 hour difference equation coefficients. The answer is not as simple as dividing by 24, and the methodology to accomplish the task has yet to be developed.

An example of a simple data item that is invalidated by modeling assumptions is the range of an aircraft. The model does not consider the weight of an aircraft's weapon load when consuming fuel. Thus an aircraft carrying a small, light weight, weapon load should be able to travel further than the same aircraft flying with a heavier weapon load. Because of this simplifying assumption, what was expected to be a very simple data item to collect has become a challenge. The range data are now hand manipulated to develop average ranges based on the scenario and the theater of operations.

3.3.2 Terrain Data Generation

The software concepts for the automatic generation of terrain data was and is a tremendous idea. The terrain generation system uses image processing equipment to develop a pixel information database from hardcopy color maps. These data are combined with other available digitized terrain data as input to the generation program that uses a series of heuristics to interpret the data and assign terrain values to the hex overlay. Through programmer and user evaluation, the program's heuristics were improved as more databases were built using the system.

Managements only problem with the terrain generation program was that they did not implement the last step of any software development project: sell, sell, sell. R&A engineers were shocked to learn a full year after JCS/J-8 took over the JTLS project that project managers did not know that an automated terrain generation package existed, and they have shown little interest in investigating the capability.

3.3.3 Tools

More time needs to be spent developing tools to take existing automated databases and translate them into the data required by a combat model. This is more than reformatting the data to match the data structure of the combat model. This requires that personnel familiar with the assumptions and meaning of the existing military databases work closely with the modelers who are familiar with the assumptions and meanings of the model data to determine what needs to be done to insure that not only the format matches but that the assumptions also match.

An alternative is to change the combat model to read existing operational databases. This approach sounds intuitively appealing, but we feel that such an approach is impossible. As we have already mentioned the assumptions on the data are bound to be different and thus the data will need to be manipulated in some manner before they are of use to the model. Furthermore, since a model is a subset of reality, the military databases, in all likelihood, contain more data than is required by the model. Combat models have enough data problems, and do not need to save or keep track of unneeded or unwanted data.

Such tools would be extremely helpful when building and gathering unit data, target data, the TPFDD data and the strategic resupply data for JTLS.

3.3.4 Prototyping

Database data should use more prototyping to help cut down on the database size, and the data entry requirements. By prototypeing we mean that common entity data should not be stored on the entity, but stored on an entity that represents the common data.

For example, there is a lot of unit similar data in JTLS that could have been prototyped. Each unit does not need to have its own supply consumption data. These data could have been held by a prototype entity which would be assigned to a unit through the use of a prototype number. If all brigades had the same consumption data, the brigade unit would not hold the consumption data, but a consumption data entity would hold the data. All brigades would then point to the single consumption data entity instead of repeating the same data numerous times in the database. This does not reduce the possible level of detail because in the worst case each unit would have its own prototype and the database would be as large as it is now.

There are two ways that prototype data could be implemented. The first is fairly efficient if a commercial database management system is used for the scenario preparation function. Using this method, the relational database system would have defined for each category of prototype information a separate table. Using the example from above, the supply consumption prototype information would be held in a table by itself. When a unit is built it is assigned a consumption prototype. When the data are written to the interface file, the appropriate consumption data are added to the definition of each unit. The interface file is not any smaller than a non-prototype database, but the combat model does not need to include the prototyping data structures which can result in accessing data indirectly through several levels of prototype structures.

The second method is to include the prototyping structures in the combat modeling database. The interface database will be smaller, but the indirect access problem should be watched closely. It is possible to create the prototype structures, assign the information to the objects that use the prototype information and then destroy the prototype data structures all within the combat model. This allows the interface data file to remain small and insures that there are no long term time degradation problems due to the indirect access problem, but the internal memory used is again as it was whether prototyping was used or not.

3.4 TEST DATABASE

Just as the characteristics of a combat model change based on the purpose for using the model, the characteristics of a database change based on the purpose for using the database. A test database should be small, compact and have at least two objects of each type represented in the combat model. A test database does not need to be realistic, in fact it is probably better if it is not based on real world data. A tester needs a database that can be easily manipulated and produce pathological situations to test the bounds of the logic. Thus air weapons that kill absolutely everything and absolutely nothing are required to allow the tester to enter boundary condition logic that can happen when using a stochastic model.

We have seen complicated and extensive test databases that burden the system unnecessarily. For module and logic testing, the databases should be as small as possible, but still provide a robust cross section of objects to insure all logic avenues can be entered. If the test database is large, the tester spends too much time reading in the data, initializing the system, and processing the unneeded interactions.

A functional validation database needs to be realistic and reasonably robust. It should match the size of the database established during the initial phases of model's functional requirements study. Code and logic errors will always be found when working with the larger database, but the problems found with this type of database do not warrant using a database of this size for coder testing.

An operational database is always going to be larger than the developers expected or were led to believe. We have yet to encounter a model where this was not true. The JTLS design specifically states that playing areas would never need to be greater than 2000NM by 2000NM boxes. Recently, the United States European Command was disappointed to learn that they could not create a 5000NM by 5000NM playing area because of hex distortion. PLAN ACCORDINGLY!

6.0 SOFTWARE LIFECYCLE MANAGEMENT

6.1 SYSTEM SELECTION

The process of software life cycle management really begins with the selection of the hardware. As with most projects, the selection of the hardware for JTLS was influenced by a number of factors. At the Army War College, MTM was resident on a Honeywell 6000, and that was the machine upon which the SFD was to be executed. The analyst/programmer responsible for the design and majority of the coding for the SFD was intimately familiar with IBM mainframe machines, and with Digital Equipment Corporation Mini-computers. The most available resource was a DEC 11/780 operating twelve hours a day (midnight to noon) under the VMS operating system. That machine was selected for the development of Enhanced MTM (EMTM) for the SFD. For the SFD, the EMTM code was ported to the Honeywell 6000. At the end of the SFD, the team was quite familiar with that machine also.

By the time a hardware decision was required, it was apparent that if a prototype was to be up and running in less than a year, it would not be possible to include full air, land and naval representations. Based on the status and availability of interactive models, and the desires of the sponsors, the decided to implement complete air and land functions, and a skeleton naval module. The intent was to connect with some existing interactive naval model, to get the naval functionality. The most likely candidate was the Warfare Environment Simulator (WES) model. WES operated on a DEC 11/70, and was being ported to the VAX 11/780, under the VMS operating system.

At the same time, there was a concept being discussed between the NPS and DARPA that was referred to as the C3 Laboratory Suite concept. NPS had been selected as the site of the DOD Command Control, and Communication (C^3) graduate education effort earlier, and DARPA had funded an experimental computer laboratory at for C^3 experimentation. The C^3 suite was composed of a VAX 11/780, supported by RAMTEK graphics and other peripheral equipment. Within the modeling community, the VAX architecture was being widely accepted. The Computer Science Department had a VAX 11/780 that had some excess available processing time. The US Army TRADOC Operations Research Activity (TORA), the TRADOC Combined Arms Operations Research Activity (CAORA), and US Army CAA all either had installed or were installing VAX 11/780s.

The VAX 11/780 was a relatively new architecture, that appeared to have significant potential for growth in capability and expansion in size. The machine had only been widely marketed for a few years.

Finally, the machine was within the budgetary constraints of the project. The USREDCOM Contingency Analysis Subtask project was not critically short of funds, but also did not have large amounts of discretionary funding. The implementation plan required provision of a computer at NPS, another one at JPL, both for development of the system; and eventually a suite of equipment for the principal sponsor, USREDCOM. The procurement of all three seemed feasible.

It was some combination of all of the above factors that led to the decision to implement the JTLS system on a VAX 11/780. The precise date of the decision is lost, but the decision had been made before January 1983.

Discussing the software and hardware separately is misleading. The fact that the VMS operating system is as user friendly as it is, and had as much capability as it did was certainly a factor in the selection of the VAX architecture. On the VAX architecture, both VMS and UNIX were available. The analyst/modeler was familiar with both operating systems, and found VMS more user friendly, and sufficiently capable for the required functions.

There was never any serious question as to whether the combat model would be written in a general purpose language or a simulation language. If the system was to be up and running quickly, the overhead of building a simulation structure in a general purpose language was unacceptable. The only question was which simulation language?

The US Army was in the middle of a significant effort to upgrade its combat modeling capability. Several new Army models were either operational or being developed on DEC VAX equipment in SIMSCRIPT II.5. There was a requirement that the selected language be capable of linking in modules written in other languages. SIMSCRIPT has this capability. Senior analysts at R&A and JPL, as well as at the sponsoring agencies, were familiar with SIMSCRIPT, and had used it. None of them were familiar with any large scale combat modeling project that had been successfully completed in any other simulation language, but there were such successes in SIMSCRIPT II.5.

The choice of SIMSCRIPT was easy, and straightforward. There have been times over the last five years when we have been quite dissatisfied with the support from the language vendor, and with some limits of the language, as we have pushed the language to its limits. There have been times when we would have preferred that the hardware and software were the responsibility of the same organization, to preclude the occasional debate as to whether a problem is caused by the hardware, the software, or the operating system.

We have written simulations in other languages since we started JTLS, but only when forced to do so by circumstances. Some have been more successful than others, but we believe that all would have been better written in a simulation language.

On balance, we believe that the selection of language and operating system was a good one. If SIMSCRIPT had not been available on the DEC VAX series, we might well have chosen another hardware suite.

6.2 DOCUMENTATION

JTLS is at least as well documented as any existing combat model. The suite of documentation available when the reins were passed from USREDCOM to JCS/JAD was fairly complete, professional, and quite nearly up to date. The content does not match the DOD software standards precisely, as we understand them. We believe that it is much better suited to the intended use than some of the "required" documentation.

One of the strong points of the JTLS documentation is that it was written by the analysts who wrote the system being documented. Jokes about programmers, analysts and documentation abound, and they have the rueful tone of humor that is true. In general, people who like to design and write computer programs, and combat models in particular, do not like to write documentation. They persist in stating that the only way to know what the model does is to read the source code. Without delving into that discussion, we believe that the best way to document a model is to have the analyst/modelers document it. Getting them to do so is an opportunity for management to excel.

Editors who speak non-jargon, and other editing/publishing resources are needed to refine the product of the efforts of the analysts. In all cases, however, the analysts should provide the initial input, and review the final product to ensure that some well meaning editor has not changed the meaning of the original. As a bonus to technical correctness, analysts, in the process of documenting their own code, will sometimes discern previously overlooked errors or omissions in the code. We have almost always done so.

Two documents that we believe are indispensable for any computerized combat model are an Analyst Guide, and for multiple program systems, an Interface Specification.

The Analyst Guide should not be an advanced player guide. It should be a technical modeling document, that discusses the assumptions made, the algorithms used, the derivations made from the algorithms, and the simplifications of the algorithms made for expediency, speed or solubility. The Analyst Guide should document ideas, not code. Properly written and documented code does not need another document that says "Lines 50 to 112 solve the quadratic equation for the real roots."

The Analyst Guide should discuss the use of the solution, and perhaps the implications if there are no real roots. Attempts to bowdlerize the Analyst Guide to the level of some least common denominator reader/user, will, we believe, result in the statement that the only way to understand what the program is doing is to read the code being precisely true.

In JTLS, we adopted the convention that the analyst/programmer whose program or module was to receive data from another program or module was responsible for specifying the content and format of the data. The specifying was usually done in concert with the sender/provider of the data, but it was the receivers' responsibility. We believe that this convention was very successful, and worked better than either a joint responsibility, or having the sender specify the data format and content.

The result of this convention was a document that specified the content and format of all data passed between programs. We call it the Interface Specification, but its full name was JTLS Software Engineer Maintenance Manual, Volume 1, Model Overview and Interface Specification. This document is beyond price if a new organization is to take over a system, but again, one of its principal values lies in the process of producing it and explicitly writing what is being passed between programs and between modules within programs.

6.3 SOFTWARE DESIGN AND DOCUMENTATION LANGUAGE

An early decision in the design process was to use the Software Design and Documentation Language (SDDL) as a documentation tool for the programs of JTLS. That requirement was followed in the CEP and SVP through and including release 1.3. We do not believe it ever received even lip service in the other JTLS programs. We are confident that the requirement to keep the source code in a SDDL acceptable format that long was an error.

For very low level languages, such as C and FORTRAN IV, a design and documentation tool provides a useful adjunct to the design process. For sufficiently high level languages, the usefulness of another design and documentation tool is more questionable, especially beyond the very early design phases. SIMSCRIPT is at least on the edge of being a sufficiently high level language.

The version of SDDL available for the project was the latest one available in 1983. It had a few shortcomings. For example, it was not capable of distinguishing between the two SIMSCRIPT invocations CALL and SCHEDULE. Standard SIMSCRIPT syntax for invoking the creation and filing of a new event is:

- SCHEDULE A UNIT.ARRIVE IN T DAYS

Where UNIT.ARRIVE is the name of a predefined event. The word A may be replaced by AN or THE, but one of those forms must be present. Despite the fact that SDDL was touted as being ideal for use with SIMSCRIPT, it was not capable of recognizing that the article was a "noise word". The JTLS preamble still has a DEFINE TO MEAN in it that substitutes SCHEDULE A for SCHEDULE during the first pass through the compiler. SDDL was also incapable of distinguishing between global variables and DEFINE TO MEANs.

The output routines of SDDL had a hard-wired limit of one thousand pages and ten thousand lines of code. JTLS passed those limits before September of 1983. Attempting to find a routine or variable using the SDDL references frequently led to such questions as "Which page 219 is it on?" or "Which Line 1812 is it talking about?"

Individually, these seem like minor annoyances. The problem was the invidious comparison with SIMSCRIPT. SIMSCRIPT provides detailed local and global cross references, which include all calls, schedules, invocations, and distinguishes between the locations at which variables are changed and those at which they are merely accessed. This is a very useful facility for the programmer.

SDDL did provide a very nice automated indentation feature. This feature permitted automatic formatting of the multiple condition and loop structures. It also provided a visual cue (an arrow) on the printout to all invocations and locations from which the routine could be exited. The automated indentation proved a mixed blessing, as discussed in the next section. A programming standard that required single entry and exit points for each routine vitiated the usefulness of the exit marker. The invocation arrow was very useful, although a way to distinguish between calls, and schedules would have been an improvement. The failure to recognize function subroutine calls was irritating. A thorough module invocation tree was also provided. It proved very useful.

As soon as the project got far enough along that any representation of the code was being entered on a machine, that representation was entered so as to be compilable using SIMSCRIPT. At that point the utility of the version of SDDL to which we had access was largely past. With a high level language, and our policy of having the analyst/modeler do much of the complicated coding, the absence of a tool that provided the facilities that SDDL provided would not have caused any deleterious effects.

6.4 PROGRAMMING STANDARDS

Programming standards are a necessary fact of life. They are difficult to enforce, especially when the individuals actually doing the coding are fairly senior analysts. They are especially valuable when they exist and are enforced for a program or system that is being developed by more than one group, and during the maintenance and enhancement phase of the project.

6.4.1 Data Packing

SIMSCRIPT permits the explicit packing of integer (and pointer) data down to the byte level in arrays, and to the individual bit level for entities. Packing data precludes the waste of memory involved in using an entire thirty-two bit word to store a value that can only take on values of zero or one.

The storage requirement for a typical combat model entity could probably be reduced by one third or more by using bit packing. SIMSCRIPT provides what appear to be very efficient "unpacking" routines. In at least one large combat model, preliminary investigation had been conducted to compare model performance with packed and with unpacked data. The results indicated that the model did not run more slowly to any significant degree when the data were packed.

In the course of that analysis it was, however, discovered that some packed data items were packed so that the maximum value was smaller than the values being used in operational databases. This does not cause a SIMSCRIPT error, although the value stored simply has the high order bits dropped.

Data packing also places constraints on the size of the values of the packed data elements. We decided that memory was not important enough to place the limitations on the data or to risk the possible error. Data packing was and is prohibited in JTLS. We believe that to have been a good decision. In 1984, JTLS was installed on the CAA VAX 11/780, with two megabytes of dynamic RAM. In October of 1988 it will be used on the USEUCOM VAX 8650, with one hundred thirty-two megabytes of memory, scheduled for upgrade to two hundred fifty-six megabytes. Memory limitations have never been a serious problem for JTLS.

6.4.2 Source Code Formatting

As noted in the preceding section, SDDL provides a very nice capability to automatically indent the source code print as each subsequent conditional section or loop structure is encountered, and to terminate the indentation at the end of the structure. The source code, once processed by SDDL is very readable, especially given the liberal use of the DEFINE TO MEAN capability of SIMSCRIPT and readable variable names.

Unfortunately, SIMSCRIPT does not provide any automatic indenting capability. The SDDL processed source code that is so readable and easy to follow because of its indentation structure is very difficult to follow in the print of the compile, because all the executable lines start in the first column and the structure is extremely well concealed. Two solutions are possible. Either turn off the automatic indentation feature of SDDL, and indent the source code itself; or every time a compile of a routine is performed, process it through SDDL also. As the size of the executable grew, the second option became an unacceptable use of computing resources and analyst time.

We opted for the actual indentation of the source code. This makes it easy to follow the logic even in the uncompiled version of the source code, but takes away one of the advantages of SDDL. There are still a few routines in the CEP where all the lines start in the first column and there is no indentation. When configuration management procedures let us into the routine, we indent them.

6.4.3 Variable Naming Standards

We count two clear successes and one missed opportunity in the area of variable naming. Early on a standard was adopted that specifically did not limit the length of variable names to any arbitrary size, and suggested strongly that the variable names be as close to self explanatory as possible. Some were better than others.

The structure that contains the data that describe the characteristics of the combat systems in the simulation probably should have been a permanent entity rather than an array. Given that it was an array, the name COMBAT SYSTEM CHARACTERISTICS is relatively easy to understand. AIRCRAFT CHARACTERISTICS, SUPPLY STATUS, CARGO TRUCK PK ARTY and others are easy to understand.

We also adopted a standard that the name of each attribute of an events or entity would begin with a two or three letter prefix that indicated exactly the type of entity to which the attribute belonged. The prefix was followed by a period. If an attribute starts with "AS." you can absolutely rely on the fact that it is an attribute of an AIR SQUADRON surrogate entity. If it starts with "UT.", it belongs to a unit entity without exception, and "AM." indicates that the owning entity is an AIR MISSION.

The relatively small number of exceptions have not caused any real problems, but probably only because they are exceptions, and are known. We hope to get the last of them out when we remove the surrogate entities in the fall of 1988. Those two areas we count as signal successes. We not only recommend them, we have used them in all the other models we have created during the last five years.

We missed an opportunity in the area of specifying standards for DEFINE TO MEAN macros. We did specify enough of them to help with code readability. GREATER.THAN, IS.GREATER.THAN, and IS.LATER.THAN are all defined to mean ">". We believe that the phrase

- IF TIME IS.LATER.THAN LATEST.TIME

is easier to read than most other formulations, even for a Real Programmer. For that type of DEFINE TO MEAN the demand for a quick recognition capability is not as stringent as it is for examples like TYPE, UNLIMITED, TANKS and FUEL. The latter type, typically used as indexes to arrays, or maximum values to which to compare some value, need to be easily recognizable, and should take a form that is noticeably different from other variables, both local and global. A SIMSCRIPT program with a global DEFINE TO MEAN of TYPE cannot have any global or local variable called TYPE. This is another reason to have a specific form for the DEFINE TO MEAN.

For several years, CACI engineers, the developers of SIMSCRIPT have advocated the use of a single leading period to indicate that the reference is to a DEFINE TO MEAN. ".TYPE" is not much more difficult to read than "TYPE", and could easily indicate to the reader that this is a DEFINE TO MEAN.

On one of our other projects, we worked on a model that used a single leading period to indicate that the item was a local variable, and two or three leading periods to indicate a DEFINE TO MEAN. We didn't and don't particularly like that style.

We do recommend the adoption of an easy to recognize standard format for the DEFINE TO MEAN. We have no better suggestion than the leading period.

6.5 BULLET-PROOFING

An unresolved implementation issue exists in R&A as to the extent to which it is appropriate to "bullet-proof" a model. By bullet proofing we mean the protection against errors that should never occur. If a target is not ever supposed to be referenced in a list once it has been destroyed, then checking that it has not been destroyed before accessing its attributes is bullet proofing. If an air mission is never supposed to be flying if it has zero aircraft, then checking the number of aircraft before dividing by it is bullet proofing.

The two schools of thought agree that model crashes are unpleasant. One school holds that if the model is to be of any use for analysis, the errors cannot be tolerated. The types of errors that would cause the model to crash are severe enough to invalidate the analysis in any event. Quite correctly, they note that it is easy to overlook a message to the controller that reports that some internal error has been located that renders the last three days' work meaningless. No one, they point out, ignores a crash. Clearly there is merit in this position. A crash makes it immediately obvious that something needs to be fixed.

The other school holds that it is better to err on the side of preventing the crash. When JTLS is being used, whether as an analysis tool or not, there is usually a reasonably large number of fairly senior military officers involved in the game play, and stopping to repair a crash in the middle of the effort is wasteful of their time, bad press for the model, and may endanger the effort. There is some merit in this position also.

If the model were always being used in a pure analysis mode, or always in an exercise driver or training mode, it would be quite a bit less difficult to arrive at a truly satisfactory solution. In the best of all possible worlds, the answer would be easy. If there were a mistake, we would let the model crash so we could find the error and fix it. If the answer were wrong, we wouldn't want it anyway, no matter what was the purpose of the use of the model.

In cases where the answer to the question being asked comes from the players, rather than from the model itself, the issue of bullet proofing is more troublesome. If the model is only providing stimuli to the players, it may be more appropriate to bullet proof the model.

We have no answer to this dilemma.

6.6 FUNCTIONAL VALIDATION THROUGHOUT DEVELOPMENT

The concept of a combat model functional validation is a test wargame at which the sponsoring agencies provide the players to exercise the capabilities of the modeling system. Its function is to demonstrate that the required capabilities have been provided. Usually, such a test begins with a set of scripted tests, and concludes with an essentially free play wargame.

We have discussed the requirement for an experienced, skilled test team in earlier sections. The functional validation provides the final test of the product before release. As the test team brings a different point of view to the testing than the coders and implementors, so the end user brings a different perspective than the test team. The operational user brings a knowledge of military doctrine and tactics that is more current than either the analyst/modeler or the test team. Ideally, this team is composed of operators, those who execute in the real world what the model emulates. They ask the model to do things that neither the designers nor the test team ever considered. The functional validation performed with the end users having their hands on the system is the acid test of the product. It is expensive.

A good functional validation requires at least two weeks, one of scripted tests and one of free play. Depending on the size of the model, more time may be required. Representatives of the development team must be present, and for JTLS, twelve to fifteen representatives is not excessive. The model will rarely emerge from this test unscathed.

We believe that every release should be preceded by a major functional validation at which the specified deliverable capabilities of the system are tested by the users, with their hands on the system.

6.7 CONFIGURATION MANAGEMENT

Some time in the process of going from conceptual design to mature product a model must be brought under configuration management. By that we mean specifically that the only changes that can be made to the source code of the programs that make up the system are those that are approved through a Configuration Management Process. Failure to bring a model under configuration management will lead quickly to multiple versions of the model. This will require an inquiry concerning each analysis that is performed with the model, whether an approved version was used, and if so, which one.

From the point of view of the analyst/modeler, configuration management is an imposition made by management to preclude the model from being as good as possible, as soon as possible. From the point of view of management, configuration management is a last ditch effort to retain control of a group of technicians who have no appreciation of resource constraints, delivery schedules, or software product maintenance.

The truth lies somewhere between those two poles. Most analyst/modelers are quite willing to tackle any design coding problem and pursue it to the death. The imposition of constraints as to what can and cannot be changed is, from their point of view, unnecessarily restrictive. It keeps them from fixing known errors in the model! It also keeps them from introducing new errors into the model.

Most managers do not really think of their analysts as running wild in the code, spending hours and hours of expensive analyst time and CPU time tweaking the model to get that last ounce of performance out of it. They are the ones who have to deliver the product within a time and resource schedule, and to make sure that once it is approved and delivered, no changes are made to invalidate the acceptance of the product.

Once the model is under configuration management, the cost of even the smallest change increases significantly. Placing the model under configuration management too early in its life cycle is an expensive mistake.

In spring of 1984, the time between periodic calculations of consumption of supplies was changed from a global variable, to an attribute of the consuming unit. When the change was made, the global data value was not removed from the database. By the time the omission was noticed the system was under an initial form of configuration control. The parameter could not be removed without an Engineering Change Proposal being processed and approved.

The data parameter is still in the database. The administrative overhead of removing it has precluded doing so. It doesn't really hurt anything. It is read in, never accessed again, except at an ASCII checkpoint, where it is written out, so that it can be read in again. It simply is untidy.

We believe that JTLS was brought under configuration management too early, probably by about six months. We should have waited until after the November 1984 exercise at the AWC to baseline the model. We strongly recommend not placing the model under configuration management until it is absolutely mandatory.

7.0 CONCLUSIONS

Our conclusions concerning the design, development and implementation of a large scale combat model have been presented in all the preceding sections. Most of them require technical judgement and the ability to make trade-offs. The most important conclusion that R&A can draw given our experience base in developing combat models is to start the combat model simple and grow. It is important to start a major project with a small well defined list of requirements and get the model up and running with a rudimentary capability. This phase should not last much more than a year. If it lasts longer than that, project momentum is lost.

The design of the database and data accessibility have become more important than the models and algorithms that are to be used in the model. This portion of the conceptual design must be extensively reviewed because it is the backbone on which the remainder of the modeling system is built. If not built properly, the life cycle of the model will be shortened because the database and data organization will not be able to support the increased "weight" of the model as it grows and progresses to maturity.

From there, a good model development project simply requires highly motivated, dedicated personnel with a strong analysis background, imagination and good communication skills combined in a diverse but supportive team arrangement. The team needs to include the end user, functional area experts, modelers, and computer scientists. Management needs to support such a group with a mixture of strong guidance and professional freedom, and needs to create a non-combative working environment based on communication, trust, and project pride.