A CASE FOR MICRO-TRAINERS

David R. Pratt, Ph.D.
SAIC
Orlando, FL

Amy E. Henninger, Ph.D.
Soar Technology, Inc
Orlando, FL

ABSTRACT

Over the last several years, there has been a major push to develop large-scale interoperable simulators and simulations that cover the depth and breadth of military operations. While these systems provide an ability to do end-to-end mission emulation, they tend to require elaborate software development efforts. The requirements base for these systems is quite large and can contain a variety of requirements at a number of different levels. Oftentimes, the harmonization of these competing, and sometimes conflicting, requirements results in compromises in the system that can have a negative impact on system performance, program cost and program schedule. The ultimate effect is a negative impact on readiness, by delaying or depriving the training audience of its tools. Consequently, even though these systems are able to provide training in the coordination aspects of warfare, it is, to some extent, at the expense of functionality in individual training tasks. That is, as "Swiss Army knife" systems, these trainers are able to provide useful and convenient training tools, however, for any given individual task, the single, monolithic, meta-system does not perform as well as a system developed solely for that task. This forms the basis of our argument: that we should augment the large scale M&S system, with families of task specific Micro-Training devices. As task-specific devices, they are smaller, less expensive and better suited to a given, specific task than a larger system built on a series of compromises.

ABOUT THE AUTHORS

Dr. David Pratt is currently the Vice President and Director for Technology for the Orlando Group of Science Applications International Corporation (SAIC), a Fortune 500 company. Dave's responsibilities include developing and fostering continued leading edge Information Technology (IT) and Modeling and Simulation (M&S) technologies. Supporting the Group headquarters in Orlando and offices in Tallahassee, eight other states and four overseas locations, he provides both strategic and tactical guidance in both technical and programmatic matters. Before joining SAIC, Dr. Pratt was the Joint Simulation System (JSIMS) Technical Director for the largest simulation software effort ever undertaken by the Department of Defense. Former Tenured Associate Professor of Computer Science at the Naval Postgraduate School and Adjunct Teaching Instructor at the University of Central Florida, he has an extensive academic background that includes over 50 publications.

Dr. Amy Henninger is a Senior Scientist at Soar Technology, Inc. and has worked in the military simulation community previously as a staff scientist at Science Applications International Corporation, a research assistant at the Institute for Simulation and Training, and a Research Fellow for the Army Research Institute. Henninger earned B.S. degrees in Psychology, Industrial Engineering, and Mathematics from Southern Illinois University, an M.S. in Engineering Management from Florida Institute of Technology, and an M.S. and Ph.D. in Computer Engineering from the University of Central Florida. Her research interests are in the areas of machine learning and statistical modeling. Currently, Dr. Henninger serves as Adjunct Instructor in the Modeling and Simulation Graduate Degree Program at the University of Central Florida.

A CASE FOR MICRO-TRAINERS

David Pratt, Ph.D.
SAIC
Orlando, FL

Amy E. Henninger, Ph.D.
Soar Technology, Inc
Orlando, FL

INTRODUCTION

Over the last several years, there has been a major push to develop large-scale interoperable simulator and simulations that cover the depth and breadth of military operations. This paper presents another approach – Micro Trainers, which could be used to complement existing systems. After providing a definition of a Micro Trainer and motivating the need for Micro Trainers, this paper focuses on three major areas: System Development and Maintenance Costs, Training Effectiveness, and System Availability. Moreover, we examine the issue of interoperability, and consider how it affects design and implementation.

Definition

We define a Micro Trainer as one system that covers a training need related to a specific use case, where a use case is a description of user initiated events and user-system interactions (Jacobson, 1992). In the training and simulation community, for instance, one example of a Micro Trainer would be a part task trainer (e.g., trainers for tactical and acoustic displays, radar, electro optics, flight/avionic management, etc.). According to MIL HDBK 1379-4, a part task trainer is "a device that permits selected aspects of a task to be practiced independently of other elements of the task". Thus, one of the pedagogical advantages of part task trainers is that they can break down a physical task into easily understood pieces of information and allow for repeated practice of that task in a low stress environment. Typically, the physical configuration of part task trainers consists of workstations under the control of an instructor facility and these trainers specifically support individual training.

Investigators have demonstrated the benefits of using part task trainers in military training (Chatham and Braddock, 2001). In our model, these benefits would also apply generally to the notion of a Micro Trainer. But, while the training and simulation community has formally adopted the concept of doing part task training (MIL HDBK 1379-4) with part task trainers, there is really no counterpart for mission training. That is, no construct exists to describe simulations designed to support specific mission threads or phases of a mission. To describe such a construct, we coin the phrase "mission thread trainer".

In our taxonomy, mission thread trainers are characterized by attributes such as their:

1. ability to be deployed quickly
2. disposable nature
3. high amount of component reuse
4. selective fidelity

We liken a mission thread trainer to a part task trainer in that it is designed to support a specific use case, but distinguish it from a part-task trainer in that the use case it covers relates to a specific mission thread over time as opposed to a specific skill or task (e.g., gunnery or driver trainer).

One example of a mission thread trainer is Microsoft Flight Simulation, used by the Navy and Air Force for pilot training. As a training tool, this simulation has reduced level of fidelity for non-essential aspects of mission. For example, a high-resolution terrain data base would not add value to this trainer, given the use case. Another example of a mission thread trainer, Delta Force by Novalogic, was designed to take down a building and specifically intended for mission training as opposed to combined arms training. In this case, while it would need a high-resolution terrain data base, it would not require a full-blown M-16 ballistics model. These examples illustrate one of the central characteristics of a mission thread trainer – the ability to reduce model complexity (i.e., fidelity, in these instance) according to use-case specific applications.

Figure 1 graphically illustrates the mission thread trainer concept.

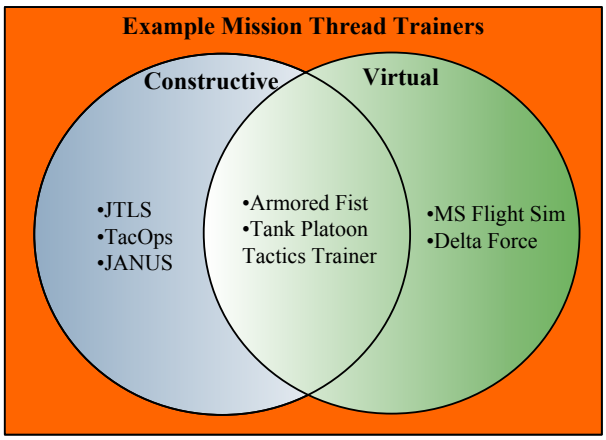

Figure 1. Mission Thread Trainer Concept

The mission thread trainers we explore in this paper exist in either the constructive domain or the virtual domain. One could, however, also envision a mission thread trainer operating in the live domain. For example, paintball used for tactics training could be considered an instance of "live" mission thread training.

Joint Theater Level Simulation (JTLS) started out life as a simple model to exercise the staff of USREDCOM. Developed in roughly one year by a small team for a given task on a given platform, it clearly fits in the Mission Thread Trainer category. However, like most successful systems, it has been expanded and modififed so many times, it no longer fits into the micro-trainer realm (Roland 2002). The same applies to the Janus system.

TacOps is an electronic version of the traditional hex-based strategy board game. It makes no pretense at having an accurate terrain or weapon effects model. Rather, it is designed to solely exercise the basics of tactics. Like TacOps, Armored Fist is a commercially available war game. As a tank platoon commander, the player must maneuver his tanks in series of engagements. In doing so, it simplifies the dynamics, weapons, terrain, logistics and tactics to provide a visually appealing environment. The idea here is to provide a popped hatch view of the world and be able to control forces in a movement to contact and during the engagement, not all the more mundane things (logistics, road march, air defense, dealing with mechanical failures, etc.) of being a tank platoon commander.

On the pure virtual side, games such as MS Flight Simulator and Delta Force are very popular and provide great experiences in their limited domain. While Delta Force might provide a reasonable simulation of a building clearing, it does not provide the insertion and extraction and nation building missions associated with the SOF community.

Another classic example of a virtual micro-trainer is the PowerScene/TopScene system. This system provides pilots a view of photo draped gridded elevation terrain with some of the more military relevant objects placed on it. It is designed solely for the purpose of familiarizing the pilots with the terrain they are about to fly over. It does not have duplicates of the real aircraft's controls, dynamics, or full mission capability. It was designed for a single purpose and performs effectively within those design specifications.

Motivation

The Department of Defense is actively pursuing advances in modeling and simulation technology to provide the ability to link live, virtual, and constructive simulations, and allow warfighters to train in a realistic, fully integrated environment that supports the entire spectrum of training (Part II: Today's Armed Forces Readiness, DoD Published Report). That spectrum is vast and requires coverage ranging from the individual-level to campaign-level mission rehearsal. As demonstrated in a DARPA/USSOCOM call for ideas on how to improve training, even just one specialized domain, Special Forces, for example, requires a host of different training platforms that are not routinely part of mainstream training. As seen at http://safe.sysplan.com/scihelpamerica/index_.html :

> "*Pilots in SOF units must have access to the latest in high fidelity, virtual reality mission rehearsal systems. They must be able to fly profiles in real world databases not ordinarily found in common simulation. Maritime SOF must be able to accurately navigate small boats in up to Sea State 3, at night and in hazards and executing all of this at extremely high speeds (inducing a whole new set of shock mitigation problems). This requires training far above that required of other navy coxswain/helmsmen. Ground forces too… must be proficient in high altitude, high/low-opening parachuting. This requires a unique training system that can train the soldier to navigate to a precise landing zone, at night, in zero-zero visibility*."

While not all of these tasks lend themselves to human-in-the-loop (HITL), virtual training, they do illustrate the vastness of the training domain.

Moving from specialized task training to specialized mission training, McDonald et al (2001) recognized the need to support mission rehearsal training through design and implementation of a constructive CGF system for USAF security forces. Critical to success of this system was the fact that it be affordable and be able to train decision makers on how to plan and execute air base defense missions, while still located at their home stations. As such, it would allow decision makers to maintain team skills between training rotations at regional training centers once every three years. To achieve this, McDonald designed a very specific set of behaviors for VR Forces, a COTS CGF system. This approach, contrary to other development approaches used in mainstream CGF systems, was very focused and specific to needs of individual program. That is, instead of developing a large-scale system that encompasses a number of unneccessary requirements for this application, McDonald developed a smaller, less complicated, more specific system to better address the set of particular training requirements.

This type of reductionist approach adopts a very basic engineering design principle, often applied to software development – "problem reduction" (Minsky, 1985), sometimes known as "divide and conquer". Very simply, the divide and conquer principle proposes that the way to write simple solutions to complex problems is to divide the complex problems into a number of simple problems. This can be accomplished by breaking a large program into several small programs that have well-defined interfaces. The efficacy of this principle has been demonstrated by modeling experts in a variety of domains, including software development (Lekkos, 1976; Porvin et al, 1991; Smith, 1998), control theory (Murray-Smith and Johansen, 1997; Narendra, Balakrishnan, and Ciliz, 1995), pattern-recognition (Hampshire and Waibel, 1992), behavior-based robotics (Brooks, 1986), and behaviors for computer generated forces (Henninger, et al, 2000). Typically, its value is derived in terms of reducing model complexity and/or improving model performance.

DEVELOPMENT AND MAINTENANCE COSTS

Introduced by Barry W. Boehm (1981) to estimate the number of man months it will take to develop a software product, CoCoMo (COnstructive COst MOdel) comes in three forms: basic, intermediate, and advanced, which range from a macroestimation model which treats the product as a whole, down to a microestimation model, which treats the product in detail. The simplest model, Basic CoCoMo, is useful for quick, early, and rough order of magnitude estimates. It computes software development effort and development time as a function of program size expressed in lines of code (LOC). Second, Intermediate CoCoMo estimates software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel, and project attributes. Lastly, Advanced CoCoMo incorporates all of the characteristics of the intermediate version but adds a supplementary assessment of the cost driver's impact on each step of the software engineering process.

CoCoMo may be applied to three classes of software projects: organic, semi-detached, and embedded. Organic projects are relatively small, simple software projects in which small teams with good application experience work to a set of less than rigid requirements. Semi-detached projects are intermediately complex software projects in which teams with mixed experience levels must meet a mix of rigid and less than rigid requirements. Embedded projects are software projects that must be developed within a set of tight hardware, software, and operational constraints.

Basic CoCoMo

From Pressman (1997), the Basic CoCoMo equations take the form:

$$E = a_b(KLOC)^{b_b} \quad (1)$$

$$D = c_b(E)^{d_b} \quad (2)$$

where

E is the effort applied in person-months,
D is the development time in chronological months, and
KLOC is the estimated number of delivered lines of code for the project (express in thousands).

The coefficients a_b, b_b, c_b and d_b are presented below in Table 1, according to class of software project.

Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Table 1. Basic CoCoMo Model Coefficients

Intuitively, because b_b is greater than 1, it is easy to recognize that as LOC increases, E increases at an exponential rate. In other words, increasing LOC by an order of magnitude (e.g., 10) does not translate into an equivalent increase in order of magnitude for E, as the latter increases exponentially. This concept and the log-linear form of CoCoMo is corroborated by a number of researchers (Briand et al, 1999; Kemerer, 1987) as well as by Brook's (1995) in "The Mythical Man Month", by demonstrating that a linear increase in program size does not translate into an equivalently linear increase in programming effort. Brooks likens the error in this thinking to assuming that the extrapolation of times for the hundred-yard dash shows that a man can run a mile in under three minutes. Clearly, even a simple software estimation model such as Basic Cocomo, demonstrates the relevance of this concept to software development costs.

Complexity

With additional data gained from a growing and increasingly matured software market, Boehm has improved on his Basic CoCoMo model, with CoCoMo II (Boehm et al., 2000). In contrast with Basic CocoMo, CoCoMo II is based on more contemporary programming efforts and is sensitive to more factors that affect development effort. So, whereas Basic CoCoMo simply considers KLOC, CoCoMo II also accounts for factors such as: complexity, data base size, reusability requirements, etc., as well as KLOC. As a means of demonstrating the economic advantages of the mission thread trainer concept, we developed a series of CoCoMo II models for a series of hypothetical software projects with. That is, we estimated the development effort (***E***) as a function of kilo-lines of code, where LOC ranged from 10000 to 1,000,000 for projects of increasing complexity. Based on our software development experience, we deemed these settings representative of systems developed in defense modeling and simulation community. The results generated from this CoCoMo II model configuration may be seen in the bar graph of Figure 2.

Clearly, the bar graph in Figure 2 demonstrates the tradeoff between size and complexity of program and cost of the development effort. This tradeoff is even more evident, when comparing the bar graph representing a CoCoMo II estimate, with the lower area plot representing an estimate based on linear extrapolation of 10 KLOC estimate. For example, according to CoCoMo II model, a 10 KLOC program requires 37 man-months to develop. Extrapolating this estimate linearly, suggests that a 1000KLOC program would require 3700 man-months. More correctly, however, the use of CoCoMo II model in the latter case reveals an estimate of over 13000 man-months required to complete the software development effort.

In the defense modeling and simulation community, one factor that contributes to development of large-scale, monolithic systems is the large number of requirements placed on these systems. For example, JSIMs has requirements to model tasks for both the detailed Special Forces and aggregate theatre command. Requirements such as these add a lot of complexity to the system; however, there is no mutual benefit to these models in being part of the same platform. Given the extra complexity required and the increased costs associated with the extra complexity, an alternative approach might be to separate those models into smaller (i.e., more "use-case" centric) systems. Adopting this policy would encourage a reduction in fidelity where it is not essential to the specific use-case. In terms of fidelity, for example, a Special Forces model would not require Corps level units and a theatre ballistic missile defense model would not require building interiors.

In addition to real costs associated with increased systems complexity, one must also consider the cost of lost opportunities. For example, spending a lot of time building/maintaining a large system preempts opportunities to build new and future systems that may contain new doctrine. That is, if a system takes 5 years to develop versus 1 year to develop, there may be a loss in expressing doctrine changes in a timely manner. Moreover, the cost of modifying large systems for new Tactics and Doctrine could actually exceed the cost of developing an entirely new system. Again, Brooks addresses this concept and warns against the propensity to maintain and enhance existing systems as opposed to simply developing new ones.

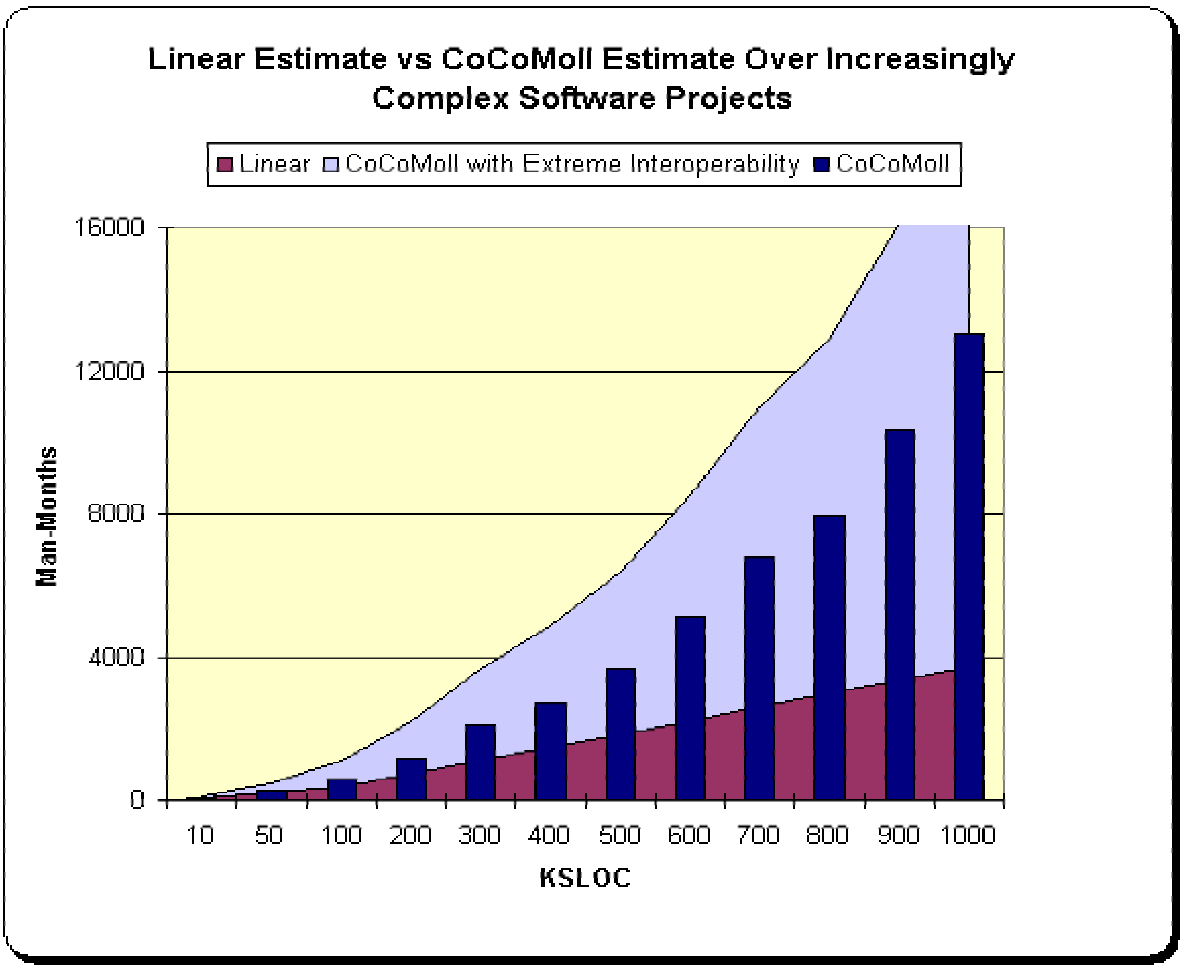

Figure 2. Software Development Costs Over Increasingly Complex Projects

Interoperability

One of the factors that can add to complexity of requirements and thus, increase the cost to develop simulations, is the requirement to build systems that are interoperable. Complex interoperability requirements, at times, may only add limited value for trainers with specific mission functions. For example, requirements such as making JSIMS concurrently interoperable with a virtual aircraft simulator and Theatre–level command and control systems can add a great deal of complexity to a system with respect to synchronization and resolution of models, platform constraints, fair fight issues, etc. Inevitably, this increase in complexity comes at a price. In the CoCoMo II model, for example, one of the factors representing added complexity is "Required Reuse". This factor accounts for the additional effort needed to construct components intended for reuse on the current or future projects (e.g., creating more generic design of software, more elaborate documentation, and more extensive testing to ensure components are ready for use in other applications). To estimate additional costs of interoperability requirements on software development, we increased the value of the reuse factor from nominal to extremely high. The results of this configuration of CoCoMo II model are presented in the upper area plot in Figure 2. As evidenced by comparison of this plot with the basic CoCoMo II plot, the additional costs of these types of requirements are not only high, but they too tend to increase the development costs at an increased rate.

With respect to the mission thread trainer concept, the plots in Figure 2 suggest that it may be economically prudent to develop a larger number

of systems that individually cover a subset of the requirements and collectively cover all of the requirements as opposed to developing a single, monolithic system that individually covers all of the requirements. The following two sections on Training Effectiveness and Availability consider additional benefits of adopting the Micro Trainer approach.

TRAINING EFFECTIVENESS

Clearly, a system tailor made for a particular application can be made more suitable for that application than a general model. This concept is evidenced in Chatham and Braddock's (2001) review of the Interactive Multi Sensor Analysis Trainer (IMAT). The Interactive Multi-Sensor Analysis Trainer (IMAT) is a PC-based tool that allows a sonar operator and a submarine's tacticians to visualize a very complicated acoustic situation and determine how best to use their sensors. According to Chatham and Braddock, IMAT is "an example of a single training device that changes the behavior of sonar operators so that they achieve an order-of-magnitude increase in submarine search area". Their review continues by saying that "an investment of a few million dollars in this training research and development (R&D) project has demonstrated performance enhancements that far more expensive programs have not achieved."

Schneider (2001) reports on proof of concept tests of COTS PC flight simulators in undergraduate pilot training. His results demonstrated the effectiveness of training with Micro Trainers by showing that students who used an air force-enhanced version of Microsoft Microflight Simulator achieved individual "Time-to-Maneuver-Item-File" sooner and more consistently than did students who did not train on Microflight. Thus, not only can Micro-Trainers be developed at lower costs, but reseach into their training effectiveness is showing that even very low-cost trainers can be effective training tools.

AVAILABILITY

Aside from Microsoft Flight Sim, the training and simulation community has embraced use of other enhanced COTS products for training (e.g., Delta Force, Operation Flashpoint, being enhanced and/or used by US Army and Marine Corps). One of the reasons for increased use of COTS systems such as these is because of advatanges with respect to availability. Use of these systems does not require a major scheduling effort or a large support staff. For example, the TacOPs simulation is available to run on PCs, whereas its large-scale counterpart, Corp Battle Simulation (CBS), only runs on VAX stations. Until recently, even the Closed Combat Tactics Trainer (CCTT)-SAF was only available on AIX workstations. It has recently been ported to Linux (Burch et al, 2000). Examples such as these demonstrate that, from the perspective of availability, micro-trainers have an obvious advantage over the larger, monolithic simulations.

DISCUSSION

Throughout this paper, we have focused on one key thread, the more diverse the requirements, the bigger the system. What flows out from this is the larger the system, the longer it takes to build, and the more expensive it is. As the system grows, the less likely it is to provide optimal benefit for any specific application.

As members of the M&S community, we have seen a large number of users advocate the use of off the shelf computer games as viable training solutions. What are they really saying? They want systems that are inexpensive, easy to use, available to them, compelling, and above all, they want them now. In doing so, they are accepting, sometimes unknowingly, the fact that the systems might not fully meet their needs. There is a trade space in which we can operate to deliver systems faster and at a lower cost; but in doing so, there has to be some relief from some of the requirements placed upon the systems.

There is an old saying: "Perfection is the enemy of good enough." Clearly, this applies to the case of micro-trainers. They are not designed to be "THE" solution, only part of the solution. The challenge is finding out what is "good enough". But, that is a topic of another paper.

REFERENCES

Boehm, B., *Software Engineering Economics*. Prentice-Hall, 1981.

Boehm, B. et al. (2000) *Software Cost Estimation with COCOMO II. Engineering Economics*. Prentice-Hall.

Briand, L., El Emam, K., and Wieczorek, I. (1999). Explaining the cost of European space and military projects. *Proceedings of the 21st International Conference on Software Engineering.* Los Angelos, CA.

Brooks, F. P. (1995). *The Mythical Man-Month.* Reading, MA: Addison Wesley.

Brooks, R.A. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, RA-2:14-23.

Burch, B., Hughley, P., McCulley, G., and Dietrich, D. (2000). Close Combat Tactical Trainer SAF on a PC. *Proceedings of 2000 Interservice/Industry Training, Simulation, Education Conference.*

Chatham, R., and Braddock, J. (2001). Training Superiority & Training Surprise: Final Report. Defense Science Board Task Force. Office of the Under Secretary of Defense for Acquisition, Technology, & Logistics. Washington, D.C.

Hampshire, J.B., and Waibel, A.H. (1992). The Meta-P,I Network: Building Distributed Representations for Robust Multisource Pattern Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol(14), no(7), pp. 751-769.

Henninger, A., Gonzalez, A., Georgiopoulos, M. (2000). Modeling Semi-Automated Forces with Neural Networks: Performance Improvement through a Modular Approach. *Proceedings of the 9th Computer Generated Forces and Behavioral Representation Conference.* Orlando, FL.

Jacobson, I, et al. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach.* Wokingham, England: Addison-Wesley.

Kemerer, C. (1987). An Empirical Validation of Software Cost Estimation. *Communictions of the ACM*. Vol(30), no(5), pp. 417-29.

Lekkos, A., and Peters, C. (1978). How to Develop Module Logic Using Pseudo-Code and Stepwise Refinement. *Proceedings of the 15th Design Automation Conference*. pp., 366-370.

McDonald, B., Weeks., J., Hughes, J. (2001). Development of Computer Generated Forces for Air Force Security Forces Distributed Mission Training. *Proceedings of 2001 Interservice/Industry Training, Simulation, Education Conference*. Orlando, FL.

Minsky, M. (1985). *The Society of Mind.* New York, NY: Simon and Schester.

DARPA/SOCCOM (2001). Scientists Helping America: Advanced Training Systems. <http://safe.sysplan.com/scihelpamerica/training.html>

Murray-Smith, R., and Johansen, T.A. (1997). *Multiple Model Approaches to Modeling and Control.* United Kingdom: Taylor and Francis.

Narendra, K. S., Balakrishnan, J., and Ciliz, K. (1995). Adaptation and Learning Using Multiple Models, Switching and Tuning. *IEEE Control Systems Magazine*. June, pp. 37-51.

Porvin, S., Reynolds, R., and Maletic, J. (1999). An Empirical Study of the Use of Problem Reduction as a Paradigm for Problem Solving in Software Engineering. *Proceedings of the 19th annual conference on Computer Science.* pp. 618-629.

Roland, E., et al., The History Of The Joint Theater Level Simulation (JTLS). <www.rolands.com/Pdf/JTLS_History.pdf>

Schneider, D., Greene, R., Leve, K., and Jeffery, D. (2001). Microflight Simulator. *Proceedings of 2001 Interservice/Industry Training, Simulation, Education Conference.* Orlando, FL.

Smith, R., Parrish, A., and Hale, J. (1998). Cost Estimation for Component Based Software Development. *Proceedings of the 36th Annual ACM Southeast Regional Conference.* pp 323-325.