

**NSF Workshop on Modeling and Simulation For
Design of Large Software-Intensive Systems:**

**Toward a New Paradigm Recognizing Networked, Distributed
and Diffuse-Control Software**

**Tucson, Arizona
December 3-4, 2003**

Executive Summary

The exponential growth in computational and networking hardware capacity and capability is pushing the desire for ever-larger software systems to drive business, science, and military systems. Further, such network-centric or web-based systems are increasingly functionally or geographically distributed, operate in real-time, with real-world interaction, and have numerous and diffuse sources of control rather than under a single monolithic controller as in the past. Unfortunately, the science of system design has lagged behind to guide the development of such software-intensive systems. The NSF-sponsored “Workshop on Modeling and Simulation for Design of Large Software-Intensive Systems” was organized in the latter part of 2003 to explore issues associated with design of such systems. The objective was to formulate recommendations for consideration of the National Science Foundation and other parties with interests in fostering the development of a scientific design approach to large software-intensive systems.

Approximately thirty researchers, representing a variety of interests in software systems and modeling and simulation, came together for two days of intense discussion facilitated by the GroupSystems electronic collaborative environment. As a prelude to formulating recommendations, participants brain-stormed on various issues that impinge on software development. These issues include: the need to educate at an early age for software skill development; the nature of abstraction, complexity and uncertainty; the roles of formal methods and dynamic systems in software-intensive development; and the larger contexts in which software systems are developed including those of management and business. Participants’ views varied considerably on the importance of such issues, some considering some or all of them to be peripheral to the intense and gritty nature of actual program writing, while others emphasized the need to account for the influence of these contexts in the formulation of any new science of system design.

Having set the stage with general exchanges of views, the participants formulated several challenges relating to the development a new software design paradigm, the education of the next generation of software developers, achievement of reusable software, integration of user interactions, and the transfer of software research products to the market place or science laboratory. Such challenges, both technical and non-technical, would have to be addressed by a new science of software-intensive system design in one way or another, to be both scientifically and socially successful.

Finally, in their last GroupSystems-facilitated session, participants formulated and voted on a rank ordering of several recommendations. The highest ranked recommendation was to develop a new software design paradigm that recognizes that uncertainty and emergent, unanticipated behaviors are likely to be forever present in the software-intensive systems of the future due to their ultra-large, networked, distributed, and diffuse-control natures. Further, such a paradigm should recognize and provide appropriate concepts and tools to 1) deal with the inherent asynchronous nature of such next generation software systems, 2) achieve truly reusable large grain software and simulation components, and 3)

incorporate representations of user interactions with the systems to improve understanding of their behaviors. Beyond such technical considerations, recommendations also dealt with cultural and business-related influences. Such recommendations were to introduce modeling, simulation and programming skill sets earlier and more pervasively into the educational process, develop an approach to speed the transfer of software/models developed in research to the end user, and enable managers and decision makers to champion the adoption of new software approaches when needed.

TABLE OF CONTENTS

<i>Executive Summary</i>	2
Background	6
<i>Objectives</i>	6
<i>Activities</i>	6
<i>Sponsorship</i>	6
<i>Organization</i>	6
<i>Workshop Concept</i>	7
Schedule of Activities	10
Issues	11
Challenges	12
<i>Challenge: New Software Paradigm</i>	12
<i>Challenge: Asynchronous Networked Software</i>	13
<i>Challenge: M&S and Software in Early Education</i>	13
<i>Challenge: Reusability</i>	14
<i>Challenge: User Interactions</i>	14
<i>Challenge: Transfer Research to the Market Place</i>	15
Recommendations	16
Conclusions	17
Bibliography	17
Appendices	20
Appendix 1 Record of 1st Day Session	20
1. <i>Educate at the kindergarten level-abstraction, modeling, software design</i>	20
2. <i>Forming abstractions is a basic tool for coping with complexity, is it possible to capture complexity, credibility, and uncertainty?</i>	32
3. <i>Design to address asynchronous character of software</i>	41
4. <i>Specification models to describe components and their interactions in a way that explicitly prescribes their abstract roles in a system</i>	47
5. <i>Infusing formal methods-perhaps lightweight-into widespread software development practice</i>	51
6. <i>Uniform framework for security, performance, and robustness needs of large-scale software</i>	57
7. <i>Abstracting formal frameworks to allow managers to have the right level of understanding</i>	61
8. <i>Dynamic systems frameworks for designing real-time applications with distributed components</i>	67

9. Overcoming compartmentalization to develop joint programs in computer science and business departments	69
10. Coping with a software industry in flux--changing skill mixes, user expectations, labor displacements.....	74
11. Bridging gap between software design science and domain scientist users	77
12. How much control can we place on software complexity before we limit possibly advantageous outcomes? Is Linux possible under a design methodology or is it a design methodology?	81
13. Making simulation THE way of addressing predictability in software intensive system design.....	85
Appendix 2 Record of 2nd Day Session.....	89
1. Challenge: Develop new software design paradigm that recognizes the inherent asynchronous nature of next generation software systems/applications operating in large scale, networked, distributed software.	89
2. Challenge: Introduce M&S skill sets (+/- abstraction, +/-modeling, +/-simulation, +/-programming, +/-design) earlier and more pervasively into our educational process.....	97
3. Achieve truly reusable large grain software/M&S Components	102
4. Challenge: Develop new software design paradigm that recognizes the forever-present uncertainty/emergent behaviors in ultra-large, networked, distributed, diffuse-control systems.....	107
5. Challenge: How do we integrate the user interactions with the software/hardware systems into M&S?	111
6. Goal: Develop an approach to speed the transfer of software/models developed in research to the user whether market place or science lab.....	112
7. Goal: Enable managers/decision makers to understand enough about the software development process to understand when adoption of new approaches is needed.....	114
Appendix 3 Record of Vote on Recommendations	117

Background

Objectives

The purpose of the workshop was to explore new directions for a science of design for large software-intensive systems facilitated by modeling and simulation (M&S). To do this, researchers in the theory, concepts and methodologies of M&S met with counterparts in software development to consider how the disparate elements that exist today can be integrated and further developed into a robust of science of large systems design.

Activities

A two-day workshop, by invitation only, was held in Tucson, Arizona on Dec 3-4 in the neighborhood of the University of Arizona. The workshop format, with approximately 30 participants, provided a cordial environment that is conducive to free and open exchange of software and system design approaches and examples of large software-intensive design issues to challenge these approaches. The workshop was centered on three, inter-related, themes:

- Specific design issues characteristic of large software-intensive systems
- Current approaches to design of such systems
- Emerging approaches: toward a science of software-intensive system design

Sponsorship

The National Science Foundation Advanced Networking Infrastructure Research Program ([NSF/ANIR](#))

Organization

The workshop was organized by:

- [The Society for Modeling and Simulation International](#)
- [The Arizona Center for Integrative Modeling and Simulation](#)
- [High Performance Distributed Computing Lab](#)
- [Secure Network Systems Design Laboratory](#)
- [Computer Engineering Research Lab](#)

Bernard Zeigler : Professor, Electrical & Computer Engineering, University of Arizona, co-Director of ACIMS .

Salim Hariri : Associate Professor, Electrical & Computer Engineering, University of Arizona, Director of HPDC Lab .

Hessam Sarjoughian : Assistant Professor, Computer Science and Engineering, Arizona State University. co-Director of ACIMS .

Sumit Ghosh , Hatrick Professor , Stevens Institute, Director of SENDLAB

Kevin McNeill, Research Associate Professor, Electrical & Computer Engineering, University of Arizona. Co-director of CERL

Gabriel Wainer , Assistant Professor, Systems and Computer Engineering, Carlton University

Phillip Hammonds, NGIT Team Technical Lead, JITC

Workshop Concept

Moore's law of exponentially expanding computational and networking infrastructure is fueling a trend toward ever-larger software structures to drive business, science, and military systems. Software-intensive systems such as future military systems-of-systems seek to employ software to implement highly integrated and capable command, control and intelligence functionalities. Unfortunately, the science of system design has lagged behind to guide the development of such software-intensive systems. Techniques that work for small software systems fail markedly when the scale is increased by one million fold.

Symptomatic of the lack of well founded design principles is the abundance of examples of projects that cost a lot but fail to meet their objectives. Many issues arise in the design of such large, highly decentralized, collections of interacting parts. The increased connectivity and capability create new complexity that is difficult to control and dynamics that are difficult to predict. Computer-based modeling and simulation (M&S) methodology is required to address these issues since the scale is well beyond what analytical tools alone can handle and there is limited ability to do controlled experiments. Large software-intensive systems demand new M&S approaches for understanding the dynamic behaviors of very large inter-connected networks with very few loci of control and many interacting components.

As detailed in recent DOD-sponsored reports, M&S, when properly performed through various stages of the software-intensive system design lifecycle, can provide effective assistance in formulating the system's capabilities, predicting and comparing the

cost/benefit ratios of its various alternative architectures, and evaluating its achieved functional effectiveness. However, to date, the development of simulation systems has become highly costly, yielding unwieldy, unreliable, and non-reusable artifacts. Reuse of off-the-shelf packages is extremely limited, in part, because their experimentation specifications, models and simulations, and are too tightly coupled. This lack of transparency and modularity makes it extremely unlikely that one will be able to match one's currently needed functionality with that of the off-the-shelf package in each of the three dimensions.

The underlying cause that impedes the development of reusable software is the absence from widespread use of a theory and formalism for modeling and simulation that among other things, supports separation of experimentation, models, and simulators. Such a theory would suggest the directions of a science of design in which M&S plays the indispensable supporting role that is required for large software-intensive systems.

The goal of the proposed workshop is to explore directions for a science of M&S-based design for large software-intensive systems. To do this researchers in the theory and formalisms of M&S will be brought together with researchers in software development concepts and methodologies. The discussion will center on integrating technical, infrastructure, and pragmatic elements into coherent basis for a science of large software-intensive systems.

Among the **technical** elements to be considered for their contribution to a science of design are:

- *Spiral development* , a normative, flexible, risk-driven process model that is used to guide multiple stakeholders through concurrent engineering of software-intensive systems. Model-Based (system) Architecting and Software Engineering (MBASE) is a recent extension of the spiral model with the goals of tailoring a project's balance of discipline and flexibility via risk considerations.
- *Formal methods* including the possibilities of "lightweight" variants that allow for inclusion of informal elements trading rigor for expressibility. Also, for large systems, a single, complete specification may not be possible and it may be more productive to compose many partial but appropriately abstracted specifications to allow some analysis of properties.
- *Architectural principles* that provide uniform structures with known properties to organize the complexity of large systems. Architectural styles, design patterns and Unified Modeling Language constructs provide instances of such principles

Among the **infrastructure** elements are

- *Simulation and Execution Infrastructure* provide the means by which software-intensive systems are designed and executed. Executable specifications and model continuity concepts are allowing greater linkage between phases of development with great potential reduced cycle time and increased quality.

- *Modeling of Dynamic Systems* - large software-intensive systems multiply the dynamic system complexities encountered by today's real-time control and embedded systems. Dynamic systems modeling formalisms enable specifying both system structure and the environments in which can be tested.
- *Knowledge, Intelligence and Logic* are the essential ingredients of emerging large software-intensive systems. Future system developments such as reliable and trusted Cyberinfrastructure, critical infrastructure protection, global financial processing and military systems-of-systems all will intrinsically incorporate artificial intelligence, knowledge based approaches and classical and model logic-based data manipulation schemes.

Among the **pragmatic** elements are

- *Education* is the basis by which any theory of design that emerges will be promulgated to form the greatly-improved software practice of the future.
- *Research* is needed to foster examination of basic software practices and principles, to create new integration frameworks, and to test these out in scientific ways.
- *Practice* is the source of issues to be considered in theory development and the ultimate test of success for new theoretical frameworks.

Invited participants with expertise in these elements will share their knowledge and insights with the group. The discussion will then center on integrating technical, infrastructure, and pragmatic elements into coherent basis for a science of design of large software-intensive systems.

Schedule of Activities

Wednesday, December 3	
Morning	Introductions by Jerzy Rozenblit, Interim Head of ECE, Bernard Zeigler, PI, and Taeib Znati, Program Manager of ANIR, NSF
	Themes: the Workshop Organizers presented brief overviews of, and perspectives on, the central themes and the technical, infrastructure and pragmatic elements of a theory of large software design.
Afternoon	First GroupSystems Session: all participants were transported to The University of Arizona campus to take part in a collaborative environment for facilitating anonymous discussion of ideas and brainstorming.
	Returning to the hotel, participants self selected into several breakout sessions to continue considering in more depth the ideas that have emerged in the Group Systems session.
Evening	Organizers met to formulate points of discussion and potential recommendations emerging from the GroupSystems' transcript. Participants were free to network and prepare for the next day's activities.

Thursday, December 4	
Morning	Presentations: In a plenary session, participants responded to the previous day's brainstorms by presenting aspects of their own work they felt may illuminate issues that arose and support one or other research direction. Presentations were kept short to enable as many to present as possible.
Afternoon	Second GroupSystems Session: All participants returned to the collaborative environment to help formulate and vote on recommendations for research directions toward a science of large software system design.
	Plenary wrap-up session: open discussion on recommendations and further thoughts about how to stimulate and support increased research in the science of large software design.

Issues

In the first day, GroupSystems was used to gather input on 13 major issues related to the design of software-intensive systems. Group leaders initiated the statement of several issues prior to the GroupSystems session. Later, the initial set was augmented by participants during the course of the collaborative systems session.

The issues discussed were:

1. Educate at the kindergarten level-abstraction, modeling, software design
2. Forming abstractions is a basic tool for coping with complexity, is it possible to capture complexity, credibility, and uncertainty?
3. Design to address asynchronous character of software
4. Specification models to describe components and their interactions in a way that explicitly prescribes their abstract roles in a system
5. Infusing formal methods-perhaps lightweight-into widespread software development practice
6. Uniform framework for security, performance, and robustness needs of large-scale software
7. Abstracting formal frameworks to allow managers to have the right level of understanding
8. Dynamic systems frameworks for designing real-time applications with distributed components
9. Overcoming compartmentalization to develop joint programs in computer science and business departments
10. Coping with a software industry in flux--changing skill mixes, user expectations, labor displacements
11. Bridging gap between software design science and domain scientist users
12. How much control can we place on software complexity before we limit possibly advantageous outcomes? Is Linux possible under a design methodology or is it a design methodology?
13. Making simulation THE way of addressing predictability in software intensive system design
14. Other Issues

For each issue, participants were asked to consider the following

- What is the challenge and why is it a challenge?
- What does it matter?
- What are the barriers?
- Why is progress possible?
- Who will be involved in the solution?
- How can we judge success? How much will it cost?
- What are some waypoints?
- Timeline and payoffs?
- Other issues?

Participants' views varied considerably on the importance of such issues, some considering some or all of them to be peripheral to the intense and gritty nature of actual program writing, while others emphasized the need to account for the influence of these contexts in the formulation of any new science of system design.

The transcript of each discussion is included in the Appendix 1.

Challenges

In the second day, the initial set of issues was clustered into a set of on six major challenges related to the design of software-intensive systems. These challenges, both technical and non-technical, would have to be addressed by a new science of software-intensive system design in one way or another, to be both scientifically and socially successful.

GroupSystems was used to gather input on these challenges:

- How do we develop a new software design paradigm for ultra-large, networked, distributed, diffuse-control systems?
- How do we introduce M&S and Software in Early Education?
- How do we achieve truly reusable large grain software/M&S Components?
- How do we integrate the user interactions with the software/hardware systems into M&S?
- How do we develop an approach to speed the transfer of software/models developed in research to the user whether market place or science lab?

Challenge: New Software Paradigm

Develop new software design paradigm that recognizes the forever-present uncertainty/emergent behaviors in ultra-large, networked, distributed, diffuse-control systems.

- Uncertainty must be an integral part of the design
- Evolutionary approach that emphasizes exploratory of design space using simulation - observation/analysis – experimental work
- Combination of symptoms might lead to diagnosis, prediction, etc.
- How can we come up with a simulation of a software design without building it

Approach

- How to learn from natural science research to better our system design approach
- Organic system design
- Foundations, theory, implementation
- Self-organized, evolvable, social aspect
- Can we use how the Internet designed/evolved as a paradigm?
- Use M& S to evaluate better our system designs, predict behavior, etc.
- Develop Ontology of Building Blocks

Challenge: Asynchronous Networked Software

Develop new software design paradigm that recognizes the inherent asynchronous nature of next generation software systems/applications operating in large scale, networked, distributed software.

The current state of software development is characterized by the following limitations:

- it takes too long to test commercial products
- Software Engineering does not sufficiently exploit modeling and simulation
- very few systems are built from a compositional methodology
- software systems are mostly evolutionary
- often claims are made with emphatic assertion, with no empirical analysis to support them

In the future, we would like to see

- Automated Evolving Adaptive Systems
- software that is distributed and ubiquitous
- software intensive systems that are autonomic, with large-grain, off-the-shelf components
- software that is continuously working and gathering information for self maintenance and self-improvement

How we get from the current state to the future state:

- integrate simulation into the development lifecycle.
- make composability a reality
- evaluate products and services through M&S
- process improvement becomes model evaluation and improvement.

Challenge: M&S and Software in Early Education

Introduce M&S skill sets (+/- abstraction, +/-modeling, +/-simulation, +/-programming, +/-design) earlier and more pervasively into the educational process.

How to make this introduction?

- develop a national business case for M&S and Software in early education
 - demonstrate the economics and value to stakeholders
 - use and extend the existing infrastructure
- Focus on different levels
 - K-6 and high-school
 - Experimentation/discovery and problem solving
- Identify and develop basic concepts, suite of problems, and teaching materials that focuses on complexity and scale
 - Develop Magnet schools focusing on M&S
 - Motivate students and teachers to be interested

Challenge: Reusability

Achieve truly reusable large grain software/M&S Components.

How to make this happen?

- Replicate the existence of hardware components (memory, CPU, bus,...). What are possible software equivalents?
- Facilitate human comprehension and communication about components to enable inter-operability across corporate or military stove-pipes/perspective
- Develop standards at higher levels that focus on the models/abstractions underlying the software rather than the software itself

Challenge: User Interactions

How do we integrate the user interactions with the software/hardware systems into modeling and simulation of such systems?

Examples:

- when a user browsing a website finds it busy, he/she stops the process and tries again; if a million or more users try to do the same (usually they do), the network and the web server will be flooded with unanticipated traffic.
- In crisis scenarios, users behave in an unexpected manner that frequently crashes the system. Although we have enough capacity; the system failed because we did not take into consideration the user interactions with the system in these conditions.

Challenge: Transfer Research to the Market Place

Develop an approach to speed the transfer of software/models developed in research to the user whether market place or science lab. Understand the business of software development and how it drives the industry

- Observe, research, model, and simulate business dynamics –evolution of firms
- Account for the uniqueness of IT – high profit margins, etc.
- Develop methods to build business case for transitioning from one phase to the next

The transcript of the GroupSystems session is presented in Appendix 2.

Recommendations

After the discussions described above, the participants used GroupSystems to formulate seven recommendations intended for the consideration of the National Science Foundation and other parties with interests in fostering the development of a scientific design approach to large software-intensive systems. GroupSystems anonymous voting facility was employed to allow participants to rank order the following recommendations (from highest to lowest rating):

1. Develop new software design paradigm that recognizes the forever-present uncertainty/emergent behaviors in ultra-large, networked, distributed, diffuse-control systems.
2. Develop new software design paradigm that recognizes the inherent asynchronous nature of next generation software systems/applications operating in large scale, networked, distributed software.
3. Achieve truly reusable large grain software/M&S Components,
4. Introduce M&S skill sets (+/- abstraction, +/-modeling,+/-simulation, +/-programming, +/-design) earlier and more pervasively into our educational process.
5. Develop ways to integrate the user interactions with the software/hardware systems into M&S.
6. Develop an approach to speed the transfer of software/models developed in research to the user whether market place or science lab.
7. Enable managers/decision makers to understand enough about the software development process to understand when adoption of new approaches is needed.

Details of the vote are presented in Appendix 3.

Conclusions

It is noteworthy that recommendations included both technical and non-technical considerations. Certainly, the highest ranked recommendation was technical in nature and was to develop new software design paradigm that recognizes that uncertainty and emergent, unanticipated behaviors are likely to be forever present in the software-intensive systems of the future due to their ultra-large, networked, distributed, and diffuse-control natures. Further, such a paradigm should recognize and provide appropriate concepts and tools to 1) deal with the inherent asynchronous nature of such next generation software systems, 2) achieve truly reusable large grain software and simulation components, and 3) incorporate representations of user interactions with the systems to improve understanding of their behaviors.

Nevertheless, other recommendations were non-technical and dealt with the cultural and business-related context in which a new paradigm of design for software-intensive systems would have to function. To be successfully adopted, this external environment would have to be managed appropriately. Thus there were recommendations to introduce modeling, simulation and programming skill sets earlier and more pervasively into the educational process, develop an approach to speed the transfer of software/models developed in research to the end user, and enable managers and decision makers to champion the adoption of new software approaches when needed.

Bibliography

Barry Boehm and Wilfred J. Hansen, "The Spiral Model as a Tool for Evolutionary Acquisition," CrossTalk May 2001

Barry Boehm and Daniel Port, "Balancing Discipline and Flexibility with the Spiral Model and MBASE," CrossTalk December 2001

Barry Boehm, Dan Port, LiGuo Huang, and Winsor Brown, "Using the Spiral Model and MBASE to Generate New Acquisition Process Models: SAIV, CAIV, and SCQAIV," CrossTalk January 2002

Bass, L., Clements, P., Kazman, R. 2003. Software Architecture in Practice , (2nd edition), Addison-Wesley

Davis, P., and J. Bigelow. 2003. *Motivated Metamodel. Synthesis of Cause-Effect Reasoning and Statistical Metamodeling* . RAND Corporation, US.

Dagstuhl Seminar, 2002. Grand Challenges for Modeling and Simulation, Dagstuhl Report , R. M. Fujimoto, D. Lunceford, E. Page, and A. Uhrmacher (eds).

Dahmann, J.S., F. Kuhl, and R. Weatherly, *Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture For Simulation* . Simulation, 1998. 71(6): p. 378-387.

Frydman C., M. Le Goc, L. Torres and N. Giambiasi, "Knowledge-Based diagnosis in Sached using DEVS models", Special Issue of Transaction of Society for Modeling and Simulation International (SCS) on Recent Advances in DEVS Methodology, Tag Gon Kim Ed., Vol. 18, N°3, 2001, pp147-158.

NRC Committee, "Technology for the United States Navy and Marine Corps, 2000-2035 Becoming a 21st Century Force: Volume 9: Modeling and Simulation," (1997), National Academy Press.

NRC Committee, "Modeling and Simulation in Manufacturing and Defense Acquisition: Pathways to Success" National Academy Press. (2002).

Ören, T.I. 2002. Future of Modelling and Simulation: Some Development Areas. Proceedings of the 2002 Summer Computer Simulation Conference, pp. 3-8.

Overstreet, C. M., R. E. Nance, and O. Balci. 2002. Issues in Enhancing Model Reuse. *International Conference on Grand Challenges for Modeling and Simulation* , Jan. 27-31, San Antonio, Texas, USA.

Page, E., and J. Opper. 1999. Observation on the Complexity of Composable Simulation. In *Proceedings of the 1999 Winter Simulation Conference* , 553-560.

Penix, A. M., and J. M. Alexander. 1999. Efficient Specification-Based Component Retrieval. *Automated Software Engineering: An International Journal* 6 (2), 139-170.

Sarijoughian, Hessam S. and Francois E. Cellier (Editors), *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies* , Springer-Verlag, NY. 2001.

Vangheluwe, H. L., J. De Lara, and P. J. Mosterman. 2002. An Introduction to Multi-Paradigm Modelling and Simulation. In *Proceedings of the 2002 AI Simulation and Planning in High Autonomy Systems* . F. Barros, and N. Giambiasi (eds), 9-20. Lisbon, Portugal.

Vaughn, Rayford and George Vinu, "Application of Lightweight Formal Methods in Requirement Engineering", CrossTalk 2003

Zaremski, A. M., and J. M. Wing. 1997. Specification Matching of Software Components. *ACM Transactions On Software Engineering and Methodology* 6 (4), 333-369.

Zeigler, B. P. 1984. *Multifaceted Modelling and Discrete Event Simulation* . Academic Press Inc., London.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic Systems. 2nd Ed.* Academic Press. Davis.

Appendices

Appendix 1 Record of 1st Day Session

1. Educate at the kindergarten level-abstraction, modeling, software design

I agree to the teaching at the kindergarten level for level-abstraction and modeling. I especially the modeling for problem solving. Although I do not agree with the necessity of teaching software design at that level. Although we may all be engineers, there are many that are not. Besides what an awful boring world we'd live in if we were all engineers. {#341}

Certainly you don't want to teach software design or modeling at the Kindergarten level - but early in secondary education these concepts should be introduced. Probably at about age 10 to 12. At this age building in a "comfort factor" is important. {#343}

It's not clear to me how this is possible, or even necessary. We are able to teach fundamental mathematical concepts at an early age because everyone believes that math is necessary for everyone. Everyone needs to understand making change, balancing a checkbook, calculating price per unit, etc. However, the model and software design issues are certainly not a fundamental skill needed by all. The only hope is to teach analytical thinking at an early age, which is a prerequisite for good software design. {#351}

abstraction and modeling are best left till a child understands why it's necessary. say around 12 or more. {#374}

The focus should be on relating educational material with the real world so they don't feel education is irrelevant. KG may be too early to teach modeling but we certainly need to emphasize mathematics, reasoning, and basic problem solving skills. {#429}

As was mentioned in one of the morning presentations, this is the SIM generation. A lot of kids already have some experience with M&S. {#822}

We should be careful not to jump too quick. The kids can put models together but all they are doing is clicking and pointing, limited very much by what the designer's had planned a priori. There may not be any originality. What we should emphasize is their ability to understand the interfaces and carry out activities in a meaningful, original manner. {#873}

#822: well, I think that it's time for this generation to go out to the backyard again... I want the next generation to be a generation of kids playing with their friends in the block, not creating the block in their computer. The brain has 2 hemispheres, and kindergarten is the time to develop both of them. High school is the place. {#874}

It is fascinating to see how everyone assume this requires kids to use computers. It is actually about problem solving. Modeling requires a medium in which to build the model. Abstraction requires seeing the wood for the trees. Nothing to do with our boring world of engineering, thankfully. {#1086}

Perhaps the phrase "software design" in the title of this thread has something to do with the assumption that a computer is involved :) {#1089}

1.1 What is the challenge and why is this a challenge?

How do you teach abstraction which has been considered a mature way of thinking to youngsters? {#317}

large scale systems should include the different notions of complexity, {#318}

When are children cognitively ready to deal with abstraction concepts? Can kindergarten students really deal with abstraction and modeling? {#332}

Kids have enough problems why inflict this on them? {#334}

I'm not sure this is a new challenge. Seems like yet another attempt at new math. {#335}

Educating educators is an enormous challenge. {#336}

Kindergarten education has culturally evolved over hundreds of years - for instance, the 3 Rs have existed for long and the new challenge is to develop another idiom that captures the concept of abstraction education. {#337}

Make sure that abstraction and modeling as problem, solving techniques are as well supported in the educational process as mathematics and other "traditional" techniques. {#338}

lack of reference problem set that is understandable by kids {#339}

Making toys is one thing, but to figure out what needs to be taught is another {#342}

the challenge is the "test-based" mentality of K-12 educators and the disconnect between K-12 and 13-16. There is a second challenge in teaching pedagogy. You need an active learning environment. {#344}

Kindergarten children are taught to use both their bodies and their minds to work together, e.g. working puzzles, how do we help them develop abstract models that they can still use their limbs to complete? {#349}

First, do we agree that educating modeling, and software design is necessary from the kindergarten level? {#352}

Classification of Knowledge levels is inherently difficult since it is by definition evolutionary. {#353}

Children should learn to test their ideas in organized trials, then appreciate their answers are wrong and how to correct them (without feeling bad about themselves). {#354}

The challenge is that educators will never sign up to this. It's hard enough for elementary school teachers to teach "required" material. {#368}

I don't think "software" design is necessary at kindergarten level. Building with legos and puzzles is a good design exercise. {#371}

Teaching the steps to stop critically think through a problem by building a model would be a huge asset to young children. The need is to create an environment of deep problem solving. {#376}

Why is kindergarten an appropriate level? What is the appropriate age to begin introducing these concepts? {#378}

High School educators in general have a difficult time with abstraction, themselves. Middle and elementary school educators will struggle with the abstraction issue. Identifying appropriate pedagogies for abstraction at the various levels will be a worthwhile challenge. {#392}

It is my HUMBLE opinion that the notion of "teaching at the Kindergarten" level is not about teaching at that level at all, but it is fundamentally MORE about reforming our educational system to accommodate entry of NEW thoughts on giving rise to educational practices which MAKE students MORE aware about complexity issues in general {#416}

There is a relation between modeling and Piaget notion of group. Such a notion arrives at kids around #12 years. The notion of symbolic variable comes to kid around 14 years. Learning 'theory of sets' too young can be under productive: learn basis first, next learn for abstract representation. {#445}

Kindergarten kids are not of an age to understand abstraction. {#446}

Teaching about abstraction can be enforced through story telling and asking for what is the key idea or developing a story that fits a theme. In fact some of the preschool and kindergarten learning software packages do try to teach this. I think just like mathematics, computer software is increasing getting in preschool and kindergarten education. {#463}

The organizational challenges of implementing innovative ideas in large scale software design include understanding the problem from several perspectives and being able to communicate those different perspectives well. Designers need to be agile in their ability to abstract to different audiences. {#464}

To have a modeling or design environment that allows kids to play with models or software components just like they do as they play with logos. {#469}

Concur with #416 whole-heartedly {#702}

K-12 level may be a red herring. Another view is to build on other trends to achieve our goals, e.g. children's exposure to simulation games. {#1045}

1.2 What does it matter?

Problem solving skills are vital in all walks of life. {#340}

Students are unable to benefit fully from UG programs because they are unfamiliar with these approaches. {#345}

The doctors will tell you that most of development capability is formed in the first three years. {#346}

I think it's too early to start at Kindergarten. {#347}

Problem solving is important but the idea of abstraction at the kindergarten level doesn't seem appropriate. We're talking about kids that still have problems with tying their shoes and are learning the alphabet. {#357}

If we are able to teach young people more abstract ways to thinking and solving problems, they will be able to use these more abstract methods later in life and this will help them develop the cognitive reasoning to build complex systems in the future. {#372}

Early education matters because it builds comfort in the learned skill. I agree with 347 in that its too early at Kindergarten level - but better at 10-12 years of age. {#373}

Good software design skills are a result of both education and practice. While we can "teach" how to design a software system, only through years of experience can one become truly competent. {#393}

The issue is to unlock the inquisitiveness of the children: w\the world needs scientific literate citizens. {#394}

Finding basic principles for developing increasingly higher-level abstractions is necessary to manage model complexity. {#402}

M&S is part of the "Fluency in Information Technology" movement - skills, concepts, capabilities. Being better users of computing, and some will be designers. {#404}

Barrier one is the scientific and mathematical phobias inflicted on the teachers.

Barrier two is a culture that denigrates science {#414}

1.3 What are the barriers?

lack of a reference problem set that is understandable to kids {#333}

Our imagination {#348}

Educating educators and developing curriculums. Overcoming existing inertia in primary and secondary education. Attracting people to education who are capable of dealing with abstraction, much less teaching it. {#350}

Lack of appreciation by educators of the significance of these approaches and lack of understanding of how to teach them. {#355}

Applicability to what the kindergartner's world view would be one barrier {#389}

when teaching kids something, it is important to make sure that they learn something that can be easily used. When teaching coding, they can write simple programs and see what they do (for example, draw something on screen). It is a lot of fun. If you want to teach kids modeling, how do they use this knowledge? {#390}

Finding the teachers to teach these skills in a way that it changes behaviors and excites the students {#396}

Educators won't agree to this. {#397}

Children have more concrete cognitive skill needs than they do abstract cognitive skill needs initially. Children need to be taught to solve concrete problems initially so that they have a sense of accomplishment. {#400}

Class size and teacher education. Since the size of classes have been growing, the teachers generally don't have the time to cater to these ideas. {#430}

It's hard to "scale down" M&S and other computing ideas to the essence of the exercises. Also teachers need to be trained to deal with complex technology exercises {#441}

Not easy to evaluate interdisciplinary research in an incremental fashion. {#453}

Standardized testing is a huge barrier. There is no longer an opportunity to spend class time developing a skill that is crucial, but not for the next test. {#454}

I have to be concerned with the software design objective - when I think how software design techniques have changed over the past 30 years or so, I wonder if trying to teach this at an early age might be too specific. Basic concepts - axioms perhaps could be taught - but design concepts will change as the student grows. {#515}

1.4 Why is progress possible?

I'm not sure that it is. How is this different than New Math from the 1970s and 1980s? Does every person really possess the intellectual gifts for this kind of work? {#370}

The significance of alternative approaches to problem solving is emerging as a major shortcoming in our education system, creating a climate of acceptance. {#377}

It isn't. {#398}

Not sure that it is possible at the K level. Maybe more likely at grades 3-5. {#401}

The NSF has funded a number of high-school and middle-school education programs. My sister a Physics professor (Meera Chandrasekhar) has been developing some for now several years, and I hear good reports. So, it may be possible in other fields as well. {#403}

I conduct workshops on computational science and engineering to faculty in MSIs and many of the attendees are outside the technical areas. Many of these folks teach pre-service teachers. given the right tools and the right teaching

methods, these folks are thrilled with what they can do. We get great feedback from them on how they change their classes and how the new teachers are making progress in their schools. {#457}

Simulation games for kids are part of the culture, teaching them economic, physical, and computing models without making a big deal. It's already happening! {#476}

It is happening, but most kids are not understanding what it is. It is not being presented in any way that is understandable to the kids. {#518}

The idea that this will all work from a K level is out of the question, but I believe it will work better at a junior high or high school level. This is the level that you could take kids who are interested and allow them to participate. Any lower levels would really not be feasible. {#495}

Progress demands investment at multiple levels and by different organizations.

Basis research is being shortchanged with emphasis on short-term returns based on economically driven policies. {#512}

Let's keep in mind that even the teachers may be very excited about learning and teaching students about these new topics, the kids are just too young and they don't have the same mind set as adults and it's definitely not good to force on them. Intuitively, I strongly oppose such a recommendation because I am a strong believer that we need to "simplify" our kid's life and go back to teach them "basic principals" rather than trying to have them always in the mode of catching up with technologies and "prepare" for the future - that by itself is causing a lot of social issues later on in their life and affecting our society greatly. {#590}

I completely agree with #590. Forcing our kids to understand concepts of modeling and abstraction is too much. I would put the effort in latter stages (starting with high school) {#613}

1.5 Who will be involved in the solution?

school teachers, departments on pedagogic, government, M & S community {#369}

Teachers, educationalists, computer scientists, government. {#395}

teachers, education researchers, mathematicians, computer scientists (theorists probably) {#407}

This can be investigated by both early childhood education as well as special education experts within colleges of education. {#410}

Researchers will be involved - because this involved breaking new ground in teaching methods. We need experiments, data collection, tests, result analyses etc. {#412}

First of all, the Legislators must see that what they're doing isn't working. School boards must let new teachers with new ideas try them out. Reward excellence, not orthodoxy. Last and certainly not least, you need to convince parents that science and mathematics are important and to demand good pedagogy in those areas. {#474}

Need to combine both early childhood/special ed educators with computer scientist/engineers in the process. {#478}

Theoretically all stakeholders need to participate. Pragmatically, end-users, policy makers, and visionaries need to play key roles. {#532}

1.6 How can we judge success?

Anything except a standardized test would be okay! {#391}

The only way that I know of to assess this kind of learning is individual assessment. {#399}

Increasing numbers of students able to design and develop software systems using model based approaches. {#405}

One measure might be a return to greater enrolment in CS {#406}

ability to think abstractly can be judged using some sort of IQ test, I presume {#409}

Cost per bug-free LOC. {#411}

By tracking the progress of the students over a 20 year period - how may go into what profession, how successful they are, {#442}

Another standardized test. Maybe we can flunk them out of kindergarten if they don't pass :)

Seriously, it would take years of study of a cohort group to see if they make it through to a CS program. There has been some study of STEM cohorts, maybe the same could be done here. (I guess I'm talking method rather than metric). {#450}

Programming has always been considered a product of abstract thought. If we are able to teach young children to develop more complex systems than they have been able to do in the past and if these meet the goals/requirements that

have been set, then this could be used as a measure. We can make it more concrete by using a school environment, as saying the percentage of students in each grade from K to 12, how many are programming complex system implementations in each grade. {#460}

Right now, the rate of entry into technical fields is abysmal. There is a great book called "Talking bout Leaving" which documents why students leave the technical fields. Revive that study at K-12 now and in 5 years. Another obvious test is the college population and retention/graduation rates. {#500}

More American kids coming into science and engineering {#513}

I agree with the 513 post there. Outside of the government, we are bleeding technical jobs as students go for easier business degrees, to become people persons. There is no incentive for them to follow their technical desires. The American engineer may go the way of the steel worker and the clothing manufacturer. {#542}

There may not ever be the silver bullet answer. However, we can develop matrices to map needs to solutions at different levels of abstractions. {#557}

1.7 How much will it cost?

Probably, not much. Education programs are not hugely expensive, relative to other research programs {#415}

The initial cost will be high, but once the current system has been updated it will not emerge as a separate cost. {#417}

I think it will be expensive to find teachers that are qualified to teach this. {#447}

Compared to the defense budget? Nearly nothing. It would be much more expensive in time and need for culture changes...again we have the new math example as a reason why this would be hard. {#462}

An experiment at a University environment, could possibly be done in the \$100-500K range on from 200-1000 students. This would translate to \$50 per student possibly. {#468}

Actually, the cost should not be the issue. It's a change in attitude. It's not free because new text and props are needed to do active learning. {#519}

Not as much as you'd think. {#520}

A cost model needs to be defined to try to answer this question. The cost can be measured as a percentage of value to the society. {#592}

MEASURING cost as a function of VALUE to society will be IMPOSSIBLE in my opinion -- at least in as far as beginning this task is concerned. This refers back to the 'visible weed' comment just before lunch. {#634}

1.8 Waypoints?

The toys markets should see a new breed of toys.
Higher scores recorded in the harder sciences {#443}

Definition of a curriculum from 3 to 21 years encompassing these approaches. {#444}

In 5 years --- that's enough to get a couple of rounds of new teachers out --- there should be a verifiable shift in attitude by teachers and students. It would take at least 20 years to clear the current ideas out and replace them with a "new orthodoxy" {#526}

a "killer app" modeling app that researchers like us, parents, teachers, and kids enjoy and respect {#528}

It will take 5 years to convince ed schools (whom I consider to be part of the problem not part of the solution) to get out of the kill-and-drill, test-em-to-death mentality. Pay off is the more scientifically literate citizen. {#534}

Yes, there the horrid disconnect between what K-12 does and what 13-16 demands. We have seen a steady decline at Clemson in the student quality, even in engineering entries. We have to quit pretending that K-12 is doing the job, back off and prepare them at the university level. {#569}

Passing the standardized testing and passing into to modeling and sim is a huge leap. {#647}

1.9 Timeline and payoffs?

10 years at least.
Payoffs would be intangibles at first. {#448}

This is a multi-year multi-jurisdictional issue with potentially significant benefits. However making significant educational changes is daunting. {#458}

two years to develop a curriculum.

further two years to implement teacher training program.

implementation complete when first school children enter university with this background. {#461}

What would be the real benefit? Would it really be easier to just de-program (retrain) students when they enter college? {#494}

5 years, then 15 years with more American students attending engineering school {#538}

There may not be a finite length of time given the moving target we are dealing with. {#606}

1.10 Other issues?

Aren't we being a bit arrogant in assuming that our anecdotal evidence with our own children and/or college students is more valuable than many decades of educational and cognitive research? {#408}

May be. But, I have not heard this discussion before. {#451}

Agree w/ #408 above...I was surprised at one person's comment this morning that all entering college freshmen are competent programmers and are already "set in their ways". This is definitely not the case. I teach at a major and well respected university, (in ECE, not CS), and only 1 of 20 or 1 of 30 have any programming skill at all. And I find it easy to convince them that "my way is better", and they should re-learn any skills they might already have. {#452}

I agree with 408 and 452, just because of one overly arrogant comment this morning about his son being an amazing programmer at an early age is insane. I entered college with NO programming experience. Sure I had read a few SAMS books, but that is not formal training. The majority of people who are entering college are not experienced programmers, most have not even seen the languages. They are taught a few short semesters and are ushered on. {#611}

The other issue we haven't considered is how technology will change over the time that we are changing the method of teaching. Will these skills still be needed in the same way they are today, will there be a market for such students, will paradigms change? {#459}

I think it is ok to start from college. Many kids, even if they know programming, once they met a real programming challenge (for example, build a high multithreaded concurrent system), they are quick to realize the advantage of system modeling. {#467}

This is not just a program aimed at software developers, these techniques are important for many other application areas. Business modeling is a good example. {#472}

Do young children ever play games (video or other) that require abstract knowledge? Many seem to be task oriented that require localized thinking. However, it is obvious that some abstract knowledge is gained by young children without us having to teach it. {#475}

We should stop worrying about programming and teach students more design and validation. {#570}

I think it is counter productive to take away childhood -- we need some evidence that teaching basic sciences "artificially" can be superior to how nature (society) is capable of doing. Do we want a 5 year old to be the equivalent of 40 year old? {#641}

2. Forming abstractions is a basic tool for coping with complexity, is it possible to capture complexity, credibility, and uncertainty?

Developing models of levels of abstraction are necessary. If abstraction is a tool, what are the levels of abstraction that are important in system design? {#496}

Complexity is a slippery concept, since it is not always apparent how complex a system is until it has to scale up. E-mail was simple when restricted to single, unconnected computers. {#497}

Forming models is the basis of all thinking and all understanding. There is a simple psychological limit: the human can deal with about 7\pm 2 concepts at a time. Capturing complexity is inherently bounded. History shows that credibility must be earned through consensus --- Thomas Kuhn is right on this, I think. Uncertainty and certainty are complementary ideas but it's hard to capture what you don't know. {#610}

The basic concept of abstraction and formulating a universal solution for it I believe is not attainable -- there exists "unbounded knowledge". {#657}

There is a way to cope with complexity: to separate knowledge that is different in nature. The basis is 3 set of knowledge (i.e. 3 models): the knowledge about the structure, the knowledge about the functions and the knowledge about the behavior of the system under consideration. Apparent complexity often comes from the merging of these 3 kinds of knowledge. Real complexity comes from the "co-habitation" of these kind of knowledge: how to maintain the coherence of the 3 models? {#980}

2.1 What is the challenge and why is this a challenge?

The real challenge is determining when an abstraction is appropriate and structuring data so that numerous abstractions can be generated and integrated. House plans are an excellent example of this. There is no single abstraction for the house, but a collection of domain-specific abstractions that can be selected and integrated by the contractors. {#456}

Obviously it is possible to encapsulate complexity. The universe does and does it in real time. There is no proof that it is infinitely complex but given that we are a part of it, it might as well be. So the question is: what examples, i.e. software examples, do we have that exhibit the artifact of capturing complexity in a big way? {#465}

The real trick is to be able to get the information you need when you need it and not worry about the rest. The hard part of this is determining what you need. {#470}

choosing the "right" level of abstraction for the task at hand and identifying a suitable formalism and organization level of the system {#480}

Major problem is how do you merge different abstractions? Xerox PARC's aspect oriented programming touches on this. Do not know if there is any major result though {#517}

getting a person who can comprehend the entire extent of the problem, analyze it to a sufficient degree to be able to create even a half-good abstraction, which captures not just the deterministic effects, but also the uncertain effects is a tall order. But with enough research this can and is done (see review articles in physics, math etc). But credibility? One would have to be an expert of equally high standing to be able to critique such an undertaking. Takes time and effort.

Our challenge thus is NOT to capture complexity but to develop an abstraction-based and hierarchy-based approach which is extensible. This approach can then be used by domain experts to capture and express complexity and uncertainty (forget credibility - that will be established by repeated use of the model). Since the approach is hierarchical, there is hope that the various models might be synthesized into a composite model of a really complex system where a model might be of some use. Various component-based models for dealing with software complexity come to mind. {#523}

By capturing uncertainty, I think you established a scope of credibility. I don't think repeated use of model establish any credibility. {#1056}

Abstraction can be used bottom up or top down. Pre-design modeling is an example of top down, while re-engineering is bottom up. the challenge is to use abstractions which preserve essential information and support analysis. Ideally we need abstractions which can be extended when necessary - core model and annotated model. {#531}

I think that it is also important to be able to not only form an abstraction but also represent it in a way that would be comprehensible to more than just a handful of people. So I would change this to be something like capture and representation {#536}

Asynchronous thinking has traditionally been limited to "real time design", which has not been taught or fostered in most of our schools. We should start to integrated real time concepts in the earlier stages of our training of IT professionals, so that Real Time is no just a field for the elite specialist. As all of our systems will exhibit these characteristics, they need to be taught to a larger group of practitioners than we have done in the past. {#588}

The challenge comes from the increase of the quantity of information about our world (physical, emotional and conceptual). The information increase exponentially. As a result, we will be unable to acquire and analyze such a quantity of information. We then need conceptual tools for understanding (i.e. decoding) our environments in order to act correctly. The challenge is then: how to access the information that I need to act? In other words, how to design the systems that will help me in order to access the data? This is a challenge because such systems must embed large quantities of knowledge: how to put knowledge into a computer? {#591}

Conventional software design methodologies are not well suited to deal with constant evolution of software intensive systems. The major characteristic of large complex systems is that they constantly evolve. Maintaining such systems requires making predictions about possible changes or alternative design organizations. New abstraction methods are necessary to facilitate effective and tractable predictions of the implications of changes in complex systems. The real challenge is to help designers to analyze "possible" alternative systems. Simulation modeling can provide a new perspective. New advanced abstraction mechanisms that facilitate simultaneous exploratory analysis of various alternatives would be an interesting idea to pursue. {#595}

Complexity isn't always inherent but it's also a stage of knowledge. NSF and DoD fund big messy projects that maybe look more complex in order to generate more funding. Witness this morning's multiple box and arrow diagrams. When is the complexity really needed and could some complexity be removed earlier? {#625}

The challenge is that we should stop kidding ourselves. There is no silver bullet, dammit. We understand the psychology of problem solving so little that the challenge is to first understand how we actually do solve (i.e., model). It is a challenge because we keep looking for more silver bullets. {#639}

Excess detail gets in the way of finding and removing more difficult problems. It costs to produce detail then costs more to remove excess detail. {#640}

Abstractions that cannot be captured and communicated pictorially do not spread well over the masses. Especially when they need cryptic language (mathematics) or conceptual arguments (philosophy) they become the tools of a few. Not the tools of the masses. With Software so prolific, and the demand significant, the bricks and mortar of the industry - namely the code has become the repository of the abstraction as well as the implementation tools. We are talking of separation of these two functions, and need to change the form factor. Only then can we scale it down and out. This is the major challenge. {#660}

It is impossible to capture complexity, but it is possible to manage complexities. Formal abstractions reduce the complexity in order to produce information's about a system according to a point of view. But complexity is the fact that a great number of point of view can be put on a unique system. So the challenge is our ability to produce many models of a system and maintaining the coherence of the set of model over time. {#661}

Can somebody provide a simple example to illustrate how abstractions can be modeled? {#677}

Uncertainty about a system means absence of knowledge about the system for solving a particular problem (i.e. reaching a goal). The problem is not uncertainty but risks. If the risk linked to the absence of knowledge is too big, we must achieve researches in order to obtain the knowledge. If the risk is small, the underlying uncertainty is not problematic. Consequently, robustness is the main response to uncertainty. {#688}

Uncertainty can occur in various forms. For instance, an engineer trying to maintain a system needs to make a decision from a set of alternatives. The implications of these decisions on the behavior of the system is an uncertainty for the designer. S/he needs to effectively explore alternatives to predict and resolve the uncertainty with respect to alternatives. {#745}

Constraining alternative techniques to describe abstractions. Without some means to limit the growth of knowledge, we run the risk of creating "poor abstractions" that could overtake "good abstractions". The corollary of this challenge is to differentiate between poor and good abstractions. {#753}

Development of reusable abstractions, reusable theories, etc., will help. {#774}

Can you give examples about good vs. poor abstractions?? - ref 753 {#775}

I think the evaluation of abstractions falls into the credibility issue. Capturing credibility is a key issue {#793}

Abstraction is fundamental to science. All our theories and models are abstractions. Their predictability and refutability helps differentiate poor from good abstractions. Though these are not the only differentiators. {#838}

How will we measure complexity? If we can measure complexity in one level of abstraction, will it have a higher/lower/same measure of complexity at another level of abstraction? In other words, I think we will change the level of complexity as we change the level of abstraction, but not necessarily in a fixed way, i.e. moving up the levels of abstraction, (getting more abstract)

does not necessarily lower the level of complexity (do we need measures of complexity for each level of abstraction?) {#938}

You only need concern yourself with your 'experimental frame', that is your context and perspective. If you use a formal method for describing your domain and complexity level, someone else who understands formal methods can abstract from your contribution. It is the documentation and common language that allows advancements to be made. For example, a chemist may understand something about nuclear physics, but it may not be part of her experimental frame, however she can read and understand the language of physics - math and can make use of ideas that are relevant. In the same way, a biologist who understands the language of chemistry can abstract to his domain relevant concepts from chemistry. {#970}

How do we measure complexity? {#997}

Uncertainty plays a role in our understanding of the software systems behavior. It also might play an important role in designing large scale software systems. If we envision components that shall interact with other components in a flexible manner they likely will be faced with uncertain and incomplete knowledge about their environment which might or might not be relevant to their functioning correctly. In the former case the components have to be equipped with the ability to deal with uncertain and incomplete knowledge. {#1004}

2.2 What does it matter?

Trends away from software engineering as programming to SE as problem understanding, design, analyze, validate. (No programming in this model). {#455}

Many problems that occur during system integration and deployment are detail issues that can be lost in the abstraction. {#466}

James Gosling's famous words "devils are in the details". {#521}

Our education has let us to think in concrete vice abstract concepts. This concretization, limits our imagination and our problem solving capability. For the most part, the majority of the population have traditionally have trouble in abstract thinking as is evidence from scores on mathematic tests. {#522}

One key problem is loss of information. Many systems have no clear underlying model of their requirements, for instance. Recovering such models is very tricky. {#535}

Abstraction is a key way that we communicate and reason about complex systems {#556}

It matters because formulating and validating abstractions IS science. Complexity of interaction is an information is in the way. {#651}

Although multiple aspects are not known to be sufficient to deal with complexity, it is at least necessary. {#762}

2.3 What are the barriers?

the extra effort, learning and understanding a formalism that supports abstraction, which should be sufficient flexible and sufficiently supportive at the same time. {#449}

Standard abstraction models. Computer science has not had its Newton yet. Specifically, we do not have a single standard collection of design abstractions that are universally accepted. I don't know of any way to get there except research and experimentation. {#493}

A barrier is in finding the right representation and training others in that representation. For instance, we already have mathematical representations that can model many things but they don't get used because not enough people are (or ever will be) trained and comfortable in using those representations {#584}

Most software engineers think operationally, not structurally. {#586}

Buzzwords {#650}

About #493: I don't think we'll ever have our Newton; the world he had to understand was overlyingly simple compared with the simplest complex models we study today. Software is so complex, and it involves so many different activities (recall the Social Science comments this morning) that there will never be a unique approach for solving the problem. {#655}

"We have met the enemy and they is us", I believe is a quote from Pogo. We try to hold ourselves to the mathematical model. Well, it took is 2,400 years to get to this spot (starting with Eudoxes, say) and there's a lot to go. The "axiomatic" method came about 600 years after Euclid and didn't get in place until Hilbert. Holding ourselves to an unreasonable standard is a killer. You cannot stop, you must go on --- but George Polya pointed out that the last step in solving a problem is to go back and ask "what did you learn?" We don't do that...'cause the journals won't publish. When Poincar'e published his paper on proving a theorem, he was roundly criticized by people much less his stature. {#684}

Our perception of complexity. What is meant? What are the components? {#704}

Teaching modeling/abstraction as a techniques and not as a science. We need to know principles and develop techniques instead of teaching "tools" and educating "technicians". As we create more artificial systems, the need for science will grow exponentially while the need for "technicians" will decline, perhaps linearly. {#788}

2.4 Why is progress possible?

We have no choice and the emergence of architectures and patterns as useful abstractions is encouraging wider adoption. {#594}

Forming abstractions is an inherently *disciplinary* process. We need to student the disciplinary approaches and recombine these. That was von Bertalanfy's vision and that of general systems. We have made progress in the area, but it's outside disciplinary lines and therefore ignorable. {#712}

Developing appropriate abstractions will be necessary for solving complexity. {#803}

Conceptual tool like systemics are now emerging (refer to Edgar Morin philosophy of complexity for an example), more and more powerful computer and networks are available, and learning software tools are currently being under development (refer to data mining for example). And the fact of the complexity is now largely recognized over the world. {#852}

2.5 Who will be involved in the solution?

K-teachers, software game developers, M&S researchers, funders (NSF, Dept Ed, Gates) {#498}

At this point, primarily scientists and engineers. The engineers need to be present to make certain that our design abstractions are useful and predictive in design activities. {#516}

Engineers, mathematicians, computer scientists and cognitive scientists. Especially the cognitive folks. It would be nice to find a good representation that a large number of people would respond to (and no, I don't mean UML). {#597}

Cognitive psychology has a major contribution to make. We are in danger of trying to re-invent solutions where a lot of useful work has already been done. An interdisciplinary approach is essential. {#604}

EWD, call home {#659}

The research community has to be involved. Is it more important that you proved a theorem or that you can tell people your method and let them go on further. I say the latter. {#719}

Traditional people but also non-traditional people such as sociologists and naturalists. {#820}

Every one is involved in the solution, but the main actors are the engineers that will build the software. Is it time to change our philosophy? From "prove before to build" to "build first, prove next". {#886}

Please, No! There is enough 'build first, fix later' in software. {#985}

It is certainly possible to CAPTURE complexity and uncertainty . But, novel ideas are now needed.. not a re-hashing of old PowerPoint slides, and certainly not a regurgitation of 'antiquated thinking from the usual suspects! {#1010}

2.6 How can we judge success?

When we are able to reason well about those quality attributes (credibility, uncertainty, and complexity) then we will know we have been successful {#605}

Wider use of easily exchanged models to document and design software. {#609}

Complexity is considered a show-stopper in projects. When a design gets too complex, there are indicators, other work stops and the design if corrected until the uncontrolled complexity is removed or the design abandoned. {#676}

We will know we're getting somewhere when we have a viable approach to comparing two "methodologies." Actually, I think you will get somewhere when the word "methodology" is removed from all discourse. {#726}

I think the point made by 676 is what should happen and clearly does happen in many cases. However, I have seen a fair number of DOD projects in which the overly-complex design was not simplified with predictably poor results. {#831}

2.7 How much will it cost?

Its hard to attach cost to this. Its a challenge, yes, to form the abstractions that address these quality attributes but I can't even begin to think about what the cost will be. {#618}

The cost will be in training the existing workforce, but people used to claim that OO programming would fail to displace Fortran and C for the same reason. {#619}

Hm...It's attitudinal. {#732}

2.8 Waypoints?

Not clear. This is a generational timeframe. What we should start seeing is a literature that contains --- dare I say it --- philosophical evaluation. Scientist have, for 200 years, steadfastly refused to examine themselves critically: that's why we get cold fusion. {#743}

2.9 Timeline and payoffs?

This is a generation long issue. The payoff is a population capable of critically evaluating science and technology. {#751}

2.10 Other issues?

The inclusion of non-technical aspects of systems within a common framework is essential in the medium term. Abstractions must be inclusive of non-technical (socio-technical, business etc.) to be really useful. {#635}

3. Design to address asynchronous character of software

Every software project is different. It's hopeless to have a "one size fits all" method for design of software. In some cases, execution speed is of primary importance which of course affects how the system is designed. Other cases require extreme memory efficiency, which again requires a different approach. Maintainability and readability are always important, and in some cases are the primary design factor, requiring yet another approach. {#471}

Asynchronous design concepts have been traditionally addressed by "real time design" concepts. {#530}

Not everything will be asynchronous. We need to define which applications we are talking about here. QoS is defined differently in different domains - response time, utilization, reliability etc. Different analysis methods result from different concerns. {#658}

Effective analysis and reasoning about asynchronous software may not be computationally feasible. Simulation can be used, yet the confidence one can instill in complex asynchronous software by using simulation will have its limitations. {#722}

Us old codgers remember when asynchronous stuff was legit...and then we discovered digital. I've got a couple texts with lots of asynchronous FSA stuff in them. We could start by going back to the 50s. One way to approach this is the realization that all computation is based on a partial ordering of the operations: remember the language Occam by Hoare? {#771}

Looking over the above comments, it seems that some of us are falling back and not seeing the objective of this effort is to address very complex (large and distributed) systems. Some comments such as "Not everything is asynchronous", but in this large, distributed environment, it may be. {#963}

3.1 What is the challenge and why is this a challenge?

Do we have a succinct definition for asynchronous software? {#473}

Can someone please provide a good definition of "asynchronous character of software"? {#479}

can someone comment on why process algebra can not be used to model check async software? {#492}

This is important. I am not sure about design though, does any methodology address asynchrony or time at all? Looking at parallel computing in the context of special relativity is an attractive concept and one that should be

pursued. There is way too much hand waving in this (design methodology) field, way too much hand waving. {#524}

Asynchronous implies that there is no central event that synchronizes execution. In synchronous hardware, calculations are tied to a clock that, in the simple case, causes all calculations to be posted simultaneously. Distributed software doesn't share this characteristic. {#525}

Process algebras are great, but they are mathematical and not engineering abstractions. Specifically, can you scale process algebras to really large systems? The math is certainly there, but the engineering application really isn't. {#533}

reply to #533. umm... what are engineering abstractions? process algebra uses abstractions like processes, events and channels. Further more, model checkers like SPIN from Bell Labs uses concepts similar to C. I agree they have scalability problem though {#589}

Question to #525: can you explain why distributed software doesn't share this characteristics? {#631}

The reason hardware is typically synchronous is because it makes design more tractable, and easier to test and validate.

There are ways of synchronizing events in distributed systems that do not share a common clock - are there fundamental problems with them? {#644}

The formalisms are out there to reason about asynchronous systems. The problem is that they don't get used and won't be adopted unless we do something to make them more accessible to the average developer. So, I think the real challenge is "Increasing accessibility of formalisms to support more widespread use" {#666}

We can analyze asynchronous systems using model checking/temporal/modal logics, simulate them, perform schedulability analysis on them etc. We need to focus on supporting designers in framing the right questions, so that the correct sort of analysis can be performed. {#674}

The asynchronous behavior is a natural phenomenon in any system. The problem is not

in the asynchronous behavior it is the methodology that defines the range of acceptable/unacceptable behavior. Through that the software components can behave and act asynchronously according to that framework. {#754}

The challenge is the human mind ... it's really a poor instrument for dealing with asynchronous/ stochastic events. In the case, "wetware" is the rate limiting step. Example, we teach the kids to do synchronous computation all

their undergrad and then expect them to deal with the parallel distributed world without retraining...nonsense. {#794}

reply to 589. I guess the best way to explain my thinking is that we all start out in engineering learning the same math - 3 semesters of calculus, 1 semester diff eq and sometimes a bit of discrete math. Then we spend the rest of our engineering education learning these things I think of as engineering abstractions. They are bodies of information that bring the mathematics to a specific problem domain. I see process algebras as still in the mathematical stage. There are great tools, but do they relate to engineering problems rather than mathematics? {#821}

3.2 What does it matter?

Even software systems that are built of a couple of components that are asynchronously interacting with each other can show behavior that is difficult to predict. For designing large scale systems with multiple concurrently interacting components we need suitable software engineering methods that will likely integrate M&S technologies. {#654}

Software system can show complex behavior that is difficult to predict based on the asynchronous interaction of a couple of concurrent processes. If we are talking about large scale systems we need methods that support their design. {#656}

Designing simple applications with distributed asynchronous components is a really hard task. It takes years of research and development to build a new distributed operating system; moreover if we need to include real-time constraints. Better design techniques with capabilities for automated tasks are required. {#699}

The issue is Quality of Service or non-functional requirements. We confuse performance with real time responsiveness, for instance. Asynchronous systems tend to be used for critical systems, which impose various constraints that we need to meet. {#700}

Natural phenomena are inherently asynchronous. To model the real world, we must be able to see these asynchronous interactions as related, not stochastically independent. {#804}

3.3 What are the barriers?

Ask the hardware people. Asynchronous hardware designs are more power efficient and smaller than synchronous designs. Yet, hardware designers are wed to the concept of a clock. I would speculate that they don't like asynchrony because they don't have good models either. Plus, virtually all design environments deal only with synchronous designs. {#571}

Asynchronous systems are harder to reason about without machine assistance, since the problems are much trickier to be sure of. Usable tools (usable by the developer, not a mathematician) are needed. {#711}

My experience in teaching M&S, especially discrete event, is that the students lack the linguistic ability to describe what's going on. Someone once noted that distributed processing would never catch on because the sequential world has just one programming model but every machine architecture imposes a different model. Gotta break the linguistic barrier. {#817}

If this helps, there are a number of examples (life forms) in Nature that seemingly resort to multi-processing. The squid has tens of thousands of chromatophors on its skin that are individually and simultaneously turned on and off by the very large brain so it can blend into the surroundings or undertake a dazzling show of light, etc. {#841}

Very difficult to visualize (i.e. its very abstract) {#969}

3.4 Why is progress possible?

The availability of modeling and simulation techniques will make progress possible. Besides it offers the most promise, scientifically. {#638}

Why? {#680}

M&S can certainly help, but it's not the only solution available; in many cases we can build analytical models that can be solved. If that the case, the use of simulation is unnecessary. But we need to address the cases of very large and complex systems with distributed component and asynchronous behavior, including different levels of abstraction and needs (security, fault tolerance, performance). Simulation can help in deciding how to improve the resulting software as a tool for validation of intermediate results. {#720}

If we don't progress, we either limit the potential for more complex asynchronous systems to be developed or we increase the chance that they will fail to meet their required levels of delivery. {#721}

There have been some attempts for formalize asynchronicity --- CCS, CSP, and Unity as example. What we have to do is quit shooting every attempt to model asynchrony because it's not perfect within one year... look again at the math experience: 2400 years and it's still not perfect and we can't agree on foundations there, either. {#830}

3.5 Who will be involved in the solution?

Currently the problem seems to be mostly one of applied research and tool development. So I guess it is applied researchers in universities and corporations. {#725}

What is needed is a serious attempt to *unify* what we know rather than to keep running off coming up with new vocabulary and gimmicks. The vast majority of researchers will not participate in this effort. But this is what would mark the beginning of "philosophy of computational science" as a new study with similar goals as the philosophy of mathematics. {#837}

DARPA {#974}

EE/CS departments (Both departments look at these asynchronous concepts differently and have different tool sets) {#982}

One reason that EE and CS look at things differently is that they are distinct disciplines. {#1021}

3.6 How can we judge success?

Maybe when Windows stops crashing? {#728}

Computation is categorical, so we will succeed if I can study a new "paradigm" and show how it's just part of the Chomsky hierarchy without all the machinations we now go through. {#845}

3.7 How much will it cost?

Basic research is relatively cheap. Applied research costs more, so I guess a few tens of millions. {#733}

This is a long term issue --- we probably have usable theories now if we'd actually use them. Besides, philosophers are cheap :-) {#850}

3.8 Waypoints?

Usable model checkers and simulators within CASE tools. {#735}

There's really two issues: algorithms with decidable properties and one's with undecidable properties. We're getting somewhere if we can deal with decidable properties on a routine basis. Routine means some program does it. The halting problem doesn't disappear. {#869}

3.9 Timeline and payoffs?

Prototypes in next three years.

deployed in industry in 5-8 years.

payoffs once deployed, in more reliable systems and shorter testing times. {#740}

5 years on the "routine" stuff. Payoff is speed and correctness. {#872}

3.10 Other issues?

We still need to think what the important questions are that we need answered about different categories of system. {#747}

4. Specification models to describe components and their interactions in a way that explicitly prescribes their abstract roles in a system

Agree this is a good goal. I believe that system specification has been an active research area for decades, with (IMHO) little to show. {#514}

We need specification models that can be easily understood by the customer and the developer. Models perhaps that can depict the mental image of the system under consideration (and its functionality) so that the mental models between developer and customer/user can be compared. {#541}

We need mechanisms for representing different aspects or facets of components and understand how they interact during design. Not every designer sees a component the same way. {#593}

Several challenges emerge:

- notation/language

- levels of abstraction that need to be specified

- relationships between the levels

- How do you know that you are finished? {#620}

This challenge has been out there for several years. The real question is how do we use the models to reason about crosscutting properties (e.g., aspects) and system level quality attributes such as security, performance, credibility, and the like? {#685}

Are specification models the same as spec methods, e.g. Z? Are spec models ways to write specs and models? are these specs used as models? Don't understand. {#708}

Is it possible to separate the specification from a low-level design adequately? My experience tells me that no specification system is happy with a high level spec and at a lower level, it become as design, almost an implementation.. What are the right boundaries? How do we not implement the system in a pseudo language in the process of specifying it ? {#757}

The challenge is: how transform a model into another? Software design can be seen as a model transformation process that go from the user requirements to the software behavior. Model transformation is similar to language translation. We have a lot of work in front of us ... in a short term, the challenge is how to forget technologies in order to focus on knowledge? {#758}

4.1 What is the challenge and why is this a challenge?

I am not sure this actually means something. So if a component model is specified (many are to a greater or lesser degree) then wouldn't the specification identify the component's role? {#568}

Not if it has more than one role. {#596}

The challenge is in directing more effort to the specification phase of development. Developers generally move past this phase too fast and correct later. This is rationalized by most with a belief that you can't get it right anyway - so move forward and adjust later in the development lifecycle. Generally this results in higher costs and more re-work. More research in better models for specification might be in order. {#599}

What exactly would the specification say ? One could keep it very small - in which case it'll be widely used in all sorts of ways - or make it so exhaustive that it's perfectly useless.

I'd say - create a spec that caters to a small community and let it evolve. If it has any merit to it, it'll collect various relevant ideas along the way. If it does amount to anything, one could do a halfway design update (iteration #2 !0 which places it on a more formal level. One shot specification and holding onto it won't accomplish much. {#637}

A specification needs to capture the assumptions and obligations of a component to facilitate modular reasoning. {#665}

Standards for specifying components require levels of abstraction, since we may have a specification which relates in a parameterized manner to several loosely equivalent components. When we move to more detail, aspects such as memory utilization vs. execution speed may differentiate these. {#769}

The consistency among different abstraction levels is crucial. That is, the commitments made at the previous level of abstraction should not be violated. {#836}

4.2 What does it matter?

It matters for the sake of cost of the system and usability. {#603}

Reuse is crucially dependent on solving this, among a few other factors. {#778}

Because designing is reasoning about specifications, not about the way a specification is implemented with a particular technology: the behavioral specification of a component is the stable feature of a component. {#781}

4.3 What are the barriers?

Lack of precise tools which are acceptable to the development AND customer community {#607}

Lack of experience in building with reusable components especially those developed elsewhere. {#783}

Lack of incentives within current business processes to reward development of reusable components for use outside the current project. {#792}

Users want technologies, not specifications! Because a specification that is not implemented has no value, but a technology, even poor, can have value added. A guarantee of implementability must then be provided. {#814}

4.4 Why is progress possible?

There are good research efforts ongoing now in this area - particularly in the area of empirical software engineering. {#617}

Other good efforts in assertion-based verification, equivalence checking (in hardware), static type checking. {#636}

Emergence of common notations, especially UML. {#797}

4.5 Who will be involved in the solution?

Software engineering researchers using techniques from other areas of social science, computer science, and perhaps psychology. {#632}

Software engineering educators and researchers. {#642}

The OMG has a successful track record in developing standards of this sort. {#802}

According to reliable sources within, OMG has strayed from its founding principles {#1019}

Industrial standards bodies that effectively require you to purchase your place at the table are not a good model. {#1062}

4.6 How can we judge success?

Emergence of a genuine market in reusable software (and probably software/hardware system on chip) components. {#808}

4.7 How much will it cost?

Depends on the commitment of corporations to make standards real. {#812}

4.8 Waypoints?

Formation of an OMG RTF. {#813}

4.9 Timeline and payoffs?

Payoffs are the cutting of costs in software development, without resorting to offshore sweatshop business models. {#819}

4.10 Other issues?

This really needs serious industrial muscle. The Banking and finance sectors have been working on common objects to support interoperability for years. Other sectors need to get on board. {#826}

5. Infusing formal methods-perhaps lightweight-into widespread software development practice

We have wonderful examples of where this has already occurred. Static type checking is widespread and is every bit a formal method. Compile generators that work from formal grammars are equally successful. Both are successful because the math does its job without bugging the user. {#615}

There is an increasing corpus of work demonstrating the successful insertion of formal methods-based efforts into system developments. {#663}

The emergence of XML as a means to define model interchange is a very significant help here. Anyone can read a model defined in UML using rational Rose into their model checker, assuming that they can generate the underlying labeled transition system. Hey, now there's a thought.... {#834}

One problem with all "formal design" processes is (IMHO) is that there seems no way to "debug" them. A design that seems perfectly reasonable on paper, and one that has been reviewed by a competent team, often will still have major flaws once the implementation begins. And since the design has been "approved and validated", the implementers usually just hack a seat of the pants solution on a case by case basis, rather than rethink and re-document part of the design. {#934}

Comment #934 generalizes to almost any development process. Need to allow for engineering judgment. We don't need to be slaves to formalism; use it where and how it is useful. {#967}

5.1 What is the challenge and why is this a challenge?

Formal methods has the potential for improving trustworthiness, reliability, etc. of systems. However, formal methods have had an extremely difficult time being adopted. {#491}

Formal methods tends to stay on a very abstract level, which make it insufficient as design proceeds to involve more and more details. {#585}

Why do we continue to require CS students to take 3-4 calculus classes and one discrete math class? Shouldn't it be the other way around? {#600}

There is no way anyone will understand the utility of this. The correct way to express these ideas is in the form of a computer language. {#612}

some formal methods are emerging to deal with low level details. I know there is work being done on using DSPIN to check for object reference, lifecycle, exception handling issues in java code. {#614}

scalability is a big problem for formal methods {#622}

Old formal methods were too hard to use, and therefore not used {#629}

how to model dynamic systems is difficult using formal methods. I do not know any good method to model the construction and destruction of threads. {#630}

Formal methods have not shown a ROI in many cases due to the high cost or difficulty to use {#633}

most commercial shrink wrapped software does not require formal methods. Think Microsoft {#646}

Let's get real and not think academically. In a pinch, and most projects are pinched, any type of methodology is thrown out the window for a quick fix. Everyone wants there to be a methodology, but there is none. People will continue to make bucks selling their "ideal" solution, but there remains a huge disconnect. {#675}

Microsoft is using lightweight formal methods on various parts of their code body. Also, if one wants to consider low level details, work at UT-Austin, Intel and AMD have been at such levels. In fact, there was one UT-Austin PhD which used FM to reason about object code and demonstrate various properties.

One point that is particularly important, is that FM is not an all-or-nothing technology. {#679}

Infusing formal methods should be focused on FM as one tool of many. To do so we need to reduce the time it takes to both learn and use them. The lightweight techniques are nice but even some of them need effort that developers might not want to undertake. {#717}

Why lightweight? Formal methods don't guarantee good designs although they may make more precise and validate designs. {#737}

One basic challenge here is to "define" what lightweight formalism means - we throw the term around like we have a common understanding, but we don't. While I agree/believe that less rigor in the formal approach might assist in its adoption in general - more research is needed to define very carefully how much rigor can be given up before we lose credibility with the formal approach. {#749}

The ROI issue continues to be an issue. Though, at least within the security space, there is the use of the Common Criteria (EAL4+) that recommends

some uses of formal methods. Various companies are looking at the higher levels for product differentiation. ROI is improving as costs decline. {#786}

I said this in the main thread, but it bears repeating here. Lightweight formal methods, if defined as formal methods that don't put more burden on the user, have been highly effective. Static type checking is an application of formal methods. So is generating parsers from formal grammars. The ROI argument can certainly be made for these technologies. I think we tend to use formal methods to describe only those things that we are still researching. (Not to imply that the type theorists are done!) {#843}

The challenge is threefold:

1. To find ways of generating formal models from rather informal software designs without the designer knowing it is happening.
2. To find ways of allowing designers to frame useful questions without knowing modal logics etc.
3. To give back answers to these questions that make sense to the person who asked them. {#846}

5.2 What does it matter?

Formal methods has the potential for improving trustworthiness, reliability, etc. of systems.

There are numerous application domains where improvements in the above are mandatory in the post 9/11 world. {#511}

Formal methods if properly used, could result in lower cost systems from a lifecycle perspective (we need to address lifecycle cost vice system acquisition cost) as the main criteria of success {#648}

Capability to think abstractly in a world of concrete designs. {#653}

Software engineering can't be engineering without some math foundations. Need to be able to reason about designs. {#744}

I don't know that we can say that formal methods result in lower cost systems - at least not without substantial empirical data. I believe that formalism probably improves safety and reliability - but the cost issue may be a different story. {#766}

I defer to Ty's quote from Lord Kelvin. A man with the absolute temperature scale named after him must have something to say. If we cannot be precise, we can say nothing. {#853}

5.3 What are the barriers?

Has historically been pitched in a manner not conducive to industry adoption. Complex technology inconsistent with normal practice. Need to better understand tech transfer models so as figure out how to present the positive attributes of the technology in a positive and adoptive role. {#527}

Education. Math is being pulled from CS curriculums rather than added. How many formal methods educators are there out there? {#623}

Must convince the students that this is relevant. {#627}

Education and training. {#742}

Too much calculus in undergrad computing curricula, too little discrete math. Get up to speed in sets and logic and formal methods aren't so hard. {#756}

UML currently is the 800 pound gorilla in design but there's no guarantee of good designs from using UML. Nor from using formal methods. {#761}

Separation between the formal method specification and what the user really sees; most people can learn how to write C code, a few of them know how to build a C compiler (using the standard formal notations to define syntax and semantics), and a few of them actually understand all of the theoretical details and are able to go forward in the definition of theory.

We need to separate:

- . Theory
- . Tool building based on theory
- . End user tools for the end users

Now: keep in mind that we spend lots of time to teach how to program in a programming language; this is a main issue here: any effort in making formal methods more usable implies spending a good amount of time in developing usable tools and spending the time in teaching them. {#767}

We need to remember that the formal method must be understandable to the end user/customer as well as the developer. While training the developer in its "correct" use is one barrier - an even larger one is making sure that it can be used as a communication tool between the developer and customer/end user. Software development is still a joint activity between the two communities. {#780}

There is still a perception that formalism means correctness proofs of code. I think this is still a barrier in its acceptance. {#796}

This stuff is *HARD* and don't really have mature enough understanding of how to use it. {#870}

5.4 Why is progress possible?

Much better understanding over the past 10 years on the engineering aspects of formal methods. Whether it be lightweight tools or better focused uses of the technology. Tools are better as well. However, still generally requires better than average personnel (on the most part). {#537}

helpful in understanding and verification of systems {#608}

Success stories with the use of formal methods (e.g., in specification V&V, identification of redundancy or omission in specs; etc.) {#815}

More computing power and memory.

More significant application pull, especially in life critical applications. {#877}

5.5 Who will be involved in the solution?

FM experts, high assurance technologies and application domain areas. Lessons indicate that a multidisciplinary team is more likely to adopt. {#555}

Federal CIO Council {#682}

DARPA {#683}

commercial software industry {#723}

DARPA is not attractive to students {#724}

Discrete math textbook authors and publishers. {#772}

Those able to bridge the two communities. {#878}

DARPA should be attractive to students [albeit graduate students maybe..] What is perhaps more important is that DARPA develop a critical eye to the future. It used to have good leadership back in the days when we were fighting to get our first satellite up! Today, it lacks both leadership and vision. Unless this changes and changes in a significant way, DARPA is not an ideal candidate. {#1036}

There's a capability within the European Framework 6 funding stream to fund networking efforts. Such projects fund the bringing together of stakeholders to discuss R&D directions and to aim for increased interoperability between

projects. Funding of such interdisciplinary efforts may be necessary here. {#1052}

5.6 How can we judge success?

Folks like Boehm are looking into this. Various commercial/government efforts have kept track of costs over phases. Should be able to measure improvements in the ilities and in costing. {#573}

When formal methods use equals that of software inspection {#776}

When we can reliably predict the reliability of realistic systems. {#881}

I would take issue a bit with #573. Boehm and Basili both are empirical researchers and both support more traditional methods (at least they have historically) - these being walkthroughs/inspections; English language specs; customer/developer teams working and talking together. I have not seen either or them talk about or write about formal methods being a pragmatic approach. {#1008}

With regards to #1008, as of last Spring there was a project starting at UCLA to assess FM. {#1057}

5.7 How much will it cost?

Tough question to answer because adoption of formal methods can range over a scale of capabilities. Some capabilities, if properly packaged, will be easy to adopt and likely will not be overly expensive. Other approaches, being more comprehensive and incorporated into tools could cost in the millions. {#621}

5.8 Waypoints?

Model checking as standard within CASE tools. {#883}

5.9 Timeline and payoffs?

Increased predictability of systems. Reduced costs. Comprehensive analysis. Will complement the results arising from a testing-based milieu. Traceability of requirements, etc. {#643}

5.10 Other issues?

In reality we have to accept that formal approaches only cut down on uncertainty. They will probably be most useful in focusing our testing plans on areas of maximal uncertainty. {#891}

6. Uniform framework for security, performance, and robustness needs of large-scale software

Cool idea; unfortunately these separate goals are often conflicting. Security almost always costs in performance. Perfectly "Secure" software would have to undergo so much V&V testing that the product would be obsolete before it's released. {#529}

We often use the term "robustness" in software. Can someone please provide a good definition? Usually it's something like "robust in presence of failures". Software failures? We are to design software systems that will work even in the presence of major coding errors? What about coding errors in the "error recovery" code? {#539}

robustness refers to reasonable behavior of the system for inputs not discussed in the specification of the behavior of the system. {#1087}

Validating large software intensive systems would be a daunting task. The challenge is to come up with systems that are designed to have in-built self-testing similar to self testing hardware. {#540}

Normally, software is written to operate correctly under normal circumstances. As such, this is a time consuming and challenging task. Robustness requires us to go a step ahead and ensure the correct operation or some systematic exception handling in the event of a deliberate attack or accidental failure. {#662}

Need to address software safety at the same time as security {#664}

well.. we can not even do a good job of analyzing the performance of a software system. How can you expect us to think about all 3 aspects? {#668}

Forget it. For us big-scale simulation people. we need mathematical and algorithmic robustness first, performance next and system robustness third. If that comes at the expense of security, so be it. We'll isolate the system. {#670}

Need to integrate into the Enterprise Architecture (EA) world. Note importance of EA in current federal government IT environments. {#678}

It would be hard to make a uniform framework for these attributes since many systems differ so greatly. However, if you stick to specific domains or families, then this may be more possible {#727}

The Army has a security procedure that all distributed applications have to go through, unless they are web based. Thus the thousands of web applications that are coming on line. They simply will not play with these rules. The basic developer is not trained to develop software and ensure of its protection. If there

is a separate team that does security, they are generally slammed with other issues. If they did find an issue they would lob it back to the programmer, who is too busy to comply. It is simply a lose-lose issue. {#741}

I agree with 727. I don't think you can have one security architecture for all systems. {#851}

Security, Performance, and Robustness needs are in the eye of the beholder. {#875}

Somewhere in here the concept of "trust" is struggling to emerge. We face a world where software will be coming from outside, as purchased COTS components, as a downloaded applet, as a roving agent etc. For this to work we have to establish trust. The others are surrogates for this. {#899}

This is an excellent research issue and there is work in this area today. You cannot just "isolate" the system - systems today aren't isolated and when networked there is no guarantee of isolation. Performance and security are generally at odds with one another - but current research in High Performance Computing security has shown that security can be enforced with acceptable performance degradations. Still - there are tradeoffs that must be made. {#936}

One way of trying to have these addressed within a uniform framework is to assign costs and risks to them. {#956}

6.1 What is the challenge and why is this a challenge?

No metrics to accurately characterize the system behavior and functionality. Also, they do not scale as we move from network level to application level hierarchy. {#626}

As was stated in the morning session, complexity is a huge problem. Greater complexity equals greater potential for security problems. Complexity usually means greater need for performance. Finally "robust" software is a lot easier to envision than to implement. {#628}

Conflicting requirements {#652}

Mobility and flexibility are at odds with security. Wireless security is a complete mess. {#667}

Wrong question. Need is for ways to analyze tradeoffs among these nonfunctional properties and against functional requirements. Trade-off not only in quality but in effort to produce the quality. {#799}

Evolutionary software development based on recursive and holographic reasoning in order to pass over contradictions. Separating knowledge and

function. Designing software able to acquire the knowledge (i.e. the model) that is required in order to solve the problem (develop one time, uses many times). {#946}

This is a challenge partly because we seldom engineer security in from the beginning. Take for example High Performance Computers today - they were engineered for speed which partially involved stripping out security protections. After a system demonstrates vulnerabilities - we try to patch. Breaking this paradigm is a challenge. {#950}

6.2 What does it matter?

If I can not characterize the behavior, configuration, and functionality, accurately, we will not be able to build systems that are predictable, secure, maintainable, evolvability, etc. {#649}

We are becoming exceptionally dependent on software infrastructure. Security and reliability are just too big to ignore. {#671}

Lack of a uniform framework ensures that there will be interoperability problems. Worse, there will be security problems. Robustness will be a unrealized goal. {#755}

Security, Performance, and Robustness needs 'are in the eye' of the beholder. THIS issue is the contributing element behind 755' s comment {#888}

6.3 What are the barriers?

We do not have scientific way to build such systems. Until we develop that, we will be designing software systems in ad-hoc approach, lead to unpredictable systems, not secure, etc. {#669}

Such {#673}

These issues interact with each other in ways that are difficult to predict. It's not sufficient to simply have models or frameworks for these facets. We must understand how they interact. {#709}

6.4 Why is progress possible?

BECAUSE we are better than APES, and are capable of discarding our orthodoxic beliefs with respect to how we approach a solution! We are not even close in this regard. Therefore, the sooner we get on THAT path, the better.. {#900}

6.5 Who will be involved in the solution?

Federal CIO Council {#705}

DARPA {#706}

DARPA ... yeah right!!! maybe back in the days when we were fighting to get our first satellite up! The statement that comes to mind which most adequately put the problem into focus is a line said by Sidney Portier in "Guess who's coming to Dinner", where Sidney's character expresses more than just his nature of general discontent on 'acceptable conventions.' I paraphrase here what Portier's character said to his father... "Not until your generation has laid down and died, will your dead weight be off of our backs... you have got to get off our backs!" Novel ideas are now needed.. not a re-hashing of old PowerPoint slides, and certainly not a regurgitation of 'antiquated thinking from the usual suspects! The same goes for Fed. CIO council! The so called 'Grey Beards' [wisdom-less] are not going to solve this problem. The problem requires a level of intellectual sophistication that has yet to surface. {#978}

The problem goes well beyond software, to incorporate the system as a whole (software, hardware, and the users). And hell, why stop at only these aspects of the system? An exceptional leadership organization will be needed, with highly effective management expertise and a clear understanding of how to bring about such a much needed transformation of the current environment. Should be a non-profit, but without the baggage of say MITRE or RAND. Of course, expertise across the spectrum of domains will also be needed. {#1060}

6.6 How can we judge success?

6.7 How much will it cost?

6.8 Waypoints?

6.9 Timeline and payoffs?

6.10 Other issues?

7. Abstracting formal frameworks to allow managers to have the right level of understanding

Not at all sure what this means. Does it mean that managers should be able to read formal specification documents and design documents? No way! I have dealt with dozens of "technical managers" in my career, and they are managers, not technicians. They understand the content of the requirements and design documents only at the very highest level, depending on their technical folks to understand and explain them. {#587}

This is critical! The ability to influence managers, administrators and legislators has to be a priority. We have found that success comes from getting these types to buy into formal methods even though they have a limited understanding of the formalism in question. It is possible, but requires much patience and the ability to abstract to various audiences at various levels. {#616}

Absolutely critical. The "everybody sees everything" DoD approach is going to be a huge mess. We should be saying "everybody sees the right things" {#681}

Managers are concerned with budgets, staffs, deliverables etc. Abstracting formal methods do not seem to have a direct relationship to PMs needs. {#714}

By formal frameworks, I assume you mean set-theoretic ideas and formal logic that ensure certain properties to the work that are trying to do.

Expose that to the technical person who is going to be affected by it. For heaven's sake don't confuse others with it. The technical person can communicate the guarantees that these frameworks provide as some of the characteristics/properties/guarantees of his work. But it should stop there.

E.g. do you go around telling your manager that your computational results (a) were generated by a C code ? (b) it was ANSI C ? and (3) that the results comply with IEEE floating point arithmetic ? {#718}

No, but even dumb managers want to have the ability to say that the projects they are in charge of are done in a way that would allow independent V&V. A formal approach gives the managers that piece of mind and engineer and technicians a framework that they have faith in as well. {#729}

I've heard this before. I don't think that managers care. The ROI is too low for it to pay off generally. So if I go to a manager and say "I have a technique that will require a lot of training and will give you a small amount of additional benefit over what you are currently doing" they'll laugh at me and kick me out of their building. I can see where it would be more necessary for critical systems but in general it's not that important. {#748}

What they care about is their bottom line and their credibility with their superiors. A formal approach can provide both if applied appropriately. {#868}

Surely we are talking about integrating worldviews to ensure consistency? It is very clear that technology departments in large organizations have problems in understanding business needs, while business units find it hard to understand technical issues. A largely overlapping means of expressing things is needed to alleviate this. {#937}

Yes, I think so. I think that language is mathematical in nature, but it need not look so imposing and symbolic. {#1000}

The challenge of knowledge reduction (abstraction) is to develop communication means (e.g., immersive visualization) to work with human cognition in accordance with the expected users (i.e., managers). {#1025}

7.1 What is the challenge and why is this a challenge?

Providing evidence to leaders that formal methods result in better algorithms and better software, both in terms of performance, scalability and reusability. {#645}

How do you relate the formal framework understanding to the manager's focus on schedule, budgets, personnel? {#871}

A challenge would also be to provide evidence that formal methods are well-established and stable and that their employees can be trained to comply to them. If the change is going to be substantial, the benefits need also be substantial. {#887}

The real challenge is to establish a language/notation/medium for communication that reduces misunderstanding among the different domains of stakeholders within organizations. These include technical, organizational, socio-technical and human resource interests, at least. I don't know how "formal" such a medium would be, but it would need clear and agreed semantics for all aspects involving an overlap of concerns. {#948}

One challenge is that management tends to panic early in a formal methods approach. Managers need to see progress toward an end delivery being made and formal methods does not show that progress early on. This isn't true in all development projects - but in most. Safety critical developments seem to accept this approach up and down the ranks - but not, for example in business system development. {#988}

Agreed. However, the business community has fears too. We just have to relate a formal approach to address overcoming those particular fears. {#1035}

Higher level concepts should be accessible more easily (less time and effort) to more people in comparison to their lower level counterparts. {#1040}

Yes, but if those higher level concepts can be shown to be built from their components and those components are reducible to mathematically expressible and verifiable expressions which can be shown to be valid at every level from the lowest to the highest, then you really have something useful for an organization. {#1072}

Well, it would be great if validity can be shown to hold uniformly across multiple abstraction levels. I am afraid this is not possible since verification and validation can not be complete -- we can verify and validate within some experiment. {#1083}

7.2 What does it matter?

Managers, especially those who are not technically oriented, but who make technical decisions can be just as easily convinced by a technical approach that is not formal, but delivered with conviction as they can by a technical approach that is formalized. It is import for us to constantly insist on formal descriptions of ideas and approaches so that they can be scrutinized by others. {#672}

I would add that most managers do not have enough engineering background to understand when they are being deceived. {#687}

Excellent point! There should be a place these managers can go for some objective advice. A formalism requirement would allow objective consideration of an approach by many different sources. {#713}

Managers have to be persuaded that the formalization is worth the effort. To get managers into it I am not quite sure whether abstracting formal frameworks is the way to go, though. I would rather suggest to develop a suitable collection of success and failure stories that are related to the problem at hand.

The presentation of the (intermediate) results should be made at a level that allows managers to draw their own conclusions. However, I do not know whether this is what you meant with "abstracting formal frameworks". The presentation of the results would likely require the development of new visualization techniques that support the understanding of large scale software systems. {#770}

Managers, even dumb ones, recognize the value of math as a language that is unambiguous to those who understand it. They may not understand it, but they acknowledge its power to persuade. In the same way that they may not understand French or German, but recognize it as valid language for the expression of some ideas. Math is better because it is understood across domains, languages and cultures. Well most anyway. {#849}

Systems fail because they do not meet their requirements. Often these are badly framed, incomplete or misunderstood due to failures in understanding. {#955}

Decision-makers generally have to interact with audiences that have varied backgrounds and therefore important to distill knowledge to make it useful. {#1053}

7.3 What are the barriers?

Overcoming the idea that a formal approach is too inaccessible. Again, this requires technical people to be able to abstract to different audiences at different levels. This requires the use of analogies, metaphors or examples that convey the idea. Mathematics is a very precise metaphor, but not the only one that can be used to convey information to decision makers. {#701}

Most formal methods are simply mathematics. There is a very fundamental difference between mathematics and engineering models. Engineering models bring mathematics to problems in a pragmatic way and predict information of interest. Fourier analysis is a great example of a mathematical theory that has been brought to bear on a problem through development of engineering models. We simply lack engineering models in software development. {#731}

Need to make it relevant to the manager, if they don't see the relevance, they won't use it. {#879}

Exactly! You have to appeal to managers' fears or desires or something to get their attention. If you can provide something that alleviates their concerns or achieves their goals, they will be your biggest supporter! {#893}

A view that says we are all involved in meeting the business, human and other stakeholder needs of our organizations does not exist. {#961}

No, it does not. However, if we express our ideas formally and the formal language is comprehensible across divisions, then those domains could abstract from the other domains relevant information. {#1079}

Formal methods people are part of the problem. They say things like "math is unambiguous"; "testing cannot prove the absence of errors"; "formalism

precisely defines the spec". Either we don't understand what motivates managers or we sell them short - managers realize that math is unambiguous - but they worry about their technical staff being able to use it, communicate with it, and not make mistakes with it. They know that testing cannot prove the absence of errors - they also know that nothing can, including FM. They are less concerned with precisely getting a spec right than they are with making progress on the project knowing that the specs will change as the project moves forward anyway. Sometimes FM practitioners are their own worse salespeople. {#1091}

7.4 Why is progress possible?

Progress is possible, but it takes conviction on the part of 'marketers' of formal methods. There also must be a critical mass of engineers who are on-board and committed to delivering the message. We have found that it takes at least three respected engineers and a couple of years to deliver this message in an organization that had no previous formal approach to model/software design. {#750}

Managers deal with abstractions today with regards to budgets, progress, and personnel. They have charts, visualizations, etc that they use daily to understand how well they are doing with regards to these important issues, now we want them to understand technical/domain understanding as well as business needs. {#889}

Yes! We just have to relate the formalism in way that they also understand. They certainly don't understand complex economic, psychological or accounting models, but they are given the tools to visualize and make good decisions (ostensibly) from those tools. We should be able to provide analogs for technical matters. Formalisms at the right level of abstraction can do this. {#935}

because this is the reason systems are now known to fail to be released. {#962}

Could you explain your comment a little more or give an example? {#1046}

7.5 Who will be involved in the solution?

For a medium sized organization - 500 to 1000 people It takes a reasonable, communicative, capable technical person who has the respect of management. It takes a manager who will listen to innovative ideas or one who is on the spot to fix things and leave a trail showing how they were fixed. It takes 2-4 engineers who can implement the formalism and build software that delivers on the promise that a formalism is a better way to go. It takes a public relations campaign in the organization to let others know that it was successful. {#768}

Cross-disciplinary researchers with a view encompassing technology, business and social psychology. {#968}

7.6 How can we judge success?

Success can be judged by the spread of the idea of a formal approach horizontally and vertically through the organization and by outsiders inquiring as to how they can use the same approach. {#777}

Improved success rates in implementation of software intensive systems. removal of barriers between stakeholders in organizations. {#972}

Embedded knowledge delivered to the decision-makers has something like 80/20 ratio despite the filtering done. {#1066}

7.7 How much will it cost?

3 years and about 1.5 million dollars, a marriage an ulcer...etc. {#789}

7.8 Waypoints?

Pre-plan. Find managers who are open to new ideas that are V&V-able - not too hard really. Suggest that there are approaches out there that give them peace of mind and traceability.

- 1) Introduction to formal approaches. Metaphors, analogies, historical precedents.
- 2) Simple examples done quickly that can be shown-off.
- 3) Suggest that this approach be tried on a real problem that they are faced with.
- 4) Deliver on what you promise, that is don't promise too much too quickly.
- 5) Keep delivering, improving; finding new applications
- 6) Get some public acknowledgement! {#818}

Education is the critical on-going process. Get as many participants in your educational programs as you can. {#828}

7.9 Timeline and payoffs?

See previous thread. The time line should be about 3 years. For a medium sized organization. {#823}

7.10 Other issues?

Emphasize the benefits of formalisms in other domains, disciplines. Acknowledge where formal processes break down. {#833}

We need to start talking to the other research communities involved. {#979}

8. Dynamic systems frameworks for designing real-time applications with distributed components

Combining Software/Hardware/Communication into one strategy/method. {#601}

Please define "frameworks". Does this mean a fancy GUI for displaying data flow, subroutine calls, message exchanges, etc? While such things are nice for "view graph engineering", there's no substitute for a solid understanding of multi-process interactions, gained by experience, not by a fancy tool. {#602}

Framework in the general sense of a methodology, techniques, algorithms and tools to support that design task. A solid base to understanding process interaction, system architecture, timeliness, criticality and distributed data/software {#798}

Let's call a framework something like an algebra: there are sets, functions, relations, axioms, for a formal theory and a pragmatic body of knowledge for solving problems possible in the theory. However, I think this is a wrong question: Are dynamical systems concepts a suitable basis for a framework for applications with distributed components first and add real-time later. {#892}

8.1 What is the challenge and why is this a challenge?

What does the "real-time" imply here? Isn't this true for "non-real-time" systems? {#715}

This dynamic framework business is way to complex. The framework should simply let components operate i.e. it is the 'motherboard' that allows components to work. Let the complexity be tackled by components and their assemblies.

if the assemblies become too complex, have a hierarchy of assemblies. Make the framework be such that it admits hierarchies of components. One of the easiest ways this can be accomplish is by making a framework self-similar or recursive i.e. framework A can encapsulate framework B, where both A and B are frameworks of the same type. {#746}

It is true for non-real-time systems, but it's even harder for systems with timing constraints. Some of the existing frameworks work decently for non-real-time apps., but fail when timeliness is put into consideration. {#811}

I wrote a paper that tried to tie DEVS (as an example of a general framework) to the logical realm through category theory (for the algebra). The categorical world has shown that the fundamental issue in computation is partial orders, something easy to manipulate. If you force dynamical systems into the programming paradigm, then we get rid of a lot of problems. This is a challenge because current categorical concepts are too blunt an instrument.

Dynamical systems can also tie into topological concepts (like limits, termination), hence helping us on the whole computability question. {#944}

8.2 What does it matter?

I feel that dynamical systems are a correct formulation. They are understood by most engineers although my science colleagues are totally unfamiliar with the term --- but not the concepts. Having a consensus starting point would be a big help. {#952}

8.3 What are the barriers?

foundational thinking as it occurs in some areas of computer science and mathematics is unlikely to be tolerated by non-theoreticians. some of this stuff is heavy lifting; hence, the reluctance to use formal methods. {#964}

Right now, this is pretty esoteric stuff. CS theory is bogged down on NP-complete stuff. Categorical or logical foundations are not widely known in the community and non-trivial to acquire. {#1016}

8.4 Why is progress possible?

Many of the pieces are available. If you take "semantics" as the issue --- and I do --- then algebraic techniques are the obvious starting place. {#971}

8.5 Who will be involved in the solution?

People who have real applications with extreme reliability/correctness requirements. And theory people bored with NP-completeness. {#976}

8.6 How can we judge success?

As in the "abstracting..." answer, "routine" things should be routine. Research lies in the undecidable properties. {#981}

8.7 How much will it cost?

You will have to change the way programming is taught and develop the tools. It's perhaps more redirection of money. {#987}

8.8 Waypoints?

I'd say in 5 years one could do the survey of models we have and to develop the algebraic techniques for transforming models. {#995}

8.9 Timeline and payoffs?

This is a long range problem. 20 years. The payoffs are identifying routine and non-routine; easy from impossible. One might dare call it "software engineering" at that point because those ideas of engineering design that we use for control systems today would be used by designers. {#1002}

8.10 Other issues?

9. Overcoming compartmentalization to develop joint programs in computer science and business departments

We have this. It's called MIS. {#734}

Validating large software systems {#598}

It is easy to confuse the tool and the product. In the case of software it is both a tool and a product - thereby supporting a fairly large industry segment. Business schools have traditionally looked at business generically. But, the growth of this segment and its influence on almost all other industry sectors brings us new and challenging issues in running the business. It makes a lot of sense to bring these two schools - computer science and engineering and the business schools to study and explore this space. {#707}

This is baloney. Are we so arrogant that we think that everyone should be a computer scientist? Everyone needs to understand abstraction, modeling, layered design, etc. Of course not, just as I have no need to understand corporate finance and accounting. {#716}

Yeah, finally someone called it for what it is. Modeling and Sim have its purposes and its place. There are uses for it. But for the majority of the populous there is a huge lack of need for this. {#1077}

Problem is not stated well. Should be addressing not academic departments only, but also the business world of the developers. Recommend changing to

Overcoming compartmentalization to develop cohesive diverse teams of specialists from different organizations. {#730}

Definitely agree with that comment. Compartmentalization is a huge issue in any large organization. {#736}

agree with 716 {#752}

I have to disagree with #716 and #752 from the perspective that the systems we are building are becoming much more global in nature involving suppliers and clients of service commodities. The supply chain management folks are going to have a big impact on how we develop service-based systems (both web services and grid services); we can learn quite a bit from them in that respect. {#800}

Disagree with (#716). Software engineering has a closer connection to the business school issues than even the EE part of ECE. for instance, people issues,

project management, coordination, and program management are key ingredients of the software engineering practice. These are today only discussed in the Business school and graduating students come poorly prepared for a life in the real world. {#810}

I agree with 716. We don't teach project management in other engineering disciplines, why do we think that it is beyond business schools to teach software project management? They might actually prove to be better at it than we are. Because we have so little software science I think we have become far too reliant on software management. {#880}

I have been told by students that project management books and classes deal with the factory scenarios that are more related to the past than the present IT industry. Comments please. {#897}

I disagree. I recently attended a PM class and it was very good about covering the traditional industry, but there was an entire section on IT. It was great and really revealing. {#1068}

One of the real challenges in this space is that the process of management is as much people management as it is project management. Industrial Engineering stuff that was developed for the manufacturing sector dealt with labor issues for a class of labor that is not as educated, trained, or intellectually demanding. The only people who cater to more sophisticated people management issues is the business school, or some HR discipline. Thus the rift. Comparing software engineering to manufacturing does not work - because repeatability is the key to manufacturing - where as in design and development we labor over one thing. Finally, we all share the work break down principles, but coordination, enforcement, and management are very business issues. Some will argue that it has to do more with military strategy and teamwork than Gantt charts. {#983}

This is too restrictive. We need a range of social science input to deal with making our techniques useful. Software engineering is about people designing systems for use by people. {#996}

This is a classical problem in interdisciplinary development. The problem exists because the institutional structures make it happen: most departments pay lip service to interdisciplinary but don't reward it. {#1013}

I believe that software engineering degree programs now emerging will partially address this issue. I come from an ABET accredited SE degree program - the program crosses departmental boundaries now with a heavy contribution from Industrial Engineering (PM, Quality); Computer Science, and some business skill development. SE's need a much broader education than a CS or EE or ECE program provides {#1030}

Can someone write down a curriculum that a software engineering 1st line manager, and director (2nd line manager) needs to have ? {#1064}

9.1 What is the challenge and why is this a challenge?

How is this relevant to DLS? {#866}

Maybe you haven't noticed... Computer Science is REALLY NOT a Science! {#1071}

If we want to design effective systems, we need to capture their role in organizations. That involves understanding business processes as part of requirements capture, but also the social and organizational impact of what we do. Build a great system, which alienates workers when it is deployed, and you have to ask if you have been successful. {#1007}

Problem is not restricted to business. The challenge is that we've always had institutions that we separate. Clemson was reorganized to try to make it happen; it hasn't. It takes a long time to change attitudes. {#1022}

From an academic's viewpoint - university traditions are a challenge. Working across departments or colleges becomes an administrative nightmare and there is little reward structure build in. {#1039}

9.2 What does it matter?

We need to build systems that deliver on what they are built for. {#1011}

this is what it's all about, fans. Name your favorite two disciplines. In today's world, no meaningful problem is solved in one discipline. We must break down these walls and realize that each party brings expertise to the table. {#1029}

This really "doesn't matter". We need to focus on making sure our students have a broad education, are open to new ideas, appreciate a global perspective, and are able to work as a team. Addressing a question like getting the CS and Business dept to work together isn't important. {#1054}

Traditionally, Computer Science departments are considered the elite (science) oriented and thus "harder" domain to master, while Business Schools are considered the "easier" and less elite department. To have them work together, they have to come to think of themselves as equal peers that both have something to bring and take from the relationship. Must make it a win-win situation. {#1069}

9.3 What are the barriers?

Lack of expertise in both in academia {#624}

The language of communication is currently very specialized, and although we see the need, the departments don't see the same need as they don't communicate or interrelate often. {#738}

Human nature. {#739}

Poor media for communication. Fear of straying into unknown intellectual territory. Arrogance. {#1014}

The academic problems are rife. Tenure and promotion are the purview of the departments. If someone has a simple well known idea that can be applied to a problem in another discipline, such activity will be given no reward. I know that at our school the new assistant profs are told not to do interdisciplinary work. {#1043}

#1043 you hit the nail on the head. Academic politic that values one kind of work over another pays little head to what the industry or world needs. {#1081}

CS departments think of themselves at the elite and business schools as a lower level of education and "easier". What needs to be understood is that one is not the inferior/superior to the other. When I was an undergraduate in EE, we used to joke if you can't make it in EE then you can always go to Business. Additionally, when I was a young EE, it was considered an act of "prostitution" to go into "management" rather than stay in the R&D (hard) environment. {#1082}

9.4 Why is progress possible?

Large corporations are seeing this as a key problem area. {#1018}

There is some pressure from the industrial world to get into interdisciplinary work, be able to work in teams, etc. Ultimately, progress is possible because people retire and slowly the new comes into be the orthodoxy. {#1049}

9.5 Who will be involved in the solution?

We all are: Administrators of universities, faculties, and the user of the academic product. {#1051}

9.6 How can we judge success?

We will judge success to have occurred when companies can form interdisciplinary teams without all the posturing. Now, individuals will posture---human nature---but we will see team efforts as being the norm. {#1059}

9.7 How much will it cost?

It's gonna cost human capital. People in this workshop probably are less likely to see interdisciplinary as a problem. However, the university culture changes very slowly. The other cost is problems not getting solved. {#1063}

9.8 Waypoints?

One way point is to see less rigid curricula with more "capstone" projects done with interdisciplinary teams. {#1067}

Another is to see seasoned engineering managers come back to the CS department for short term courses. Now-a-days they go and get an MBA {#1073}

9.9 Timeline and payoffs?

This is a hard one because it's cultural. Us old codgers remember when there were very few women in the technical fields. This is not nearly so much the case today but women still make much less and have fewer managerial positions than men even after 40+ years. Payoff is enormous: more problems solved. {#1076}

9.10 Other issues?

10. Coping with a software industry in flux--changing skill mixes, user expectations, labor displacements

This is really a huge issue. When is the last time we had a major change in, say, electromagnetics that had effects on how we teach and design electrical systems? In contrast, we've changed paradigms for software design in fundamental ways at least 3 times in the last 20 years. That doesn't even include the smaller changes! {#763}

Agree this is a huge issue. The "software paradigm" seems to shift every few years so that skills that I have are continually becoming obsolete. At one point I was a real guru at Dos and Turbo Pascal. I couldn't get \$0.10/hour now with these skills. I have not taken the time to come up to speed on the Windows API (I should and would enjoy it) but just cannot find the time. Learned Unix and SunOS in late 80's...these skills somewhat transferred to Linux, but not really. Have recently become fairly skilled at C++ and the STL, and I wonder how long these skills will be relevant, especially in light of how many man hours I have in learning these things. {#785}

well. we have to accept the fact that programmers are like the steel workers of 21st century. {#805}

Interdisciplinary thinking may provide us with an edge, the ability to pull things from diverse areas and combine them in unique ways to create new products, ideas, etc. {#809}

I would submit that the labor displacements to low cost regions will have a huge impact on how well we design systems. If we are doing design in-house and sending the development to a global set of low cost regions, we need to make sure we have representations and tools that will bridge across different skill sets, cultural norms, and the like. {#824}

One would think that in such a climate, foundational skills - such as abstraction, verification and design should have increased long term value - since there are the only ones that change more slowly than programming language skills. I would like to conjecture that it would bring back a greater demand for such education - the classical computer science ed. But, hardly see that trend. In fact, the opposite is quite true, when no formal CS education is OK so long as one can hack VB or Java, or know the EJB stuff. Where is this going? {#847}

When all else fails appeal to vanity or exclusivity! {#882}

reply to 847. Well there are a lot of demand for low skilled programmers (used to be at least), but only a handful of architects with abstraction skills are needed per

project. So it is a smaller job market. This should not be surprising since a lot of the code in a software system are just routine code/ {#903}

The temporal ness of software skills is largely driven by an industry that can change standards as a means to keep control over a market segment. Why else do we see fights such as that between Java, C++, and C#. SQL managed to keep things straight for a long time, but finally the divergence helped one company or the other. This is at least one of the aspects of a volatile market. The other element is pure fad, and advertisement value. The consequences of this on the labor market have been both positive and negative. As skills get narrower, we need more people, although they will never quite be 100% utilized. So, if you choose to run Lotus notes, then one needs two or three people who know that, and several of them don't know much else. Gradually, we will become like the medical industry. Where between 15-25 people show up to deliver a baby - work for a few hours, and then disappear. We are almost there now in many segments of the business - thus the growth of the services market. {#920}

Global outsourcing is an huge gamble in this area. I have seen several companies taking things back in house because of maintenance problems. {#1023}

User expectations are a serious problem. technology departments are now very conservative and defensive in restricting user requirements, to stop requirement creep overwhelming outcomes of otherwise "successful" projects. {#1027}

10.1 What is the challenge and why is this a challenge?

The challenge is to train people so they are able to offer very high quality work over a reasonable time period. In my humble opinion, quality of training may provide the key in the long term {#795}

The challenge is that everyone wants to "improve" software design cycles, reliability, etc. It seems that every improvement means that old skills are obsolete, and new skills are required. Why can't we build upon previous knowledge rather than replace it? {#807}

Convincing American students to enter a field where they see declining salaries, job insecurity, and short careers. {#884}

What do you call a 40-year-old programmer? (Scroll down)

An insurance agent {#885}

How will there ever be experienced people in the computing field? Will they only survive as consultants? Who can mentor new engineers? {#895}

10.2 What does it matter?

10.3 What are the barriers?

Government policies that fail to provide full support for engineering students in a time of increasing tuition and decreasing expectations of career earnings. {#898}

10.4 Why is progress possible?

10.5 Who will be involved in the solution?

10.6 How can we judge success?

Ease of retraining professionals to change from one domain to another (e.g. from finance to HR as an example) (I'm talking about IT professionals who have a background developing finance programs moving to a role in developing HR programs) {#816}

Time to move from one domain to another {#832}

Can we achieve a certain level of stability in the skill mixes that are fundamental. {#835}

Can our work have an impact on the flux itself? Can we gain more stability in the field? {#842}

10.7 How much will it cost?

10.8 Waypoints?

10.9 Timeline and payoffs?

10.10 Other issues?

11. Bridging gap between software design science and domain scientist users

tough luck. if the domain scientist needs sophisticated software design ideas, he'll read and find out about it. If he doesn't and invents his own than (a) either he's wasted a good deal of time reinventing the wheel OR (b) the design principles that exist are no good.

Let the most capable survive. {#760}

Systems like Excel or Matlab do a pretty good job at this. {#782}

We don't need to bridge this gap. I have been involved in a number of successful software design, implementation, and deployment projects. The only way to be successful is to have both a domain expert and a software design/development expert working elbow to elbow. I don't expect the scientist to understand multi-threaded interacting applications, and he doesn't expect me to understand particle physics. {#827}

This has been identified as a major problem within European Union funded projects. We need a Babel Fish for software projects. (If you don't know what a Babel Fish is, you see the problem in action! Read Douglas Adams' Hitchhiker's Guide.) {#1042}

Good systems design and middleware can also help in bridging. E.g. by providing resource scheduling, transparent data transfer, consistent software environments across distributed resources, standard interfaces, one can make software developed by domain scientists tap into a larger set of resources (e.g. a computational Grid), which may enable solving problems that would otherwise not be feasible. {#1080}

I agree with #827. This is also answered in the previous "business" question. Best I can tell, scientists and engineers and business people suffer from the same issues about software that I have in their areas: we lack detailed technical knowledge of the other person's world. Teaming literature reports that about 1/2 of any group's lifetime is spent getting vocabulary and concepts straight. {#1090}

11.1 What is the challenge and why is this a challenge?

What domains should we concentrate on first? I would recommend making a short list such as finance, medical, communications and concentrating in these limited areas first. (note the three domains I listed are just examples, and could easily be changed.) {#759}

Users will use methods that are available to them and well-understood to design software. I think the challenge is developing techniques that are close to well-understood methods and that users can relate to. {#764}

Need models and languages that are effective and useful for both communities. {#779}

Empirical science is based on investigation and observation first and foremost. Modeling comes later. One obvious way to attack the problem is to look at complex software that exists now and identify characteristics that make it successful, unsuccessful, and why it exists at all. Usually in complexity or chaos theory, we try to discover scaling laws that do not tell us how it occurred but rather what its (emergent) behavior is. {#801}

Do we want to the domain expert to build his complex system in the vein of 4GLs of the past? Or are we addressing very complex systems and need to have developers who are needed to help translate user needs into system developments? {#829}

A science of modeling and simulation acting as a "bridge" to overcome complexities of the domain in ways the software can effectively handle them. Not creating a disciplined bridge to support knowledge transformation from one abstraction to another cannot handle complexity in a general fashion. {#844}

Domains that have advanced their products to the point they need more DLS techniques are ripe for them. Develop domain-specific models, techniques, and tools. If not ready, anticipate where problems are likely to arise soon and do the above. {#947}

Lots of people have lots of preconceived notions. I don't have time --- even if I want to --- learn all I need. But must be able to learn quickly --- lifelong learning. We don't understand the psychology and sociology of interdisciplinary work. {#1094}

11.2 What does it matter?

We need to help the customers (the domain scientists) be able to communicate with the design scientists (CS/EE/??) {#765}

If the two sides can't communicate and have a common picture, our hope to improve is greatly diminished. {#773}

We need the domain scientist to solve their problems, not retrain computer scientists to solve them badly. {#790}

Not developing a disciplined framework for the bridge will not only results in the continuation of the trends such as the Standish Group, but with higher degrees of freedom demands will make it worse. {#855}

It matters a great deal. The complexity of the natural world is large and computational scientists are in the business of modeling it. Their models will mirror the complexity of nature and will require software methods to deal with out. {#901}

It matters a great deal. CS people are not god, they are people who have very strict sets of blinders on they only see things from there point of view. Any other way they are confused. Getting collaboration from any other angle can really help solve something a different way. I enjoyed the allusion to the complex network not being complex as the human body. There is stuff to learn from other sources for sure. {#949}

Science is too complex for just one discipline --- even "just physics" --- to do it all without cooperation. {#1096}

11.3 What are the barriers?

Lack of education at all levels - decision -makers in industry, government, and "academia" -- about the role and significance of bridging the gap between the two. {#876}

Ego, big freaking egos {#953}

#953....right on. {#1097}

11.4 Why is progress possible?

I think it is possible to provide an understanding of the domain to the system designer (it has been done in the past) and the design knowledge to the domain expert (that has been done in the past was well with 4GLs), what is unique here is the large scale, asynchronous, complex nature that we are dealing with. What we have to answer, how much of both domains do we hide from the expert in that domain and how much do you need to reveal? {#1058}

11.5 Who will be involved in the solution?

Both theorists and experimentalists are needed as well as software/hardware/bio/... engineering and domain experts. {#894}

Systems experts - part of bridging this gap often includes delivering high-performance environments that are not too complex to use that domain specialists can actually make use of them. {#1048}

11.6 How can we judge success?

speed to develop from initiation to first delivery {#784}

How well the products are accepted and used by the user community. {#787}

Number of trouble reports and change requests {#791}

Increased use of the concepts, theories, and methods -- the "bridge" helps to better solve complexity or makes it possible! {#902}

A solution that is engineered that make people say wow, why did I never think of that. {#954}

11.7 How much will it cost?

11.8 Waypoints?

11.9 Timeline and payoffs?

I think lack of such a bridge is a "time bomb" -- we may find ourselves in situations (e.g., crash of critical worldwide systems such as aviation) with catastrophic consequences that may not be reversible. {#942}

11.10 Other issues?

This problem can easily morph itself to other things and degenerate into other issues and concepts. {#958}

Developing approaches for consistency of abstractions -- and therefore data -- for very large systems. {#998}

Knowledge persistence for large- or ultra-scale models is a central problem. {#1003}

Need to have ways to "recycle" and "purify" knowledge. These capabilities are important for building the bridge. {#1017}

12. How much control can we place on software complexity before we limit possibly advantageous outcomes? Is Linux possible under a design methodology or is it a design methodology?

This is a really interesting question. In digital hardware design, the most severe restriction I can think of - only 0's and 1's - is enforced. It makes much of what we do possible and seriously reduces the implementation complexity of systems. Understanding that we may need to give up some of our flexibility in software design is necessary for progress. {#890}

So are you saying that the real test of applicability of design would be whether there was adoption by an open source project? I guess this begs the question "Is open source software better than closed source?"

I am not making value judgments. Not better not worse, but the fact that it exists at all and is popular and complex at the same time. It followed no over-arching methodology to its creation. People found places that they could add code and did so. {#896}

I believe Linux is possible under a design methodology. {#911}

Agree. This is more of an organizational problem than a technical problem. {#939}

It is not at all clear that open source is better than closed source. Some indications are that the error rates in the most recent versions of Linux are moving towards that of Windows. Also, it is extremely important that the development community associated with an open source system be well understood. Various independent investigations have shown that serious errors lurk in widely disseminated open source systems and have not been fixed through numerous iterations. {#940}

Let me clarify (This is Rob Armstrong): Complexity is an artifact of software that seeks to solve a complex problem, obviously complexity is not a design goal (no one seeks to create complex software as an end in itself). Design methodologies (waterfall, spiral) limit the space of possibilities, hopefully to ones that will speed the time to solution, but they are limiting nonetheless. To what extent do these methodologies limit possibly advantageous solutions? {#941}

Linux is a reimplementaion of something that already existed. I don't see why they couldn't have used a spiral model (or any other model for that matter). If you mean the open source development paradigm, it certainly has its own set of problems. {#951}

Clarification to the second part of the question: Linux has no identifiable design or methodology (let alone design methodology). Why is it so successful? How

is it that it has become one of the most complex pieces of software on earth without a methodology. Maybe there is one. Maybe it is **organic** in the sense Salim presented earlier. I think it **is** a design methodology, but it involves a healthy amount of chaos as part of its paradigm. I think the most successful (not necessarily the best) software evolves. Evolution needs chaos to thrive -- but not too much. Evolution surely can't be planned. {#965}

Non-sequitor {#966}

Disagree w #951. Linux is a REDESIGN of something that already existed. The "already existed" can be thought of as the requirements document. Linux was and is successful due to the overall design conceived by one person, and carried through in a very well done piece of software. However, I'll side up with those claiming that open source also has problems. I personally have looked at the Linux kernel in detail, particularly the protocol stack. Even with 25 years of experience of looking at software and modifying it, I'm pretty much stumped by the TCP/IP implementation in the LinKern. There is a good "coding standard" for Linux kernel software, there is not a good "coding style" guide. The number of re-casts and re-use of variables for different purposes renders the code virtually unmaintainable. {#973}

Yes,.. and even commercial systems that have been re-designed on the average do better. In fact, there is hardly a major vendor around with software that has been around for more than 10 years without a major re-design. A complete re-write is more rare, but even today large parts of Windows are being restructured. CICS restructure was so talked about (recall use of Z) and it make the core squeaky clean. And that too was done by a handful of people. So, the Linux story should not be such a surprise. The surprise is an individual doing it on their own time and money - and then drolling it out free {#1033}

What are the root causes of complexity? E.g. making it possible for one component to change, adds a little complexity. Making it possible for all components to change makes a system **very** complex. Indeed so complex it can't change predictably. So is designing for change a root problem? or is designing to not change? {#984}

Answer to #984 : If one is in the business of developing software - frameworks which do not implement very many algorithms as well as components that implement algorithms - we design for change. Always. Even porting to a different machine involves change of some kind of other. Designing something with the hope that it remains static doesn't seem to work in practice.

So we parameterize, parameterize, parameterize Is this what you mean by designing not to change ? {#1024}

There is another answer to #984: complexity is an artifact of the problem the software is trying to solve. Put another way, complexity is an artifact of how the software is/was-intended-to-be used. Complex software, like Windows or Linux, solves a complex problem. No one wants software to be complex for its own sake, it *must* be complex to do the useful things that it does. {#1065}

Software that is able to "evolve" in the face of changing requirements, as Linux has, is a sure sign of a well designed piece of code. I often say that "the design is holding together", meaning that as new features and requirements are added, they can be done so without a complete redesign or a complete rewrite. The question is, did Linus set out to design software that would evolve as Linux has, or did he set out to have a nice, clean design with clean interfaces? I suspect the latter. {#993}

Linux has become robust over time with extensive use and more and more bugs fixed. It is not the methodology, instead usefulness and extensive use of the product that has made it successful. {#994}

#994: But isn't that a de facto, possibly emergent methodology? Or a methodology for which the software solution will be found, but by a means that is unpredictable? {#1020}

How many people in this room have ever re-compiled the Linux kernel to add or remove a module or feature? I have, and let me tell you it's not for the faint-hearted. There's a nice x-window GUI that guides you through the process but determining which boxes to check and which to uncheck is pure guesswork without looking at the source code. Since I'm a systems guy at heart, I actually enjoy this, but doubt that many others would. Certainly Linux is not ready for my Mother to install and deal with (neither is Windows for that matter...) {#1009}

Yep I do it all the time. What I can't seem to do is install windows on bare metal (empty PC). But that is probably because I don't run windows. It is just what you happen to be used to. {#1034}

I think this is a major research topic. We see claims about open source, XP etc., but it seems clear that they are context dependent. What is interesting is that good solutions to important problems almost always seem to emerge by one route or another. {#1055}

The major tools that seem to be employed in the open source community are configuration management and other source-based manipulators. I don't believe there is much more than that. Clearly, this is not enough for general systems development. Open source obviously applies to certain contexts but would not extend to general systems. {#1075}

12.1 What is the challenge and why is this a challenge?

12.2 What does it matter?

Too much control impacts "out of the box" thinking. {#992}

12.3 What are the barriers?

12.4 Why is progress possible?

12.5 Who will be involved in the solution?

12.6 How can we judge success?

12.7 How much will it cost?

12.8 Waypoints?

12.9 Timeline and payoffs?

12.10 Other issues?

13. Making simulation *THE* way of addressing predictability in software intensive system design

Is this rapid prototyping? Executable specifications? {#945}

Why does simulation have to be "THE way"? It should be a tool like any other (like FM for instance) and applied when appropriate. The problem is that the disconnect between the software folks and the simulation folks has to be bridged. I need to be able to run simulations as I develop the software without incurring a large cost in developing the simulation. {#957}

2/3rd of the lifecycle of complex software intensive system development is spent in maintaining. Engineers need to predict the implications of "possible" changes, substitutions, or alternative design organizations on the behavior of the system. Yet, we still do not have effective analysis capabilities to explore the implication of various potential changes that will lead to "possible" different systems. How can simulation modeling be used to facilitate exploratory analysis? {#960}

957 is correct often we put too much emphasis and hope on modeling. By calling it THE way you are excluding other solutions that may come along. Simulation is a tool, that is it. It is not going to solve all of the world problems. {#975}

In the digital hardware community, simulation is ubiquitous. VHDL and Verilog are everywhere and the next generation of languages is being proposed (SystemVerilog, SystemC, Rosetta, etc). Unfortunately, system complexity is increasing exponentially and simulation capabilities are increasing only linearly. The hardware community is begging for new predictive mechanisms. When the semiconductor industry council defined requirements for their next generation language, they were (in order of importance): (i) formal semantics; (ii) constraint representation; (iii) support for static analysis; (iv) REDUCED dependence on simulation. There are other factors at work here, i.e. the design abstraction level has been RTL for about 14 years, but it is worth noting their problems. It is also worth noting that with only one exception, the next-generation languages noted above are all simulation languages. {#986}

Re #986: Some AMD chips were formally modeled in ACL2 (essentially a theorem prover). Of interest, is that the formal description could be compiled and ran faster than the AMD C-based simulators. Yet, the ACL2 description could be formally reasoned about. Also, in general, mathematical formalism allows for symbolic reasoning. This allows for enhanced scalability. Today's description of the firewall analyzer and network analyzer falls roughly into the same camp. {#1085}

Depends on what you're simulating {#991}

Simulation would be helpful in many aspects of software architecture development. There are some interesting questions that could be answered by having access to simulations as inputs to the design decision process. However, we still need to have access to other tools in addition to the simulations because we can only capture so much and some of the really interesting properties are only observable when you have the real system. {#999}

suppose people simulate a system for non-functional property such as performance? How credible would it be? {#1012}

is there such thing as simulate the reliability of a system? {#1015}

What is meant by "simulating a software intensive system"? Isn't this just debugging and testing? Isn't running a piece of code under every conceivable set of inputs a "simulation" of how the software will behave in the field? On the subject of debugging, I'm appalled at the sorry state of debugging skills in the software community today. Even programmers with years of experience often have not a clue how to go about tracking a particularly elusive bug. In the early days of O/S design and implementations there were "In Circuit Emulators" (ICE) that could "simulate" a piece of software (perhaps an interrupt service routine) and capture precisely the behavior for post-analysis. I miss those days and wish such tools were still in widespread use. {#1031}

The issue is to enable "effective", scalable, and tractable predictive capability to help an engineer by providing a capability to consider 1000s of alternatives concurrently. Current practice in the software industry (i.e., Navy's CEC system) is to make a few changes and test weeks/months to determine implications. What new advanced simulation modeling methodologies can help us analyze multiple scenarios and options concurrently. can simulation models advise engineers to use certain configurations to optimize specific quality objectives? can a simulation model decide on its own at run-time to suggest feasible effective designs? {#1047}

It isn't. There is no silver bullet, remember. We need all the techniques we deploy and some that we haven't thought of yet. Simulation is like testing, in that they are both essential and highly expensive. {#1061}

13.1 What is the challenge and why is this a challenge?

The gap between performance of a prototype simulations of the system and actual system. {#943}

I believe simulation, integrated with modeling technology, is one of the most promising technologies for designing large-scale software intensive systems. {#959}

Do you mean simulating the design? How can this possibly be done without actually coding the components? How will we know a design holds together until the rubber meets the road? {#1044}

Consider the simulation-based acquisition (SBA) concept advocated by DOD. It may stand for a new way for system development {#1070}

One of the questions that need to be answered is that how much simulation is "enough" to address the predictability of a system. {#977}

System complexity is increasing exponentially and simulation capability only linearly {#989}

Do studies exist that have analyzed the increase of software systems complexity and simulation capability? What is meant with systems complexity and simulation capability? Does the later refer to the expressiveness of the models, the execution of simulation runs, or ... ? {#1037}

Is the simulation trustworthy? How do you validate the simulation? {#1001}

(response to 989) That's why we need new advanced simulation methodologies that can explore such non-linear complexity {#1078}

(response to 1078) I'm quite interested in knowing what technologies would provide this. I'm not a simulation person, but I worked with parallel simulation people for several years who were very good at what they did, yet got very little throughput gain for their work. Where we might win in the software community is that we can increase our abstraction levels much easier. Still, if you want your simulation to reveal detail, you have to have a detailed model. {#1093}

13.2 What does it matter?

Predictability is a crucial aspect of any engineering discipline, including software engineering. Engineers need to be able to analyze the artifacts they develop to make inference about certain properties and features of interest. Unless we have such a capability, one can not claim about the science of design. {#1088}

13.3 What are the barriers?

ROI {#1028}

Want to see simulation as part of a seamless process {#1032}

The integration of modeling and simulation with other methods for specifying and testing software systems seems to be a challenge and

barrier that has to be overcome. How can simulation for testing software be related to other model-based testing methods, e.g. that automatically generate test cases based on the specification of the software. Simulation is used for a kind of black box testing, can it also support white and grey box testing? {#1084}

I can certainly imagine doing fault injection as part of simulation. would that count as grey box testing? {#1095}

Existing simulation modeling methodologies are not effective enough to facilitate multisimulation of alternative designs {#1092}

13.4 Why is progress possible?

13.5 Who will be involved in the solution?

DARPA and DMSO {#1038}

13.6 How can we judge success?

Impact on schedule {#1041}

13.7 How much will it cost?

13.8 Waypoints?

13.9 Timeline and payoffs?

13.10 Other issues?

Appendix 2 Record of 2nd Day Session

1. Challenge: *Develop new software design paradigm that recognizes the inherent asynchronous nature of next generation software systems/applications operating in large scale, networked, distributed software.*

As Is

- Too long to test commercial products
- Software Engineering does not use any modeling and simulation
- Systems built by composition are few.
- Mostly evolutionary
- Emphatic assertion, no empirical analysis

Future

- Automated Evolving Adaptive Systems (HAL)
- Distributed and Ubiquitous
- Autonomic, Large-grain, Functionality
- Continuously working gathering information

Vision: from Here to There

- Simulation integrated into the development lifecycle.
- Making composability a reality (+ M&S)
- Product and services are evaluated, sold through M&S. (..recall Boeing 777 e.g.)
- Process improvement becomes model evaluation and improvement.

COMENTS:

May be the composability problem can be treated in a formal way through the duality between models and experimental frames. Developing frames methodology (formalization, combination with models, patterns...) could help in manipulating and reasoning about concepts within the composability context {#14}

Continuously working gathering information -->
continuously gathering, merging, and inferencing from information as system models and software develop

I really like this point {#5}

Asynchronous interaction of net worked distributed software seems to fit the "software agent" metaphor pretty well. So how much of Agent-Oriented Software Engineering could possibly be reused in this context? {#6}

As Is: Software development is based on the line of code paradigm

Future: Software development will be using large grain modules based on sensual/cognitive concepts (pictures, brain waves, etc.) {#7}

I would like to be able to generate applications on the fly using components that are distributed and can be dynamically reconfigured at run-time according to changing needs of a user. I would like an environment that allows me to design software, simulate it (e.g., simulation of the software, not simulation of the problem the software is addressing), analyze it with both formal and informal methods, and finally, generate the code. {#8}

Simulation of Software - what would be the result of such a simulation?

Would it be "how good the software is?" Its still uncertain! {#18}

particularly given the state of the art of meaningful software metrics. {#172}

Add to this, we like the distributed applications to be autonomic; they can reconfigure their components, shrink, expand their components at runtime to adopt to the changes in the environment as well as in the application state. {#13}

Vision: high reliability, high availability, robust operation (resilient to attack) {#10}

RE:SE doesn't use M&S

M&S researchers and educators have an opportunity -- now -- to identify 2 or 3 good exercises to include in software engineering labs. This should demonstrate M&S capabilities and identify problems such as terminology mismatch or representation gaps {#11}

One of the main features that M&S need to face is dealing with different abstraction level of models. So shall we put "a better way to handle abstraction" into the challenges? {#17}

and they must deal with the fact that there are inherently disjoint classes of applications {#173}

Abstraction should typically be paired with "perspective" in discussions, because the problem is not merely one of needing different levels of abstraction, but also needing DIFFERENT abstractions from one organizational or problem context to another.

Yes, learning how to deal with abstraction families is very important. However, the "grand challenge" here is recognizing that abstractions are themselves context-dependent approximations, rather than something that can be worked out once and then frozen. An implication is that those doing M&S need tools to help them create quickly meaningful abstractions for the context in which they find

themselves. In my view, it would be a mistake to attempt to do this with "only" software tools. Much better, in my view, is to have tools that emphasize man-machine interaction. For example, a user might be prompted to indicate what factors he would like to see appear in an abstraction of a model that will have 3 rather than 30 variables. The technology might test the appropriateness of the suggestions, add others, and so on. This would be in contrast, however, with the technology merely spitting out some meaningless statistical "model" as an abstraction of the previous model. {#47}

I agree with this. {#174}

We are entering an era of application development by addition to a globally networked infrastructure. There are major issues in defining how to allow access to this resource pool. One sees both malicious attacks and unanticipated loading problems in the Internet today. As more sophisticated applications are added, how can the infrastructure adapt to cope? {#19}

Software is logic. And logic is mathematics. The development of software is, therefore, limited in the same ways math is limited. Applying manufacturing principles to the development of software is like applying manufacturing principles to mathematics. That is to say, it is misguided.

This is the cause of the symptomatic frustration expressed in this challenge. {#21}

Only mathematicians may agree that logic is mathematics, etc. The more general version of this comment would probably emphasize structure, logic, systems thinking, and mathematics, not mathematics alone. That generalization would probably also have more face validity to managers and non-mathematician scientists. {#55}

The standard paradigm for major software "components" today is one of occasional releases. Since such a component (e.g., Exchange Server 2000 or Office 2003) will undoubtedly have both bugs and inappropriate behavior in any given complex system in which it is embedded, it is common for IT departments to spend a great deal of time testing a new release before installing it. That testing, however, is not particularly hi-tech, much less guided by a useful theory of comprehensiveness and assisted by high-performance tools. The result is slow introduction of new releases and, when problems with it are found, even slower "fixes" (waiting for the next release). An alternative vision might include: (1) a science and technology allowing rapid and relatively comprehensive testing of the new release (or a competitor) in the environment of interest, such as a particular enterprise; (2) immediate tailoring of the release for that particular environment where problems encountered are serious; (3) maintaining configuration control with a vast number of variant systems deployed; (4) issuing new releases that incorporate many of the individual fixes or adjustments but continue to treat others as special-purpose; and, at the same time (5) maintaining interoperability.

Wow. This would be hard to accomplish. Presumably, many "fixes" desired would have the potential to undercut standards and interoperability. Which? What protections could be built in at the time? What does interoperability mean in this context? {#22}

I have to refute the assertion that the number of systems built via composition is small. On the contrary, they are numerous if you include library routines in the class of composable components. Look at Java's class library as an excellent example - tons of stuff that we would have written ourselves only a few years ago. This is definitely ad hoc composition, but it is composition. {#29}

Often, these examples are followed by the observation that "perhaps composability is only feasible with very low-level 'components'." {#62}

Are we seeing this already with Matlab and the emergence of automatically generated code for embedded systems? This would make a nice motivating example.

Could somebody in M&S write up some simple Matlab models to go along with standard SE texts? {#23}

Yes; there are many Model-Driven Architecture tools that support both simulation and code generation. These are not trivial systems that are being generated nor are they small. {#34}

I agree with these points. In fact, I think formal methods would be the way to go if we would like to see simulation and modeling used to compose systems and evaluate various compositions on the fly. Conflicts in formal specifications of various components can be automatically detected. {#57}

How to address the requirements discussed by #13, #8, with the hard timing requirements of applications with real time constraints? There has been some advance in solving these issues, but none of the existing techniques is able to cope with all of these issues simultaneously. M&S can help in solving some of these problems, moreover to validate the desired behavior of the applications. Vision: M&S should be an integral part of the design phase, and it should help in the creation of better testing practices. {#25}

1. Formulation of challenge.

First thing, is I think that you should make this not just the software design paradigm but the *system design paradigm." If you do the design right, then the assignment of function to agent can be made at the proper time.

Secondly, there have been *formal* models of this --- notably CCS and CSP ---- that are well worked over and available as starting points. Recall Occam is a fine-

grained parallel concept that has statement level asynchronicity as a language primitive. Unity is also available. CSP has had a lot of formal development as has Unity.

Thirdly, algebraic principles are required before you can manipulate the designs. A lot of work in theoretical computer science has been put into these algebraic ideas. They should be reviewed and exploited. {#27}

The current problems and limitations facing our applications include interoperability, scalability, security, evolvability, and maintainability, etc. The current software development models address none of these issues. We need a new software development paradigm that enables us to address these challenging research problems in a way we can adopt the policies that govern them as we learn more. {#28}

I think software development is "problem-driven". Some of the aspects such as interoperability and evolvability comes into picture only for certain problem domains. Not all software needs to have such features. {#38}

I would like to see holistic approach towards software systems with persistent testing, evaluation, design evolution and adaptation during deployment. {#31}

Building composable simulation models (and therefore software-intensive systems) from high-level concepts is a challenge from technical point as well as economics and societal aspects. Given we can achieve this objective, we would need to develop in parallel concepts, policies, and tools to manage potential explosion of many alternative approaches. {#33}

This is exactly why disciplines that do use simulation - hardware, embedded systems - also do synthesis from their simulation models. {#50}

The notion of a modeling and simulation environment to create new software products and services is very appealing. However, the problem is that the task of developing the simulation may be just as hard as developing the product itself.

>> I agree with this. How do we know that the result of such a simulation is VALID? And if a software reaches a new version, how do we change the model (kind of reverse engineering) in order for it to still validate the new version of the software {#35}

(Addressing #39) Yes, but at what point in the design process do you want this level of accuracy? I would argue that at the requirements stage a representation that provides high level analysis capabilities (including simulation) is needed and that later in the process as more information is gathered, more detailed models will be needed. If we are going to take all that time to fully specify the behavior

and other properties then you lose some of the advantages of being able to do tradeoff analysis. {#53}

In response to #43, this is definitely the case, but the hardware folks were driven by the economics of failure - it's pretty hard to patch an ASIC device. Also keep in mind that hardware people synthesize devices from their models. That was the key to widespread adoption. {#58}

We cope by understanding co-evolution as a basic process. The growth of malicious and unanticipated behavior is and should be directly proportional to the growth of engineered or teleological artifacts. Admitting that development (a.k.a. design) always results in some type of co-evolutionary dynamics is the first step to coming up with a kind of meta-design method that exploits co-evolution. {#36}

We need building block software modules with accurate models for their attributes and functionalities. We need a composition algebra that enables to compose, abstract many software modules into equivalent module without losing accuracy, functionality, etc. {#39}

We will have to transition from the line of code paradigm to a "chunk" paradigm and we should encourage this sooner than later. We can learn a lot from the hardware world. In the 1960's ICs were just being developed and we used SSI. Soon after there was an evolution to MSI and the LSI and then VLSI. There was resistance in each stage, and we can expect that in software as well. Today, HW is designed using software tools, that not only design, but simulate and validate all designs before fab, similar capabilities need to be developed for software developers and they probably have to evolve in a similar manner to how hardware evolved the IC concept. {#43}

Regarding #18's comments (which I think are valid): "Simulation of Software - what would be the result of such a simulation? Would it be "how good the software is?" Its still uncertain!"

The answer to how good the software is is based on requirements. One requirement worth simulating would be performance. Can a simulation model be constructed which evaluates the performance tradeoffs of design decisions? I definitely think so. {#44}

We need to discover, compose, change software modules to build applications on the fly; we need to trivialize the software development process so any one outside computer science field can write very complex adaptive distributed applications. {#45}

Interesting! {#69}

Research in artificial intelligence has been tackling some of the core challenges that are being addressed here. Not all of the issues being raised here are being researched. For example, there does not seem to any specific focus on issues such as business aspect of the design of large software-intensive systems. {#51}

Separates structure, functions and knowledge. Defining a component software as an agent (structure) that implement a problem solving method (function) in order to achieve a particular goal (knowledge). The design of a software is then the couplage of agents that are competent in solving particular problems. Design becomes a knowledge management problem coupled with a cognitive process modeling where simulation is used in order to verify that a problem can be solve by a particular organization of agents. {#54}

Re #39 -- I don't think we need or want "accurate models", per se. What we want is a ubiquitous set of methods for generating and evaluating models. Accuracy is the effect and the cause is generating models and applying a maximum number of critical eyeballs to the models. Accuracy will emerge (and, hence, be dynamic and adaptive as theoretical understandings change). {#59}

Re #60 -- Why just modeling and simulation? Also, are you assuming that requirements specifiers are not modeling requirements? Modeling is a staple of the design process. There are all sorts of models out there (UML and the like). The real agenda here shouldn't be to introduce modeling and simulation because the fact is that modeling is already used. Simulation is definitely useful. {#72}

We need to do a better job of defining exactly where in the software lifecycle does M&S apply and make a case for it with evidence. For example - how do we use M&S techniques to get better requirements identified. How do we use M&S to move through the design, code, and test stage in a cost effective way? In what domains of software application development is M&S more likely to help? How do we address the issue of training the vast numbers of software engineers that need to be schooled in these techniques? How do we measure the benefits? {#60}

We need to investigate techniques in which the simulated code can be used like the end product code. There has been some efforts in this sense, but we still don't have a science on how to use models for simulation and for execution. Vision: we should be able to have composable components based on independent models that can be later used for end execution. {#61}

Although modeling and simulation are usually referred together, I think modeling poses more challenging than simulation. Simulation is "execution of the model", while the theory of modeling deals with system design and composition. Again, it is very crucial to make a distinct separation between models and simulators. {#75}

75 is right on the mark. Be it formal analysis or simulation, figuring out the model and what should be modeled is far more difficult than the proof or the simulation activity. {#106}

Conventional popular component-connector based approaches to software design, in my opinion, are not capable of leveraging effective modular compositional reasoning about software. To facilitate modular compositional development and reasoning, one needs to (1) use models with well-understood mathematical properties in isolation and then (2) use the connection style and configuration of the system to derive a mathematical model of the subsystem or system under development at the architecture level. With the component-connector based approaches the derived (predicted) property is the global feature of the composition. Therefore, once a change is made to the system (large complex software intensive systems constantly evolve), the analysis need to be performed from scratch. Alternatively, we need to seek different modeling abstractions, where the mathematical (behavioral) model of the system is based on the modular composition of the mathematical models of the individual components. We are pretty good in doing this from the structural point of view. Algebraic methods (as discussed in another comment) can be a fruitful direction to pursue. {#84}

Re #84 It depends on what you really want to simulate. For instance, I may be interested only in a particular quality attribute. I don't need to model all of the behaviors. Furthermore, my model can evolve as I learn more about the system. So, if I can capture some really cheap information about the system to learn about throughput for instance (with estimates on how long a component might take to do its part) then that is a lot easier and I still get the ability to do tradeoff analysis without the trouble of writing a complex specification. {#102}

Re #75 -- I don't think simulation as "execution of the model" is a complete enough definition. Simulations are effective models (in the sense that analog computers engage in effective computation and in the sense that formal systems are required to be effective). A simulation must actually DO something, whereas a model may or may not do anything. So, simulations are models. More importantly, they're useful models. Any model that is not a simulation cannot be shown to be useful. (though we all admit that non-executable models are subjectively and linguistically useful) {#88}

We only have one existence proof that any "design paradigm" could ever meet this challenge. And that is the natural living systems around us that evolution has created. No other design paradigm has shown itself to be up to this challenge. In the spirit of standing on the shoulders of giants, it seems only reasonable to use evolutionary computation as the primary "paradigm" for us to start with. {#101}

Re #45, studies of spreadsheets have shown a very large percentage (70%) have critical errors. An economist recently withdrew a result because his formula was wrong (if I remember correctly). Agreed that more end user software development

is possible and desirable but how are V&V awareness and techniques also transferred? {#139}

I will submit that Nature started with a deeply thought out architecture which has enabled the evolutionary approach. Without such an architecture, I wonder if the evolutionary approach would be feasible. {#156}

2. Challenge: Introduce M&S skill sets (+/- abstraction, +/- modeling, +/- simulation, +/- programming, +/- design) earlier and more pervasively into our educational process.

National Business Case

- Economics and value to stakeholders
- Use and extend existing infrastructure

Focus on different levels

- K-6 and high-school
- Experimentation/discovery and problem solving

Identify and develop basic concepts, suite of problems, and teaching materials that focuses on complexity and scale

- Magnet schools
- Motivate students and teachers to be interested

COMENTS:

Is this so fundamental as to pervade the entire educational process at the primary and secondary education levels? {#9}

I would submit that it should be M/D&S&A (Modeling/Design, Simulation, and Analysis) that is introduced earlier. {#12}

Young people today are absolutely compelled by video games. Some of these things are exceptionally sophisticated. Lego produces a "Brick" with an embedded JVM that 6 year olds can program using their graphical interface. What about this is not simulation? Maybe we should just be patient until these kids get through high school.

Yes, there are many such game applications. Perhaps somebody should inventory them in a M&S taxonomy, re-describe their principles (and flaws), and connect them with DLS techniques. Show that software and other engineering courses in college are extensions of simulation games from K-12. {#15}

I totally agree. {#155}

Need to have "fun" applications in the form of games, robotics to challenge the students. Also need to have mechanisms to reward and thus stimulate student participation in the form of contests leading to prizes and scholarships. {#16}

The national business case for M&S needs to be clearly articulated. This will include the importance of using M&S in health care, acquisition, etc., etc. Once you can articulate the business case then it becomes easier to discuss an extremely hard issue of inserting M&S-related concepts into the educational system (whether through the general curriculum or at specialty (magnet) schools. {#24}

Integrate into existing science talent shows, science fairs, etc. special category for creative abstract thinking, (develop a category for less scientific, but more creative thinking.) {#26}

And throw in Validation as the end goal of analysis ref#12 {#20}

Challenge in implementing this is to develop curriculum that is easily implemented by educators...given that the educational system hosts a large percentage of weak educators...not all, but the sad fact is that many of the weakest future teachers (in terms of content and their natural ability to teach) end up in the middle schools. Successfully teaching M&S skills relies on a clear understanding of the non-trivial concepts. For example, probably half of the teachers have a weak understanding of basic logic, not in its symbolic form, just its correct application. Of course, this is not to say that it is not worth pursuing, because it is essential. {#30}

I think we also need to address the issue of how do we fundamentally change the educational process so that this kind of early education is adopted institutionally and not just in a few programs. How do we train the teachers that must teach it? How do we make this pervasive? {#32}

The US has lost its edge in basic coding. If we are to excel it is essential that we utilize our biggest strength, the ability to think creatively....to lead the way in developing next generation systems...this is part of the business case. {#37}

And it might be the case (re. #37) that one should let the coding go (in the large) and move into higher end operations. If Ram is correct and coders have gone from being \$90/hr to \$20/hr, the business case for students taking coding-related courses in universities is about to be seriously undermined. {#48}

That was the point! {#67}

It needs to be clear what M&S concepts are being taught and it cannot take away from the crucial aspects of the 3Rs. The kids already model (whether through CIM generation tools or through older technologies (Lego, 3-D models of the human body), etc. Constructive and exciting challenges need to be put in front of

them. Probably, a lot of this is already in place; it just needs to insert itself into the curriculum. Mind you, parents will be concerned if their kids are playing computer games both at home and at school. {#40}

This is no doubt a challenging problem but can we afford to ignore it? What are the implications for the future generation? {#41}

In order to change the educational system you have to change the way teachers approach their job. Even enlightened teachers are constantly finding road blocks to new ideas. For example, NASA has funded several efforts to change the way teachers teach (University of Alabama, Tuscaloosa --- Nova project). At the debrief, the newly minted teachers recount horror tales of older teachers and administrators not allowing these teachers to employ proven methods.

An essential element of information in all this is we do not know any of the economics of this problem: we don't know how many M&S practitioners are needed and we don't know how big the market segment is. Therefore, you must first show that M&S is a *huge* industry kept out of sight by accounting practices.

Once we can detail the economics, we can start forward. I currently am Co-PI on a program of national dissemination at NSF that is teaching computational science and engineering to faculty in everything from anthropology to zoology. These faculty are actively engaged in preparing pre-service teachers in math, science, and technology. We have much documentation on the effectiveness of the program in K-12 and in lower division university. The issue is to get more of this out. Costs are minimal. The professional societies are the obvious place to start since they have credibility.

The teaching literature is clear: you have to get out of this testing mode and into an active learning mode. This will be a huge pill for the power base! {#42}

Should the business case include as the first requirement, an enlightened administration. This may help define how administrators are selected/promoted. {#46}

I see the current educational process as having sophisticated models of how to introduce mathematics, starting with the concept of groups and sets (of course we don't use those terms with our kids; we say "Find the blue objects) and moving to number and then symbolic manipulation using algebra. It also has a model of how to build language concepts, starting with names and moving to complex structures. I don't see the same model for problem solving using modeling and abstraction. That is the challenge in the short term - define a model for the introduction of these concepts, analogous to those for language and mathematics. {#49}

Good point. We need an M&S pedagogy that addresses various levels, audiences and contexts. {#66}

Language instruction, for example, has had several paradigms: classical grammar approach, audio-lingual method, Immersion, casual conversational oriented, and various other sophisticated approaches for teaching foreign languages in and out of academic environments. Here there are many arguments as to the best method and what the student should be able to do upon completion of a course. Ultimately, it is the student's innate ability, motivation and enthusiasm for learning that are the best indicators of future success in language instruction. I imagine that a student's future success in modeling languages/formalisms would be similar. {#107}

Not everybody is ripe for "abstract thinking". We need to not segregate non-abstract thinking "doers" as "second class" designers/developers/scientists. Some people do better on concrete problems and others on abstract problems. {#52}

Good point by #52 -- would be good if the transition from what the abstract thinker is able to do can be systematically handed off to those who can translate the abstractions into concrete realizations. {#63}

I do agree that there is a need for educational reform, but I think emphasis in mathematics is more important than moving M&S to a lower level. I think the primary obstacle to M&S education at the collegiate level is lack of interest as well as skill in math and statistics. {#64}

Are we talking about a process like the MIT LOGO system that was taught to children in the 80's/90's? If so, how successful was this experience with regards to who was taught, how have they evolved, ??? We should study this effort to determine if people taught LOGO evolved into our future leaders or if the impact of LOGO does not seem to have much correlation to future success. I don't know the answer, but my son was taught LOGO as a youngster in his school and he is definitely a potential leader in the CS field today. If we can't find a correlation, (quite possible in that LOGO may not have been taught too widely) then why did it not succeed as well as desired and can we avoid those impediments? {#70}

This problem will not be solved by entrenched adults concocting schemes to brainwash children into thinking the way the adults think. Technology is, ultimately, here because we create it. When the technology begins to engineer US, that is when we need to step back and re-evaluate how the technology came to be and whether or not we can direct it consciously.

So, I would propose that the best way to infuse next generations with M&S skills is to change those skills so that they are useful to the children... rather than changing the children so that they're useful to M&S. This means that the first step lies in digesting the output of clinically active child psychologists and changing the way M&S is done to fit that clinical data. {#118}

How many of what type of student needs to have what degree and type of exposure to M&S? If the answer is that we need a small number of superb people, then implications are very different than if one is thinking about reforming education broadly. Given the long and dismal history of educational reforms in reading, caution and empiricism is warranted. {#137}

A basic problem that I see with this challenge is that the existing work force of teachers is monumentally ill-equipped to teach anything about mathematics or modeling or simulation. Although many exceptions exist, the apparent reality is that the average IQ, average quality of education, and average "hardness" of education has dropped precipitously since most people in the room were children and youths.

This suggests, perhaps, a greater role for centralized programs that could be made available. Some analogies exist in the distant past. The PSSC physics curriculum was developed back in the 1960s by top-notch physicists and then used in many schools. It involved texts, experiments, and equipment. {#145}

*****Would it be possible for us to just try to cope with the status quo? What would we need to do then? I fear that all this talk about changing the K-12 has a low potential for implementation and impact. For instance, how long have people talked about something as simple as standardized tests (which of course is very bi-polar in how people react to it)?***** {#147}

This is an important perspective, but it may be wrong. Historically, and not just in our lifetimes, the great scientists did not have to be entertained to become enthralled with science and mathematics. They "took to it" for reasons that are probably genetic. To be sure, the next tier of scientists--excellent but not "great"--may have been "turned on" by Sputnik, science fiction or whatever. But I think that we need to understand whom we are trying to reach. If we look today at the top 10% of our high school graduates, what is the empirical reality about what makes them excited? I'm not sure what excites the bottom 50% should be of interest if the purpose is to assure American preeminence in science and technology. On the other hand, if we want a generally modern and technically literate society, then "for sure" we'd better understand how best to reach students who are not naturally motivated.

The reason for this length is that I worry about dumbing-down curricula, eliminating foundational subjects, and moving from rigor to pictures. Top-notchers will continue to need rigorous education, not just the learning of skills such as how to play computer games, even games involving simulation and including a lot of try-it-and-see problem-solving. {#152}

Comment on note 152: This is an important issue. If we focus only on the elite students, we create stratification. If we focus on all students, we create a watered down program from which few benefit. {#167}

3. Achieve truly reusable large grain software/M&S Components

Challenges:

-Replicate the existence of hardware components (memory, CPU, bus,...). What are possible software equivalents?

-Facilitate human comprehension and communication about components to enable inter-operability across corporate or military stove-pipes/perspectives

-Develop standards at higher levels that focus on the models/abstractions underlying the software rather than the software itself

COMMENTS:

Reuse currently succeeds best at high levels of abstraction. Java class libraries support the generalizations of widely used elements, such as windows, buttons etc. Reuse seems to succeed when it offers such generic support to a new level of application, being realized concretely on top of the abstract support features.

Continuing, Java classes also encompass many of the data and math structures required for more abstract programming. Ditto with STL (Software Template Library). This raises the level of abstraction of programming, removes the need for teach certain warts of programming (pointers) and frees up room in course syllabi for M&S techniques. What should go in these courses? {#68}

Note on comment 68: We should develop the equivalent of pointers at the next level. {#121}

We do replicate the existence of hardware components in class libraries. No one writes their own windowing system, mathematical libraries, or IO libraries for their application. They use available libraries. My Java book is 1/4 language description and 3/4 library description. {#71}

We will probably never be able to replicate the previous experiences found in hardware. Hardware development is based on Boolean logic, construction of basic gates that work as the building blocks of the larger scale system. This approach works good because the product is a digital computer (with a fixed deterministic behavior). Software, instead, is a dynamic entity that is constantly evolving.

The challenge here is how to use parts of the success stories in fields like hardware design into software development experiences. We need better methods for reuse, that will enable improvement of current practices. {#73}

Hardware is a poor analogy for issues in modeling, where the modules often depend on many assumptions that cannot realistically be enumerated, and where adaptation from problem to problem may be necessary (especially in rapidly changing problem areas or where human behaviors are represented).

The facilitation of comprehension and communication probably needs to be accomplished in part by visual methods, relevant and "catchy" examples, etc. Reading documentation is necessary sometimes, but not a good single image. Is there a literature of lessons learned from those who have been building help facilities for personal-computer software (which remain notoriously inadequate and annoying to many users)? {#74}

How do we categorize "large grain software/M&S components? What are the characteristics that are important? How do current work in aspect oriented reuse and other methods work? What has to be done to make this more effective? {#76}

It seems Composability and Reuse problem share a common denominator: the legitimacy of using a model (an abstraction) within a given context. One possible direction in our attempt to enhance our comprehension of composability and reuse should be oriented toward Experimental frames methodology (formalization, combination with models, patterns...) {#77}

To facilitate reuse of models assumptions that constrain the model's validity have to be made explicit. But even if we have modeling exchange formats that support the definition of constraints will a user trust that all the relevant constraints for his application have been included? {#78}

The observation is asked as a question requiring a prediction. The answer seems to be, however, "It depends." If we are assembling a model or program from some prior "components," we are far more likely to "just use" one if:, e.g., (1) we know and trust the creator or creator organization (I know Norm and when he says something is solid, it's solid); (2) the subject being treated is mature and based on settled theory; or (3) it's not important. In contrast, no one competent who is involved in serious analysis will trust a sub-model from someone unfamiliar without some degree of testing. The testing might be superficial or extensive, ad hoc or guided by a best-practices way to do testing. It might be based strictly on visually observed behaviors for some ad hoc test cases, on visually observed behaviors over a well designed range of multidimensional test cases, or something else. In extremis, there is no alternative but to read the code. That, however, often leads to one wanting to reimplement basic ideas. {#104}

Libraries and class overloading are not good examples of software reuse. Like the hardware case, the behavior of the components is pretty well known and static. The real challenge is how do we face reuse of subcomponents that need dynamic reconfiguration, modifications and updates. Your windows libraries will remain

almost unchanged, while your numerical method to update the movement of components within the window might change every time you discover a new algorithm. How do we reuse previously existing code goes way beyond standard libraries, and this is the real problem to tackle. {#79}

Reuse has worked in product line and product family approaches. The problem with reuse in the traditional sense is that developers have tried to reuse components in a context independent way. When the context is known there have been several successes. Check out, for instance, the SEI website on product line practices at: <http://www.sei.cmu.edu/plp/> {#80}

RE #80 -- Reuse of web services also seems to be gaining strength. {#83}

Another analogy, very different from that to hardware, is of organizations. Consider a football team. It has "building blocks" in the form of individual players, specialty teams, larger blocks such as offense and defensive teams. It also has a playbook of building-block operations. And, at the lowest level, it has sets of building-block primitive skills (blocking and tackling, etc.). Depending on the way the season develops, teams may do substantial rejiggering, sometimes in real time, but in doing so they almost always rely upon a different assembly of existing building blocks: they can't materialize new players out of the ether, they can't pull off complex never-practiced plays, and so on. Can this approach to designing for adaptiveness be mapped over into an approach to software design? And, if so, would some user communities be greatly assisted? For example, those interested in building systems of systems might, whereas those concerned with consumer-level products such as Microsoft Office would not be. {#85}

Reusability in more complex domains can likely be aided by having rigorous (perhaps formal) descriptions of the pertinent components. Rigor, with proper abstractions, will provide a less ambiguous description of component functionality and dependency upon environments. There has been generally quite a bit of work in the area, but not yet convergence. I was unconvinced by today's earlier discussion that reusable M&S is more difficult than reusable software. Many of the issues are the same. {#92}

It was mentioned today morning that composing models would be harder than composing code. I however disagree and think it should be easier to compose models. Models represent abstractions and algebraic or predicate logic specifications for abstract models will allow detection of conflicting constraints for two candidate models chosen for composition. However, same may not be possible at code level due to small code level details (not abstract features). {#93}

The thing is that assumptions that constrain the model's validity are not part of the model, but of the Experimental frame in which the model is being used. If you change the frame, you change the assumptions, and your model can be good or

bad for these assumptions. Recall that intrinsically, an abstraction is not neither good, nor bad. It depends on the context : the Experimental frame {#94}

The assumptions about the environment in which the model is valid should be part of the model. In fact they usually are when you specify the preconditions and postconditions of a model. {#171}

I disagree with this formulation. Bohr's model of the atom wasn't a good model subject to an experimental frame. It was a bad model, except perhaps for introducing young students to the very beginnings of quantum mechanics. Yes, one could say that it was a good model for describing the spectrum of hydrogen (?), and say that the experimental frame was hydrogen, but that would be a most unnatural use of language for all people who have studied chemistry, physics, and engineering. It is much more customary to say something such as, using a different example, "If the world were flat, then someone at an altitude of h could see forever, subject only to the quality of his eyes and the availability of light." This statement of the problem leads naturally to asking why the earth would need to be flat, how flat is flat, and so on. That is, it leads naturally to a less constrained model, of which the first is an approximation in some circumstances. Procedurally, one could say that the flat-earth model can be used if...., with the being seen as an experimental frame. {#122}

Design pattern is a good way in order to obtain reusable software. Refer to the M-V-C pattern for example (Model-View-Controller). This pattern comes from Smalltalk and is actually widely used in our tools, and is the basis of the actual Flash technology. The design process with pattern differs from the classical design process because coding becomes an intelligent "copy-&-paste" in which the aim is to write the minimum of lines of specific code.

Patterns are model of piece of codes, a sort of conceptual framework that constraints the development. The challenge is then the diffusion of the patterns in the mind of the developers. {#95}

The issues of reusability and validation belong as part of a process where we have measurable and artifacts that can be incrementally integrated. In other words, composition at the macro level may have inherently different characteristics than those at the micro-level. {#96}

To gain confidence in the reuse of a component, we need a body of experience with it. Just like the database product - which is reused in its entirety - it is built on collective experience with it and the ability to customize it. At a product level it succeeds because someone is constantly trying to make it more reusable. No such effort is invested in the piece parts that are used in our internal development - no wonder the reusability is limited.

Modeling and simulation are ways in which the confidence, experience, and capability of a reuse component can be established. {#99}

Re #80, 99 -- From what I understand about product line systems, the point is that reuse is based on confidence in a core set of common assets. The M&S would become non-factors in that case outside of analyzing the variabilities between products in a product family. {#119}

Interesting question, Horatio. The philosopher in me comes out. If we have a set of truly reusable large grained M&S components (TRC), then what does this mean in linguistic and epistemological sense. First linguistic: this means that we have a "word" for every concept that can arise in any application. Secondly, this means that we can form any semantic relationship that can arise in applications. On the epistemological side, that means that we have a logic (axioms and rules of inferences) that can be used to reason about the components and their interactions.

People in this meeting have consistently used the idea of semantics (meaning) on the assumption that this is uniformly understood. However, as discussed this morning, there is a huge problem in *received concepts* and that those concepts are universally held. In a poll of physicists, none had every heard the term "dynamical systems" even though they use those concepts. To get one (and only one) system you would have to have complete agreement so that there is no tacit (inherited) semantics. Not likely.

Far better, then, to have a system that allows for new definitions and methods for judging whether two things are the same. Also, the *pragmatics* of this system is what we teach and transmit. This pragmatics is how the formal system is used in the development of models from the introduced components. {#109}

The experimentation has a long history in sciences such as physics. We were taught to set-up a spread sheet, measure an object's weight, and answer a specific question - how much does the object weight. We were explicitly put into a frame not to think of what the object is made from, who built it, who may use it, etc. Using this basic concept can help working with reusability in a principled way. {#111}

Design patterns are successful in part because they focus on ways that match a particular set of needs. The equally (or harder) problem, therefore, is formulating the problem in such a way a pattern can be matched against it or a new one developed. Patterns and more generally frameworks are in a way some of the end products this workshop could produce. One of the objectives seems to find out if we can generate pattern's automatically -- e.g., similar to data mining and understanding how to for example do inductive (or from software engineering point of view reverse engineering) engineering. {#128}

Artificial intelligent shows that there exist a limited set of problem solving methods the human brain uses in order to solve complex problems. The model of these methods are called "Knowledge Templates". A template is a model at the

knowledge level. This is similar with design patterns which are models at the symbolic level.

The current set of "Knowledge Templates" defines a limited set of inference (i.e. logical transformation of concepts). Consequently, it is possible to define a set of software components that implements the set of inference.

This example shows that it is possible to define a minimum set of software components like hardware components, at least for knowledge systems which are a subset of the set of software. {#130}

Reuse comes from openness. Frankly, it's as simple as that. All the buzzwords and hand-waving about various ways to facilitate reuse are fine as long as they are undertaken as a kind of sociology, where the humans and their relationships are the primary focus of the effort.

Otherwise, if the goal is to invest artifacts (not just software) with properties that make it more likely to be reused, then the most basic, fundamental property is to make the artifact subject to an open system of development. The more eyeballs that can see it and the more fingers that can knead it, the more reusable it will be. {#144}

Is there some kind of theory saying that the "correctness" of two sub-systems will be consolidated after compose them together? If yes, why software components don't hold this property? If no, does it mean this is part of the nature of a general system? {#146}

Hardware analogy can be used for software reuse to a certain extent. Unfortunately, currently we are not able to develop software without inducing implicit dependencies. Such implicit dependencies are the major reason why reuse (compositional development) is not working. More specifically, substituting a new component in place of another in a large complex system often requires understanding the propagation of data and control. This is particularly the case in component-connector taxonomy, which is a common and popular software architecture representation approach. New taxonomies for architecture representation may be necessary to avoid system understanding for composition. The programming language community have a really good understanding of the notion of parameter binding as a composition mechanism. Can this metaphor be used to facilitate composition at higher level of granularity? {#150}

4. Challenge: Develop new software design paradigm that recognizes the forever-present uncertainty/emergent behaviors in ultra-large, networked, distributed, diffuse-control systems.

- Uncertainty must be part an integral part of the design
- Evolutionary approach that emphasizes exploratory of design space using simulation - observation/analysis – experimental work

- Combination of symptoms might lead to diagnosis, prediction, etc.
- How can we come up with a simulation of a software design without building it

Approach

- How to learn from natural science research to better our system design approach
- Organic system design
 - Foundations, theory, implementation
 - Self-organized, evolvable, social aspect
- Can we use how the Internet designed/evolved as a paradigm
- Use M& S to evaluate better our system designs, predict behavior, etc.
- Ontology of Building Blocks

COMENTS:

There are fundamental differences between natural science and system engineering that may make analogy difficult. It is important to observe and measure without impacting the measurements, but in computer science we create what we observe. That changes things quite a bit. {#82}

In nature evolution proceeds by random mutation and selection under adverse conditions. An organism has to be able to survive, but not all of its features are necessarily the result of selection, some are simply random. Software seems to be evolving in a similar manner. Features may remain after their original use has ceased. Some find new, unplanned uses, creating emergent behavior. We should be looking to support the mechanism of evolution, allowing diversity to flourish and adaption to be easy. {#86}

This seems to be of more interest when we consider web services and grid services and the fact that they can be developed anywhere, by anybody, for any purpose. The problem is that we run into the reuse problem from challenge #3. {#87}

Can we use how the internet evolved as a paradigm

The answer I think is no. Internet has evolved and is evolving and there is NO CONTROL.

We want to have a more controlled paradigm for software design and development. {#89}

It seems that the ever-present question is when does one want standards, and what should they be. To merely stand by and watch to see what emerges can be a recipe for disaster, as in tolerating a lackadaisical approach to computer security that has left us with highly vulnerable systems globally deployed. To impose the wrong standards can be disastrous. The "genius" of the internet, it is said, is that the standards underpinning it are spare but excellent. What makes them adequate? What makes them excellent? Are they? {#132}

#89: Perhaps a system of this size and nature simply cannot be controlled. It has to be constantly evolving to adapt to changing environments. #132: what makes them excellent and genius is the fact that they have scaled, and they have worked pretty well so far for millions and millions of devices without any centralized control. It does not make them *perfect*, I don't think we can achieve perfection in a system of this size. {#165}

RE #89 - Its seems however, that there is compelling evidence that we will have to account for the internet and its chaotic evolution model in some way because web services are out there and software developers are using them. {#124}

We need to address the concept of fusion somewhere in this effort. Fusion address the concept of uncertainty based on time, accuracy, validity, etc. concerns. As an example, a sensor provides data daily and is very accurate and at the time of capture provides high validity. But these factors deteriorate with time and a potentially less accurate, but more recent capture on another sensor may be more valid at this time. How do we develop the framework for fusion in large scale distributed environments. {#90}

Using organic systems as design models may not be such a good idea. You use the internet as an example. Was the design of the internet intentional? Specifically, were they designing support for the web when they started? Were they thinking about Napster or Google? I doubt it. Those things came about because of evolutionary change and really smart people intentionally pushing things in a particular direction. {#91}

Use M & S to evaluate better our system designs

Are we talking here HOW To or WHETHER we can using M &S. I think we are missing key point. If we were to use M &S to evaluate, where and how would be the next question in the software development cycle. {#97}

Using the Internet as a paradigm for system design: I don't think it was designed to be what it is today. It evolved this way. Nobody knew a priori how large it would become, and how long it would take to get there. When designing software with a specific goal/deadline this may not be a feasible strategy. {#98}

Commenting on 98: Yes, this is what I intended to say in my comment (see 89) {#103}

Is "uncertainty" the best term? To understand these comments, readers might need a definition (I do).

My mental concept of 'uncertainty' is something like non-determinism. There are several future paths from any current state. Users and the system itself must be prepared to deal with any of these future paths. Is that a fair paraphrase? {#108}

If we design software that is able to deal with uncertainty and emergent behavior this software should be sufficiently flexible and autonomous to be able to function in situations that could not be foreseen at the moment of its design. So does the concept of software agent subsume the kind of software we have in mind? In this case no new software paradigm would be required and for designing agents approaches exist, whose potential and limitations should be carefully analyzed before starting to look for a new software design paradigm. {#113}

Uncertainty comes in different forms. Non-determinism is one of them. One can use simulation to analyze alternative directions in the problem solution state space. Yet, this search needs to be done carefully enough to be consistent and relevant with the objectives, as well as the emergent conditions during simulation. Also it is important to notice that the goal of the simulation may not be optimization, but rather gaining intuition about possibilities and the conditions that bring them. New modeling abstractions and run-time simulation engines may be necessary to (1) achieve flexible adaptation and (2) mine simulation observed processes and data to detect emergent conditions to make a decision on relevant models that can be used to continue the experimentation. {#123}

The problem seems to be that we want to condition the evolution of these systems towards our own needs/preferences. That requires some selection process which culls inappropriate evolution. The Internet has worked very well for paedophiles and there exist few ways of forcing this kind of use into extinction. {#125}

The first thing you do is not teach current programming languages. These are von Neumann, which means that they teach this lock-step deterministic mentality machine model. My experience with CS students is they never form a complete model of computation and they are never able to deal with non-determinism. So start with non-determinism (Tm theory is just fine here although you have to change the λ -calculus rules. Stay in that part of the Chomsky hierarchy that encourages non-determinism.

Next, you don't let them ever program on only one computer...everything must be distributed from the git-go.

Now they have a physical basis to work from. Now you can have modeling exercises that are meaningful within the challenge. No student ever understands deadlock until they get into one.

Now, one more time, there are things like CCS and CSP that are algebraically complete and have a sound logical basis. Start here.

As I said this morning: are any of these words defined? Let me try to formulate an answer: Something is meaningfully defined if there is a program that implements the concept (constructivity). Map "forever present" onto this semantic base

(CCS/CSP/Unity). What are the logical foundations? That's your formulation. {#135}

The Internet evolved from simple protocols and standards. It evolved to be an infrastructure that enables an unprecedented variety of applications, yet it was not designed with any particular application in mind. I think what we can learn from it is that using standards and simplicity in the design of core protocols is a reasonable approach to building very large systems.

ABSOLUTELY {#138}

Internet has evolved over MANY years and through the efforts of MANY people. What about manpower and time involved in evolution of software, if we were to compare in that sense? {#163}

A remark: Formal logic defines uncertainty as the consequence of a contradiction. When an axiomatic theory generates a theorem t and its opposite $\text{no}(t)$, the whole set of the deduced theorem are uncertain. Consequently, the challenge is to define a design process that support contradictions. A way to do that is to consider contradictions as a generative process, not as a blocking fact. In order to go over a contradiction, we must consider antagoniste phenomenon as a whole, not as an opposition. For example, the reproduction process is grounded on two antinomic process : destruction of a cells (instability), and the copy of ARN (stability). Here two contradictory phenomenon (stability and instability) cooperates in order to maintain the live of an organism. The definition of the design as an evolutionary process leads to consider contradictions as a generative process. {#160}

When we build systems, we need to identify how we can observe them. This observation should be based on productive events, not added as a burden to them. That is what we do when checking out human health. There is some experience in diagnosing and predicting problems in software systems, but this has usually required a global view of the system. A challenge is to find non-intrusive. local indicators of impending problems and to create procedures (regimes) for system doctors to use. {#161}

Re #135 -- Exactly! Except that there are some good "current programming languages" that should be taught... Mozart/Oz, Pict (based on the pi calculus), Mercury, etc. These types of languages are helpful in coercing the minds of the young ones into a more universal understanding of "computation". {#170}

5. Challenge: How do we integrate the user interactions with the software/hardware systems into M&S?

Examples:

1) when a user browsing a website and found it busy, he/she stops the process and tries again; If a million or more users try to do the same (usually they do), the network and the web server will be flooded with unanticipated traffic.

2) In crisis scenarios, users behave in an unexpected manner that frequently crash the system although we have enough capacity; the system failed because we did not take into consideration the user interactions with the system in these conditions.

COMMENTS:

We need to address how to develop trustworthy systems, from untrustworthy components with untrustworthy actors. Actors are dynamic (users?), components are the more static components, and the systems are the integration of the components, actors and the environment. {#105}

It is too easy to say that people are unreliable. For them the problem is that the system is not robust enough to cope with their quite reasonable behavior. Systems work well if they are designed for humans. {#114}

Software engineering is concerned with allowing people to create systems for the use of people. Technical solutions are only useful if they operate effectively within the context of human need. Matching effective technical solutions to user needs is the key here and we do a very poor job of making it happen and of teaching how to approach it. We need to expand the scope of our thinking and to look beyond our technical horizon. {#110}

If a software is designed to interact with humans, then testing of the software will often also depend on models of humans and their likely reactions. However, developing human behavior models is a challenge and will often be based more on speculation than observation. {#134}

User interactions is one area where simulation is easy and viable. It is also an area that is fraught with problems and can benefit from both modeling and simulation. Imagine the type of advice a good simulator could give a designer on roundtrip time and likelihood of confusion? Technically the area is more viable and should be an area for our focus. {#151}

6. Goal: Develop an approach to speed the transfer of software/models developed in research to the user whether market place or science lab

Challenge: understand the business of software development and how it drives the industry

-Observe, research, model, and simulate business dynamics –evolution of firms

-Account for the uniqueness of IT – high profit margins, etc.

-methods to build business case for transitioning from one phase to the next

COMMENTS:

There's quite a bit known about technology transfer, adoption, impediments, etc. One lesson I have learned (probably not as well as I should) is that one should be out networking with potential clients (commercial, academic, whomever) and identifying their key pain points. If you can then develop a solution to the pain point, transfer will be much simpler. Don't allow the "ivory tower" mentality to pervade. {#112}

A couple of thoughts come to mind with this challenge - first, solid empirical evidence is needed that provides proof that a specific technique actually helps to provide a better product - better in terms of quality, assurance, performance, cost, speed of development, robustness, or other factors important to the developer or user. Without such evidence, tech transfer is unlikely. {#115}

Are there standard models in the business world now that do this effectively - Mathematical or otherwise? {#116}

I don't follow at all how modeling business dynamics will lead us to better software solutions. How many OS's have come and gone that were better than Windows? {#117}

Windows is CLEARLY not a superior operating system. Lacking a good business process that will very likely support the evolution of great technology borders on being criminal {#148}

Ah, but reality in the marketplace. {#164}

I would suggest a more limited goal of how to transfer concepts in a dynamic environment to other members of the same environment in an automated manner. Idea is to look at future systems as self modifying based on trusted patterns and trusted developments. Rapid dissemination may not be the best thing if not validated and evaluated prior to transfer. The DoD had a problem with data standards. The first made it into the repositories, better ones may have been kept out due to the fact that they were not the first. {#120}

I liked the comment that Dan Craigen made yesterday with respect to making the application of these very complex models (or methods) very easy for the practitioner. When we are able to do this in the large, more highly complex tools will be experimented with and those that show a good potential ROI will transfer. {#126}

Funding agencies like NSF and DARPA often delay transfer by setting up a chaotic, competitive initial phase of new technology development, leading to greater than necessary complexity, and longer time to remove the complexity.

Funding agencies should seek the simplest possible solutions - 27 box architecture diagrams are counter-indicators of progress {#127}

I take an orthogonal view on business process modeling. We need to understand the business processes for which we are building systems. Our answers must be good enough, which may include coming from trusted organizations. "No one ever got sacked for buying IBM" was once a truism. {#131}

My experience with an ERC with research codes is that they are the worst designed, implemented, and tested pieces of code ever devised. I observed that academics are so used to reaching for the stars that they don't understand the simplest business principles. Look at Ram's slides on the phases. I don't think any of us have the business acumen... We need help here.

In fact, I would claim that we don't understand the economics of M&S and hence can't do much in the line of marketing to begin with. {#142}

The problem with tech transfer through agencies such as DARPA is that historically, despite their statements to the contrary, tech transfer is only an add on consideration in the end. "Tech transfer is a lunch time activity" {#153}

I disagree with the premise that OS marketing is a example of business process modeling. As a onetime avid OS/2 user, I noted that IBM began shipping computers that ran Windows 3.1 and required special drivers to load OS/2 Warp. How do model THAT? I do not think that OS/2 fits Ram's model. {#158}

7. Goal: Enable managers/decision makers to understand enough about the software development process to understand when adoption of new approaches is needed.

Challenge: Develop a reference strategy that, for example:

- finds a champion manager**
- Identifies failures**
- shows how formalisms address these failures and overcome**
- generates a quick example on a laptop that shows them concrete**
- Constantly provides helpful input**
- responds to challenges**
- Uses education to diffuse new ideas**
- provide documentation at different levels tuned to manager needs**

COMMENTS:

First, from many of the comments regarding "managers" this morning, I am convinced many of us have never been managers ourselves. I think it misleading to take a "Dilbertesque" view of management. {#133}

One approach I like is the work that Vic Basili and others have done with the development of a Software Experience Factory at the Univ of MD and Fraunhofer Center. This has the potential to be a catalyst in helping managers make better decisions. {#136}

Managers operate in a world where they cannot afford to experiment for the sake of it. They are driven by targets. I once gave a course on OO techniques to a company which had not yet adopted any structured methods. In the end I suggested they adopt JSD or something similar first, to get them to a more managed state. "Good" techniques are those that deal with the current problems. {#140}

I want to agree with #133 - most that engage in manager bashing haven't been in that role. Largely I believe software managers are reasonably competent - its just that they are driven by different motivations than the technical staff that works for them. They have much more to consider than just the proper technical technique or the latest tool - Schedule, requirements drift, cost, political factors, and other pressures all contribute to the decisions they make. Poor managers do not survive in today's world. {#141}

Well put. I think the management domain is not 'dumb,' but it does have a short attention span for technical details. We need to maximize the efficiency of communication in that environment so that we convey our ideas in such a way that they can be independently verified by other technical people that managers trust. A formalism, for example, is a common communication vehicle that can be used to give the manager confidence that an approach is solid. {#143}

I think there is a lot of value in the area of lightweight formalism. I think managers have resisted the FM approach in the past for many good reasons - concerns with cost, training, value, etc. LW FM has potential - but it isn't well defined. We need a bit more work to show where LW FM makes a difference, how it is employed, what value does it add, and how does it impact cost, schedule, and delivery. {#149}

The problem is that really good engineers are quite frequently really bad managers. There really are different skill sets involved. Yet, we promote engineers into management regularly and provide no upward technical path. Managers should have technical advisers they trust as peers. {#154}

Also, really good managers may not be really good technically. Often they rely on engineers they trust to give them good advice that they translate into their management perspective and context. A formalism, lightweight or maybe just a higher level formalism with the details muted, allows the engineer to convey the ideas to the management level and to technical peers. The formalism inspires trust in the manager and allows her to independently verify the approach. The

manager needs some training in the formalism, but not to the same degree the engineer and his/her staff needs. {#169}

Sigh. This is an interdisciplinary problem, y'all. Management/administrators live in a different conceptual world. What is cut and dried to us looks like the Great Swamp to them and vice versa. A boss of mine once made the comment "you can't solve a political problem with a technical answer." The rules of proof are much different. Many folks have noted this above.

It's an education program. What it takes is folks of good will to sit down and go through it. It's an outreach problem. For example, is it SCS or SIAM's role to launch a public campaign to educate managers? Maybe. How about NSF: Maybe it's time for NSF to have a "non-technical wing" to educate the management community about what's going on and to work with those manages. {#157}

If you want non-technical managers (CEOs, etc) to take anything seriously the points must be made in the terms that of important to them: business objectives. This includes ROI, margins, market niches, whatever. Most technical folks do not have this background and, as said above, do not understand the corporate agenda. One does need to factor the business environment directly into the development/engineering cycle. If you don't, the organization won't have the market presence to survive. {#159}

Response to 149 - One definition of a lightweight formal method that is pretty good is a formal method that requires little or no training to integrate into the design path. You don't have to understand type theory to use a type checker or language theory to use a compiler. ROI is much harder to predict if there is significant training to insert something into the design lifecycle. {#162}

RE#162 - Yes, but there are many other definitions out there too. I'd like to see more definition and process to LW FM and perhaps some NSF sponsorship of that research. This area has great potential in the development world - the word "lightweight" conjures up thoughts of fast, agile, inexpensive, easily learned, ... {#166}

One thing to remember about any design organization is that they will be reluctant to change any design flow that has worked for them in the past. We can make recommendations about new flows and methods, but if they do not integrate into existing flows they will never be adopted. {#168}

Appendix 3 Record of Vote on Recommendations

Voting Results

Rank Order (Allow bypass)

Number of ballot items: 7

Total number of voters (N): 28

Rank Sum

- 162** 1. Develop new software design paradigm that recognizes the forever-present uncertainty/emergent behaviors in ultra-large, networked, distributed, diffuse-control systems
- 136** 2. Develop new software design paradigm that recognizes the inherent asynchronous nature of next generation software systems/applications operating in large scale, networked, distributed software
- 126** 3. Achieve truly reusable large grain software/M&S Components
- 111** 4. Introduce M&S skill sets (+/- abstraction, +/-modeling,+/-simulation, +/-programming, +/-design) earlier and more pervasively into our educational process
- 98** 5. How do we integrate the user interactions with the software/hardware systems into M&S?
- 81** 6. Develop an approach to speed the transfer of software/models developed in research to the user whether market place or science lab
- 70** 7. Enable managers/decision makers to understand enough about the software development process to understand when adoption of new approaches is needed

Number of Votes in Each Rating

	1	2	3	4	5
1. Develop n	10	7	6	5	0
2. Develop n	7	6	5	4	0
3. Achieve t	3	6	7	4	5
4. Introduce	6	2	4	2	5
5. How do we	1	4	2	5	8
6. Develop a	1	2	2	4	4
7. Enable ma	0	1	2	4	6
	6	7	Mean	STD	n
1. Develop n	0	0	2.21	1.13	28
2. Develop n	4	2	3.14	1.98	28
3. Achieve t	0	3	3.50	1.75	28
4. Introduce	5	4	4.04	2.17	28
5. How do we	5	3	4.50	1.67	28
6. Develop a	9	6	5.11	1.71	28
7. Enable ma	5	10	5.50	1.48	28

Group consensus (1.00 = most consensus): 0.28

Vote (10 Points)

Voting Results

10-Point Scale (Allow bypass)

Number of ballot items: 7

Total number of voters (N): 29

Mean

- 7.31** 1. Develop new software design paradigm that recognizes the forever-present uncertainty/emergent behaviors in ultra-large, networked, distributed, diffuse-control systems
- 7.28** 2. Develop new software design paradigm that recognizes the inherent asynchronous nature of next generation software systems/applications operating in large scale, networked, distributed software
- 6.62** 3. Achieve truly reusable large grain software/M&S Components
- 5.69** 4. Introduce M&S skill sets (+/- abstraction, +/- modeling, +/- simulation, +/- programming, +/- design) earlier and more pervasively into our educational process
- 5.34** 5. How do we integrate the user interactions with the software/hardware systems into M&S?
- 4.48** 6. Develop an approach to speed the transfer of software/models developed in research to the user whether market place or science lab
- 4.41** 7. Enable managers/decision makers to understand enough about the software development process to understand when adoption of new approaches is needed

Number of Votes in Each Rating

	10	9	8	7	6
1. Develop n	8	3	5	2	5
2. Develop n	6	5	6	3	3
3. Achieve t	3	2	6	4	7
4. Introduce	5	3	2	3	0
5. How do we	2	1	4	3	4
6. Develop a	0	1	2	5	3
7. Enable ma	2	2	2	1	5
	5	4	3	2	1
1. Develop n	1	2	2	1	0
2. Develop n	2	0	2	1	1
3. Achieve t	3	1	2	0	1
4. Introduce	3	3	6	2	2
5. How do we	5	2	3	2	3
6. Develop a	4	2	4	4	4
7. Enable ma	1	2	4	2	8

	Total	STD	n
1. Develop n	212	2.47	29
2. Develop n	211	2.55	29
3. Achieve t	192	2.18	29
4. Introduce	165	3.08	29
5. How do we	155	2.64	29
6. Develop a	130	2.44	29
7. Enable ma	128	3.06	29

