Thank everyone for attending and Edward for giving me a chance to set the stage.
Given that everyone here is aware of most of the things I’ll say, I know this is a bit like preaching to the choir.
Having been in at least one choir every year for the last 50, I know a good deal about choirs. And that experience leads me to my concern that sometimes the choir can unknowingly miss (or forget) things, and I’d like us to have some common ground upon which to start our discussions.
I want to briefly consider three areas: What is software@scale, what are the defense issues/problems with scale, and how are we going to approach evaluating the technologies and research areas presented over the next two days.
First, let's talk about the meaning of “scale” — and establish a context for Software@Scale. As you see, scale is a relative thing...
Obviously, we have had a great deal of difficulty over the last decade or so designing and deploying weapons and C4ISR systems that had critical software components. This difficulty often is described in terms of cost-schedule-performance failures, as that is the way the government usually measures success. The underlying issues, though, are rarely communicated to the public, including much of the research community. And it is often difficult to convince all the stakeholders to stop a death march program.
Permit me a second, somewhat longer video to make this point.
OK – I didn’t mean to hit you over the head with that analogy, but I really want to see this problem solved... Moving along, one obvious commonality is that the systems have more software, are more complex, and include unprecedented capabilities. These seemed to me like good places to start when defining scale.
Most of these systems and components were large (at least in acquisition dollars). But what does large mean in software? Traditionally, large has been measured in lines of code. It used to be that 1M LoC was large, then 10-20M. We’ve got systems now where we are trying to build 50-70M lines of defense code. This will just keep growing, of course. What about a billion? How soon will we see ultra-large-scale systems? But, this doesn’t really cover the full sense of the consequences of scale in software.
Scale also implies breadth and depth of deployment. There are systems, such as cellular phones or computer networks, where the actual number of lines of software are reasonable, but those lines are replicated in hundreds of subtle variants that produce a system that is difficult to validate, visualize and manage.
On the flip side, we often acquire a large percentage of the total lines of code as COTS or legacy. So I believe it is fair to say that simply considering the number of LoC is not a reasonable way of describing Software@Scale.
Scale may also be observed as scope. Many of our largest systems, and those most problematic from a development standpoint, are essentially smaller environments that evolved according to need rather than plan, and are now a massive set of applications that run in the same enterprise. Our friends in the defense intelligence community are just some of the folks facing this particular Software@Scale issue.
Then again there is the issue of complexity as both a result of and a contributor to the difficulties of software@scale. Of course, complexity in software has been a significant challenge since Chuck and Ada were partners. As systems get larger, or more critical, scale becomes related far more to the number of components, the number and volatility of their interactions, and the criticality of the operations.
Unprecedented functionality usually increases the technical difficulty and often the complexity. But in terms of our aforementioned typical success criteria, it makes it nearly impossible to estimate cost and schedule. So scale also seems to involve some sense of technological risk, though I think it would be hard to define. The work in understanding and dealing with complexity is just beginning, but it is surely part of the software@scale context.
Finally, and in some ways much more importantly in the real world, scale implies a significant increase in the number of humans intimately involved with a system. Not only in terms of users (or abusers), but also in the number of persons required to conceive of, fund, architect, design, build, maintain, operate, and evolve the system. And this actually is a concern about the uses or context of the system. If it radically changes the way a critical human activity (say air traffic control or warfighting doctrine), the scale of those changes adds to the already numerous risks of creating the large system.
Given these observations, I propose that when we talk about software@scale, we don’t limit ourselves to solely the traditional concept, design and development of large volumes of software. We should also address those activities that interact in any significant, constructive way with the scope (depth, breadth, and diversity), complexity (connectivity, size, unprecedentedness), or human components (within the system or in its context) in a new or existing software system. While it is important to look at the challenges to greenfield software systems at scale, given my current understanding of defense needs and acquisition priorities, those will most likely be few and far between. <Defense spending cuts> I believe that there is a great deal more leverage in finding new ways to understand, modify or simply govern the operation of our existing large software systems.

Now that we’ve examined the context for Software@Scale a bit more closely, let’s look specifically at some of the critical needs in this area.
OK – That is the defense acquisition process and we are a research group, so shoot me. I really had intended to leave acquisition out of this, but I do want to point out that the current acquisition system was designed for large batch hardware with long acquisition cycles (20-25 years from lust to dust). It is not serviceable for today’s needs unless it is bent wildly out of shape, and even then is marginal. The reason I mention this is because many of the technical problems we have are exacerbated by the acquisition process. Nuff Said!
The rapidly changing threat profile, constant adaptation to asymmetrical warfare concepts, the need to respond with new capabilities within extremely short periods of time, and the emergent nature of information-driven warfare result in adaptability as one of the most significant trends impacting defense systems. The larger a software system’s scale, the more likely it will have to be modified. Modifications may be at the edge (flexibility) or they may be serious changes within the architecture or core functionality (evolvability). Development paradigms and architectural concepts that support these will be of great value where there is the possibility to impact the architecture of the system (say in creating systems of systems or rearchitecting to include new functionality in brownfield systems). Both of these have been called out as critical capabilities by OSD executives.
Because most large systems evolve and there are so many often disparate pieces, the existing architectures are often not suitable for scaling up – particularly in the case of a successful rapidly developed quick response that suddenly needs to be deployed across a broad spectrum of environments. Evolving a common or central architecture is difficult given current tools. Rapid development of deployable and supportable software components, new architectural concepts or ways of building large systems from heterogeneous or homogeneous components are needed.
When dealing with large complex systems or systems of systems, we find that even though the software we build may be only a few 100K, most small new capabilities will share or pipeline data from other software, whether internal or external to the executing system. Understanding the impacts of adding even small components, making sure that the right data is coming and going, and that the changes we craft to deploy the new functionality won’t impact other legacy software in the system are significant challenges. The sheer number of internal and external interfaces can create nightmare scenarios for maintaining interoperability with independently evolving components and systems. New approaches to automated interface negotiation and adjustment and creative visualization techniques for these complex relationships could be helpful.
Validation and verification of large systems of systems can be an issue if the existing software our new functionality impacts is not available in our test environments. Techniques for improved modeling and simulation and the ability to track separate, concurrently evolving, interdependent systems within sufficiently accurate models are one possible answer to these concerns.
The depth, breadth and global nature of supply chains for software components causes incredible difficulty in organizing work, flowing critical technical information through the chain, and protecting IP while maintaining source control for security. New paradigms are needed to support communications within large teams of suppliers in strained “coopetition.”
The level of abstraction used in most large software systems has stayed relatively constant even as systems have grown to very large scales. As the systems grow, they become difficult to manage because the high level views are often not sufficiently informative and the low level views become incoherent in the detail. Note this “simplified” architectural diagram of the National Airspace System. Attempts to work at different levels of abstractions would seem to require new languages or modeling techniques.
As software systems are more critical and larger, they begin to impact the physical realm in sometimes unforeseen ways and so become less works of pure thought and more limited by the laws of physics. Lately these cyber-physical systems have generated great interest. They are one class of what Pennotti, Pyster and Turner have defined as interdependent systems, where most of the functionality is provided by software and most of the quality attributes depend on software. These systems require an integrated approach to systems and software engineering, because neither is sufficient in and of itself. As software scales, this interdependency grows, and techniques for reconciling the activities of these similar yet often competitive disciplines have been slow to develop.
While on the subject of quality attributes, the scale of systems increases the necessity to distribute and balance these non-functional requirements. They are success-critical, difficult to analyze, and incompletely formulated – often the definitions vary across stakeholders. These tradeoffs address competing requirements for performance, interoperability, security, safety, survivability, usability, portability, reusability, accuracy and other attributes. We need approaches to software tradeoff analysis, continuous modeling, simulation, and execution analyses of these success-critical quality factor tradeoff issues.
Finally, the larger software is, or the broader and deeper its reach, the more the collusion of complexity and criticality can undermine the security fight against cyberwarriors. However, even with the latest iPad concerns, insufficient security at the edges of very large systems may be less of a threat than the existing rot within. And we depend heavily on legacy and software of unknown provenance for building Software@Scale.
We’ve looked at context and issues with software@scale. Now we need those ideas to inform us as we look at the technologies presented in this workshop; as we learn about, pontificate on, and collaboratively judge them for their value to the defense of our nation. While not a perfect example, I thought the advancements in the longbow had a significant impact on the hundred years war and on the use of armor. This impact, with the help of luck and a muddy battlefield, supported Henry the V’s success at Agincourt, Crecy, and so on. If our goal is to find technologies and research activities relevant to what we have discussed, what should we use as the basis for judgment?
Here are the evaluation criteria we will be using today and tomorrow (as defined in the instructions):

Yes, there is a definite Heilmeier flavor to these, and I personally think that is a healthy start. All of these things need to be considered in just about any research investment. However, ...
My concern is that they don’t specifically say much about how or whether the technology will actually address the concerns of Software@Scale, or what the level of that impact might be. It is certainly possible that we may find a technology that is far beyond the state of the art, unique, useful, inexpensive, near term, with clear and derivable measures of success, but that doesn’t have any predictable impact on the problems we face in defense software@scale. I think we should celebrate those technologies, but not necessarily rank them high in the final list produced by this workshop. We should definitely keep them in our pocket for another opportunity or send them along to a friend with end-of-year money to spend. For this workshop, we need to believe our recommendations have a good chance to solve the DoD’s problems at scale.
So, I’d like to open up the discussion to you folks on what I’ve said. I may have been exhausting, but I was certainly not exhaustive. Fair game is definition of scale, additional problems or different characterizations of the ones I highlighted, and ways to measure the impact of technology. There is an impressive array of thought leaders and research professionals here, and I want to make sure right up front that all of us do everything possible to ensure the workshop produces high quality, immediately useful outcomes. Thank you again for participating, and for staying awake through this presentation - mostly.