Frictionless, Predictive Enterprise Engineering

Michael R. Mott
IBM Distinguished Engineer
195 Mtn Cty Hwy, #16
Elko, NV 89801
mottmi@us.ibm.com

Ben Amaba, PhD., PE, CPIM
Univ. of Miami and IBM Corporation
5301 BLUE LAGOON DR
Bldg 75T
Miami, FL 33126-7045
bamaba@us.ibm.com

Murray Cantor, PhD
4 & 5 Technology Park Drive
Bldg WTF4
Westford, MA 01886
IBM Distinguished Engineer
mcantor@us.ibm.com

Abstract. "It would be naive to think that the problems plaguing mankind today can be solved with means and methods which were applied or seemed to work in the past."
-Mikhail Gorbachev

This paper introduces a concept of frictionless, predictive enterprise engineering, which is based upon new standards and tools. This concept was created to meet the needs of large and complex enterprises that must adapt rapidly to changing threats and circumstances. Nowhere is this more evident, nor needed, than at the Department of Defense (DOD), which characteristically builds larger and more complicated systems than other enterprises. DOD systems development is characterized by decades of work to deliver complex, vertically integrated systems such as fighter aircraft, missiles or ships. DOD's suppliers are struggling to deliver systems on-time, within budget, and with needed capability despite decades of initiatives seeking to improve development outcomes. This paper examines the contribution that technology can make to the challenge of delivering ever more complex systems in shorter timeframes and at less cost than the historical norm.

New standards that have enabled the development of networks of collaborating systems are applied to create the frictionless interchange of information within a development organization. Frictionless information exchange among collaborating development organizations provides near real-time dashboard status to program and enterprise managers, which then enables predictive execution monitoring. Organizations will sense emerging problems sooner and will respond to them more effectively, which will provide a component for dealing with the current and emerging challenges of today's development organizations.

State of Complex Systems Development

"Our failure to master the complexity of software results in projects that are late, over budget and deficient in their stated requirements. We often call this condition the software crisis, but frankly, a malady that has carried on this long must be called normal"(Booch 2007) Booch, et.al. observe. The results for large-scale systems development within the government is no better (Kadish and et.al. 2006). A General Accounting Office Report (Anonymous2005a) found that it was common for programs to exceed budget and schedule estimates by 20 to 50 percent; further there had been no performance improvements in the ability to deliver required operational performance within budget and schedule estimates for 20 years. Obviously "normal" project performance remains problematical despite many studies and initiatives to improve these situations.

Increasingly an organization depends upon networks and networked applications to participate in the global economy. For this reason, more of the business functionality relies upon software. As an example, a CSIS study (Chao 2006) found that in 1960 8% of the functions on the F-4 fighter were controlled by software and that by 2000 80% of the F-22 fighter aircraft functions are controlled by software. Additionally systems increasingly collaborate to create new functionality or to enhance the functionality provided by stand-along systems, which increases the complexity and scope of the modern system development challenge. Thus we see a troubling trend to increasing system development challenges when the normal course of events is failure to deliver promised functionality within cost and schedule estimates.

We note that the means and methods used today were developed at a time when systems were less complex, teams were smaller, and software was smaller in size. On a small project there are few artifacts to be lost, requirements misunderstandings are easily cleared up, finding results from studies and analysis is straightforward, and the relationships among system components is easily understood. As projects grow, things change. Unless carefully managed, little problems become large delays as they multiply as the square of the size of the project. We believe this is a root cause of the infamous Fred Brooks' (Brooks 1995) termite infestation. Program calamities, Fred Brooks noted, are mostly caused by hordes of termites, each of which damage the structure a little bit, rather than a tornado that destroys the structure all at once. With apologies to Fred Brooks, among the problems that modern exterminators must address are

- Communications of quality requirements
- Automation of program information flow
- Maintenance of engineering artifacts in context
- More effective capture and reuse of assets
- Definition and delivery of development artifacts in machine consumable format

Traditional management consisted of "planning the work and working the plan." Although systems of astonishing complexity have been built with this approach, it is clear that as systems have increased in complexity the approach isn't working as well as it once did. This suggests that the scale of today's programs is such that this "plan and track" program management approach isn't scaling well.

Programs captured detailed plans, milestones, dependencies and deliverables. In order to manage the program work packages are developed with a week or two in duration. For a 5-year

program with work packages of 2-4 weeks in duration paths through the program will involve 50-100 activates to complete. There will be 100s or even 1000s of these paths through a large development program. Each path must be completed within some specified period of time or the program will experience a delay. The paths with the least margin are called critical paths. If a program thread consisted of 50 steps and each had a probability of completion on time of .99 then that thread would have a probability of an on-time finish of .6. With a probability of .9 for each step the thread has a probability of .006 of finishing on time. Seldom will the steps have probability of .9 and often much less. Clearly as the program scales up it is not possible to create a detailed plan that will be executed. Therefore the plan for such a program is update continually.

Comprehensive techniques (c.f. (Anonymous2007; Anonymous2005b; Anonymous2005c) have been developed to plan and track the development of systems. Earned Value Management Systems (EVMS) are used to determine to in progress program status by measuring actual task completions against planned task completions. When a program reaches a level of complexity at which creating a detailed plan is impractical then EVMS requires a nearly continuous update effort to maintain the status. At major reviews, such as critical design reviews, the technical, cost, schedule and risk baselines are synchronized and in-depth reviews are conducted. At other times it is difficult to determine the exact status of a program. When the program is part of a portfolio of many such programs the use of these methods are impractical for managing the large portfolio[1].

As an alternative this paper explores technology that will enable a sense and respond (Haeckel 1999) approach to managing a development organization. Participating organizations, which often are under contract, are “instrumented”. This instrumentation results in a notification of a change in its status (cost, schedule, or technical content). These notifications are delivered using publish and subscribe, or other similar ones, services to organizations dependent upon their deliveries. Organizations receiving such notifications update their status and if it changes, notify of that change. In this way near dashboard visibility into the status of a program can be created.

Round Trip Engineering

The Unified Modeling Language (UML) standard incorporated a concept of round-trip engineering (Fowler 2004) (figure 1), which is used to validate that the software specified is actually the software that is developed. The software development team follows a process for gathering requirements and then expresses a design solution with UML diagrams. The semantics of these diagrams allow them to be transformed into a language, such as c++ or java, which is then compiled and linked to produce the application. Once an application is developed, how can the SW team validate that it is correct?

Traditionally, application development teams created a suite of test scenarios and these were run against the application. Large applications contained very large state spaces and thus it was impractical to test the application exhaustively. This led to the concept of test coverage, which examined the possible paths through the code and selected a subset that could reasonably test the state space given a time and budget to accomplish the task. This subset coverage, which potentially leaves disastrous problems in the application, was problematical and thus alternatives

[1] Dr. Anne Sandal, who is the Program Executive Officer for Littoral and Mine Warfare in the Department of Navy, observed in a meeting that she receives a stack of EVMS reports several inches high daily and with all the detail was unable to determine the status of her portfolio.

to this problem were sought (Mills, 1993).

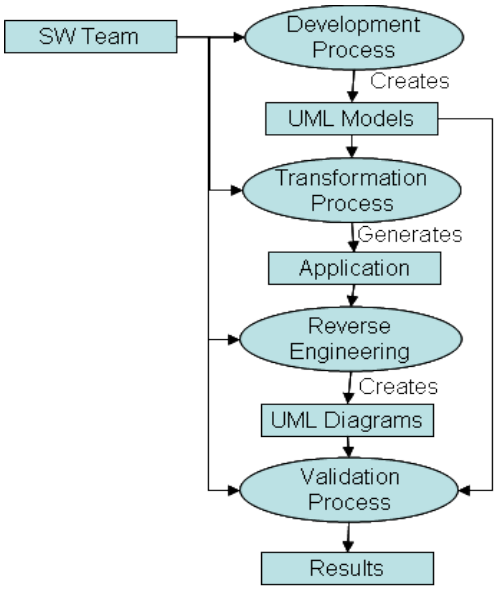

Figure 1 - Reverse Engineering Validates Correct Implementation

Reverse engineering is one alternative to validate a system is operating as intended. To use this approach a development team runs a reverse engineering tool over the application source code to create a set of UML Diagrams. These reverse engineered diagrams are compared with the diagrams that were used to generate the software, and when they match then we can say that the software that was built will function as intended. Of course the original diagrams must have traceability of some form with the original system requirements to ensure that was built correctly was also what was intended.

Similarly the system engineering process (figure 2) determines that a system under development closes, which means that the development program plans will deliver the needed content within the budget and schedule estimates. Periodic reviews of the development status are conducted in which a review of the requirements, the technical baseline, cost and schedule, and also risk is conducted (figure 3). For the technical baseline the adherence to constraints or budgets, such as weight and power, and also the capability delivery is evaluated. Among the means of expressing capability are Key Performance Parameters (KPPs), Key Performance Indicators (KPIs), Measures of Effectiveness (MOEs), and Measures of Performance (MOPs). The programs adherence to cost and schedule goals is sometimes captured with an Earned Value Measurement System (EVMS) in which Cost and Schedule Performance Indicators (CPI and SPI, respectively) are calculated.

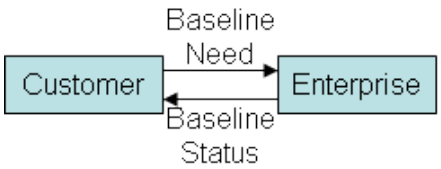

Figure 2 - Enterprise Status Assessment

Gathering up a consistent, top-to-bottom picture of the baseline status requires a round-trip assessment of the program throughout the enterprise hierarchy. Baseline requirements are

checked, changes are noted, and the validated requirement set is passed down through the hierarchy. Each organizational entity reviews the conformance of its design to requirements and budgets and passes back its technical baseline, expected cost to deliver, expected schedule to deliver and the risks associated with that work. At each level on the return trip, a supplier or subcontractor will roll up the schedule, cost, and risk estimates of its suppliers. Schedule dependencies are checked and from this effort a consistent picture of the program emerges.

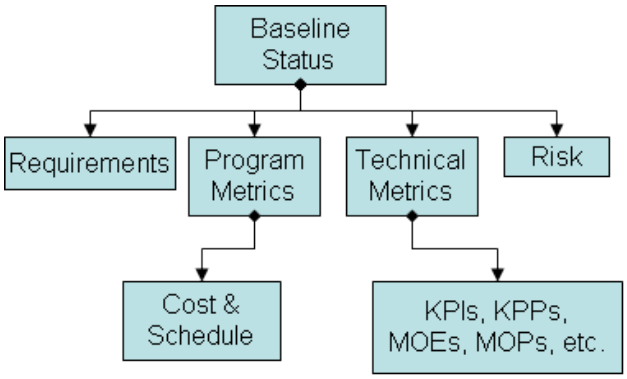

Figure 3 - Baseline Status Content

Today's enterprises must become agile to an unprecedented degree. Responsive and purposeful response to change in the competitive landscape, the threats faced, the operational environment, or the technology used is essential. The time to respond to major changes is days and weeks rather than months and years. In a large enterprise this round-trip engineering process, whether to assess the status or to update the baseline, requires months to complete with the commonly used methods and tools. This is not acceptable.

In line with Gorbachev's caution we have reached a point at which our current means and methods are asked to do more than they can reasonably do. We need new means and methods to support the extended and increasingly complex enterprises of tomorrow.

Representational State Transfer (REST) Architecture

Representational state transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. As such, it is not strictly a method for building "web services". The terms "representational state transfer" and "REST" were introduced in 2000 in the doctoral dissertation of Roy Fielding (Fielding 2000).

REST strictly refers to a collection of network architecture principles, which outline how resources are defined and addressed. The term is often used in a looser sense to describe any simple interface, which transmits domain-specific data over HTTP without an additional messaging layer such as SOAP or session tracking via HTTP cookies. These two meanings can conflict as well as overlap. It is possible to design a software system in accordance with Fielding's REST architectural style without using HTTP and without interacting with the World Wide Web.[2] It is also possible to design simple XML+HTTP interfaces which do not conform to REST principles, and instead follow a model of remote procedure call. The difference between the uses of the term "REST" therefore causes some confusion in technical discussions. Systems, which follow Fielding's REST principles are often referred to as "RESTful". (Reference

Wikipedia)

REST is a coordinated set of architectural constraints that attempts to minimize latency and network communication while at the same time maximizing the independence and scalability of component implementations. This is achieved by placing constraints on connector semantics where other styles have focused on component semantics. REST enables the caching and reuse of interactions, dynamic substitutability of components, and processing of actions by intermediaries, thereby meeting the needs of an Internet-scale distributed hypermedia system.

The following contributions to the field of Information and Computer Science have been made as part of Roy Fielding's dissertation: a framework for understanding software architecture through architectural styles, including a consistent set of terminology for describing software architecture; a classification of architectural styles for network-based application software by the architectural properties they would induce when applied to the architecture for a distributed hypermedia system; REST, a novel architectural style for distributed hypermedia systems; and, application and evaluation of the REST architectural style in the design and deployment of the architecture for the modern World Wide Web.

The modern Web is one instance of a RESTful-style architecture. Although Web-based applications can include access to other styles of interaction, the central focus of its protocol and performance concerns is distributed hypermedia. REST elaborates only those portions of the architecture that are considered essential for Internet-scale distributed hypermedia interaction. Areas for improvement of the Web architecture can be seen where existing protocols fail to express all of the potential semantics for component interaction, and where the details of syntax can be replaced with more efficient forms without changing the architecture capabilities. Likewise, proposed extensions can be compared to REST to see if they fit within the architecture; if not, it is more efficient to redirect that functionality to a system running in parallel with a more applicable architectural style.

In an ideal world, the implementation of a software system would exactly match its design. Some features of the modern Web architecture do correspond exactly to their design criteria in REST, such as the use of URI [T. Berners-Lee, R. T. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic syntax. Internet RFC 2396, Aug. 1998.] as resource identifiers and the use of Internet media types [N. Freed, J. Klensin, and J. Postel. Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures. Internet RFC 2048, Nov. 1996.] to identify representation data formats. However, there are also some aspects of the modern Web protocols that exist in spite of the architectural design, due to legacy experiments that failed (but must be retained for backwards compatibility) and extensions deployed by developers unaware of the architectural style. REST provides a model not only for the development and evaluation of new features, but also for the identification and understanding of broken features.

The World Wide Web is arguably the world's largest distributed application. Understanding the key architectural principles underlying the Web can help explain its technical success and may lead to improvements in other distributed applications, particularly those that are amenable to the same or similar methods of interaction. REST contributes both the rationale behind the modern Web's software architecture and a significant lesson in how software engineering principles can be systematically applied in the design and evaluation of a real software system.

For network-based applications, system performance is dominated by network communication. For a distributed hypermedia system, component interactions consist of large-grain data transfers

rather than computation-intensive tasks. The REST style was developed in response to those needs. Its focus upon the generic connector interface of resources and representations has enabled intermediate processing, caching, and substitutability of components, which in turn has allowed Web-based applications to scale from 100,000 requests/day in 1994 to 600,000,000 requests/day in 1999.

The REST architectural style has been validated through six years of development of the HTTP/1.0 (T. Berners-Lee, et..1996)] and HTTP/1.1 (R. T. Fielding, et.al, 1999) standards, elaboration of the URI (T. Berners-Lee, et. Al., 1998) and relative URL (R. T. Fielding, 1995) standards, and successful deployment of several dozen independently developed, commercial-grade software systems within the modern Web architecture. It has served as both a model for design guidance and as an acid test for architectural extensions to the Web protocols. (Fielding, 2000).

Towards Frictionless, Predictive Enterprise Engineering

The Internet has changed the world by enabling new businesses models, such as eBay and amazon.com. Traditional retail stores, such as Wal-Mart, use Internet services to improve their business results. All of this is enabled by the frictionless[2] flow of information between collaborating entities. The free flow of information, enabled by REST and other architectures, also enables new and more predictive approaches to managing the enterprise. An example of this principle is just-in-time inventory management. Networks provide suppliers with visibility into the inventory velocity of the clients so that they can predict when it is time to restock. This minimizes the size of the inventory and thus again reduces the cost of the business.

As Internet capabilities are enabling systems to reach unprecedented complexity, scale and scope it is reasonable to examine the way in which these frictionless and predictive properties can be applied to tackle the challenges of large-scale development organizations.

Commit, management and engineering are among the formal, or informal, protocols that bind enterprise scale development organizations (Figure 4). The enterprise can be thought of as a network of commitments. Commitments (see for instance (Haeckel 1999)) begin when a client expresses a need to an enterprise or when an enterprise recognizes a need for additional capability because of changes in the competitive landscape. Enterprise solutions are created by a network of commit protocol transactions in which a consumer (or prime) asks a subcontractor (or supplier) for goods or services. In the case of either a good or a service the subcontractor (or supplier) expends resources in order to produce a deliverable[3]. That deliverable may be a tangible good like a stereo or an airplane wing or it may be the result of a process, such as hiring a new employee if the commitment was to provide human resources services. With either goods or services, there is a specification as to what needs to be done, an offering to perform at a price and schedule, and an agreement (contract or not) to do it. The description of the good or service is sufficient to determine whether, or not, the deliverable is satisfactory.).

---

[2] One of the author's (Mott) saw a panel discussion in which Dr. Bill Joy mentioned that copyright laws were antiquated by the Internet. He reasoned that a certain friction to the copy and distribution of physical media was removed by the Internet.

[3] Steve Haeckel in private communications recommended this formulation to Mott. It is quite general and also consistent with the concept that a purposeful system exists to create a state change in the environment. That state change may result from services or it may involve delivery of a tangible entity. In either case resources are consumed to create the state change.

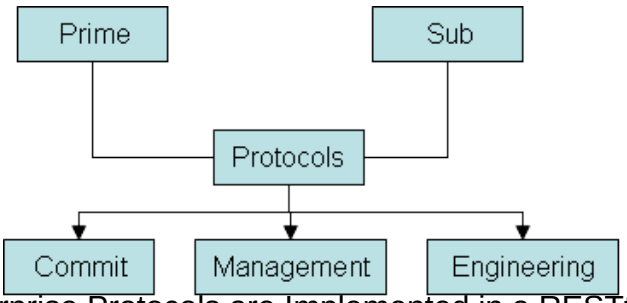

Figure 4 - Enterprise Protocols are Implemented in a RESTful Architecture

The deliverable (figure 5), which is created by an agreement reached through a commit protocol, is a response to a set of requirements. A part of the agreement between the parties is a schedule and cost for delivery of the good, or service, that satisfies the commitments. Management tracking the progress of a set of deliverables will be interested in the risks that the deliverable will exceed its budget, will be delivered late, or will fail to meet its specifications. The baseline thus includes the variances in estimation and associates these variances with the risk that the deliverable will miss one or more of its program parameters (Cantor 2006, Sep 23, 2008). For the enterprise the new deliverable class, which is based upon the RESTful architecture, deals with the internet scale issues associated with passing and aggregating status information throughout the enterprise.

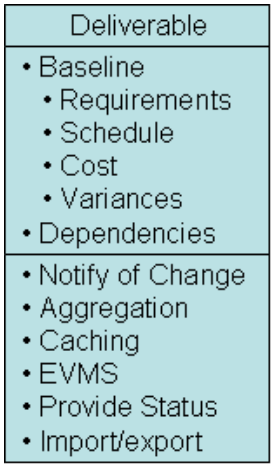

Figure 5 - Deliverable Component for RESTful Enterprise Engineering Architecture

The deliverable enables an enterprise to add new capabilities and participants in a plug ‘n play fashion. It is intuitively appealing that an enterprise that seeks to add capability through plug ‘n play components should itself be capable of easily inserting new participants into the engineering effort. The deliverable enables this capability as its mechanism supports dependencies and prime-sub relationships. To insert a new business, company, or organization into the enterprise requires the following set of actions:

- Instantiate the deliverable class for the new enterprise entity
- Define the prime-sub and dependency relationships
- As necessary create adapters to import the existing data sources of the new entity into the enterprise
- Normalize the deliverable’s information for the needs of the enterprise

- Define the roles and access control methods for the deliverable
- As needed update the services that the deliverable performs

When an organizational entity misses a milestone then the impact of that is migrated throughout the enterprise with the deliverable mechanism. We note that the authority to update the status of a deliverable is part of the governance work that the enterprise. Successful use of the deliverable mechanism requires that the enterprise establish and follow sound governance mechanisms. All of the deliverables status, including schedule, are maintained by this class. When it is determined that the schedule of the deliverable is slipping then that is updated, in accordance with governance policies. The revised schedule is cached so that it can provide its current status upon request. The deliverable also notifies the deliverable classes for its prime and also other entities dependent upon its delivery (figure 6).

The prime and dependent entities update their deliverable schedules with the latest information, update their caches, and notify their primes and the organizations dependent upon their delivery. This process continues until the status of all impacted deliverables is brought up to date. This distributed mechanism for maintaining the status of the enterprise is consistent with the web approach. The status is maintained at its source and propagated via web mechanisms. This is similar to the way status is maintained for e-business. After a customer buys an item at a web storefront the shipping and also tracking for the shipping is handed off to that service provider. Status is maintained as a service by the appropriate service provider, rather than passed to the store front and then onto the client.

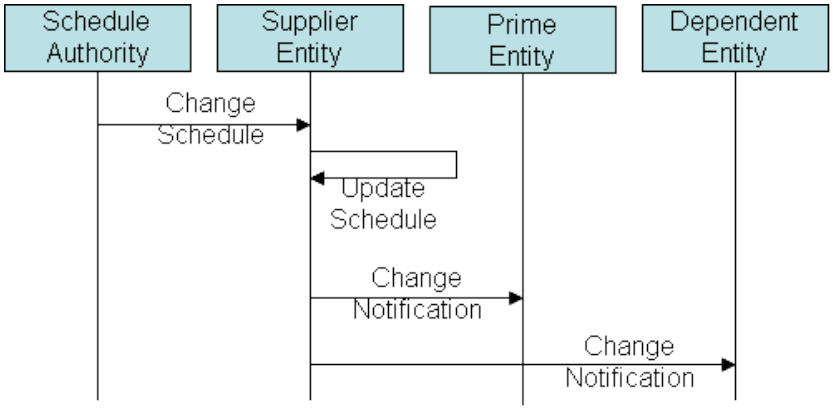

Figure 6 - Schedule Update Notifications

A deliverable class apparatus can create a more responsive and robust round-trip engineering mechanism. The enterprise, one or more of its clients, or a sponsor may request a new or improved capability. When that happens the enterprise opens negotiations with its suppliers via the commit protocol. If appropriate the enterprise can open a new version of the system as part of its request in order to maintain and manage the expected baseline change. A solicitation is sent to their suppliers and negotiations are initiated. Each of their suppliers will analyze the change request and as needed suballocate to a supplier or a prime. Thus the negotiations cascades through the organizational network and the transactions are supported by the deliverable class. New deliverables can be added; existing deliverables can be modified or even cancelled. As the agreements are reached, the deliverable classes work together to create a rollup of the new technical baseline. If used the versioning mechanism can provide dashboard status of the changes as they cascade down and roll back up through the organization. If the

change is approved the enterprise can immediately begin to execute to the new baseline. Therefore a deliverable class supported by a RESTful architecture provides a means of providing internet scale round trip engineering.

As seen in other applications of a RESTful architecture this level of visibility provides new capability for managing the enterprise. An example is just-in-time inventory and value chains. With state changes available to the enterprise the rate at which material and merchandise flows through the value chain can be measured and used to predict when more inventory will be needed. The flow of the inventory can be quickly adapted to the needs of the enterprise, which in turn reduces cost and creates more value. Similarly with the capability to assess the impact of schedule changes due to changing component delivery schedules enables the enterprise to adjust its schedules to compensate in response. The capability to predict impact and adjust schedules and costs accordingly promises to enable agility and cost efficiency to a degree not possible with current methods.

Conclusions

That development organizations are struggling to deliver needed content within cost and schedule estimates is evidence of the challenge of large-scale development. With this status this isn't, as Booch observed, a crisis but is business as usual. Development organizations are struggling to meet current system requirements while demand for more complex systems, which is driven by the capability of the web, is increasing. It is evident that new means and methods for successfully managing the development activities of large enterprises are required. This paper presents a conceptual framework that trasforms development from a plan and track approach to a sense and response one.

A deliverable class is posited to enable a frictionless flow of engineering and program data throughout the enterprise. Changes to the status of the delivery of any component will cascade through the development organization and a new status is thus created in near real-time. This class creates the possibility of providing dashboard visibility into the development of new enterprise capabilities. Further, the methods provided by the class can be tailored to support a wide variety of deployment environments.

New networking technologies, based upon the RESTful architecture, are increasing the complexity of systems at a time when development organizations are struggling with existing levels of complexity. This paper presents a conceptual framework that illustrates that the technology that is increasing the demand for more complex systems can contribute to the development of new engineering tools and methods that enable development teams to successfully meet the emerging demands.

References

*Earned value management systems - program guide*2007.

DEFENSE ACQUISITIONS: Assessments of selected major weapon programs. 2005a. Washington DC: General Accounting Office, GAO-05-301.

*Integrated master plan and integrated master schedule preparation and use guide*2005b. DOD.

*MIL-HDBK-881A work breakdown structures for materiel items*2005c. DOD.

Berners-Lee, T., Fielding, R.T. , and Nielsen, H.F.. Hypertext Transfer Protocol -- HTTP/1.0. Internet RFC 1945, May 1996.

Berners-Lee, T., Fielding, R.T. and Masinter, L.. Uniform Resource Identifiers (URI): Generic syntax. Internet RFC 2396, Aug. 1998

Booch, Grady, Dr. 2007. *Object-oriented analysis and design with applications*. The addison-wesley object technology series. 3rd ed. Upper Saddle River, NJ: Addison-Wesley.

Brooks, Fred, Dr., ed. 1995. *The mythical man-month*. 2nd ed. Reading, MA: Addison Wesley Longman Inc.

Cantor, Murray, Dr. 2006. Estimation variance and governance. *Rational DeveloperWroks* (March 15,2006), http://www.ibm.com/developerworks/rational/library/mar06/cantor/.

Chao, P. 2006. *An assessment of the national security software industrial base.* Center for Strategic and International Studies, , http://www.csis.org/media/csis/pubs/061019_softwareindustrialbase.pdf.

Fielding., R. T. Relative Uniform Resource Locators. Internet RFC 1808, June 1995

Fielding, R.T., Gettys, J., Mogul, J.C., Nielsen, H.F., Masinter, L., Leach, P., and Berners-Lee, T.. Hypertext Transfer Protocol -- HTTP/1.1. Internet RFC 2616, June 1999. [Obsoletes RFC 2068, Jan. 1997.]

Fowler, M., ed. 2004. *UML distilled*. 3rd ed. Boston, MA: .

Fielding, R.T., UNIVERSITY OF CALIFORNIA, IRVINE, Architectural Styles and the Design of Network-based Software Architectures DISSERTATION, submitted in partial satisfaction of the requirements for the degree of DOCTOR OF PHILOSOPHY in Information and Computer Science, 2000

Haeckel, Stephan H. 1999. *Adaptive enterprise : Creating and leading sense-and-respond organizations*. Boston: Harvard Business School Press.

Kadish, R. T., and et.al. 2006. *Defense acquisition assessment report.* , http://www.acq.osd.mil/dapaproject/documents/DAPA-Report-web/DAPA-Report-web-feb21.pdf (accessed 04-07-2006).

Mills, H., Zero Defect Software: Cleanroom Engineering, Advances in Computers, Volume 36, p.16, Yovits, 1993

BIOGRAPHY

Mr. Mott holds a MS in Computer Information Systems from the Univ of Phoenix and a BS in Applied Mathematics from the Univ of Wisconsin-Stout. While at Boeing Mr. Mott changed the way large and complex satellite ground stations are developed; upon retirement Mr. Mott came to IBM where he has advanced to the position of IBM Distinguished Engineer. Mr. Mott was one of four authors of an IBM Systems Journal article "Model Driven Systems Development" that described a single, tailorable systems and software engineering framework. Mr. Mott currently supports efforts to develop engineering means and methods to support system of systems engineering.

Dr. Amaba. Holds a PhD in Industrial & Systems Engineering, a M.B.A./M.S. degree in

Engineering Management, and a B.S. degree in Electrical Engineering. Registered Professional Engineer, certified in Production and Inventory. In addition he holds number certifications including Registered Professional Engineer, certified in Production and Inventory; Management by APICS ©, LEED © AP; Leadership in Energy & Environmental Design Accredited Professional, and certified in Corporate Strategy by Massachusetts Institute of Technology ©. Dr. Amaba focuses on complex, large systems design and development in Public and Industrial Sector with an emphasis on Energy and Environmental design for the Smarter Planet.

Dr. Murray Cantor's areas of expertise include software and system engineering processes, and system development management and leadership. Cantor is the lead architect of RUP SE, the extension of the Rational Unified Process for system and enterprise engineering. He also develops assets to support development organization productivity improvement. Cantor learned the trade of system development by taking leadership roles in a variety of projects at IBM and TASC. Cantor has been a system architect, team lead, project manager, development product manager, architecture manager, and program manager.

SUBMISSION

Your **review paper** should be submitted electronically via the on-line Submission Form. The completion of this form provides the necessary information needed for the review process. If you have any difficulties with this submission, please contact ASK International Conference

Your **final paper** should be submitted by e-mail as an attachment to ASK International Conference at: is10_submissions@askintlconference.com. ASK International Conference will acknowledge receipt of your final paper within five days. If you do not receive acknowledgement, please contact:

Contact Information

ASK International Conference, 2 chemin de la Serre, 31450 BELBERAUD (France) ; Tel: +33 (0)5 6135 3108 ; Fax : +33 (0)5 6769 9015 ; Email : is10_submissions@askintlconference.com