Edge Enabled Systems

Zacharie Hall
Command and Control Directorate
Communications-Electronics
Research Development and Engineering Center
Aberdeen Proving Ground, USA
zacharie.t.hall@us.army.mil

Rick Kazman, Daniel Plakosh, Joseph Giampapa, Kurt Wallnau
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA USA
{rkazman, dplakosh, garof, kwallnau} @sei.cmu.edu

***Abstract*—Users of today have ever-increasing levels of technical skill with computing and communication technologies. For example, on the battlefield, some soldiers are capable of creating or modifying existing systems in response to needs that were not anticipated by the designers of the original systems. In a growing number of situations this ability is crucial, because the soldier must be able to adapt rapidly to a dynamically changing operating environment; thus the software must also be adaptable. Software architectures and software development methods must be created that enable user innovation "at the edge" so that users can be as effective as possible in the face of changing missions and unanticipated needs. In this paper, we describe the characteristics of edge systems and the edge organizations in which these systems operate, and make initial recommendations about how such systems and organizations can be created to serve the needs of users at the edge.**

Keywords-edge programming; crowdsourcing; agility; adaptability; edge enabled; open source; ultra large scale systems; socal systems; architecture; community based software

I. INTRODUCTION

Traditionally, military software has been designed to discourage the end user from modifying the application, to ensure application integrity, security, reliability and consistency. However, today's Warfighter relies more than ever on information technology; this means that they must be adept at using technology and to rapidly adapt to changes in their environment. Today's Warfighters have substantial and ever-increasing levels of technical skills. In the battlefield, some soldiers are capable of creating or modifying systems in response to needs that were not anticipated by the original designers of the systems. Thus software that the Warfighter uses must also be adaptable. Software architectures and development methods must be defined or developed to help enable user innovation "at the edge" so that Warfighters can be effective as possible in the face of changing missions and unanticipated needs [1].

These methods of facilitating the user at the edge have been referred to as "Edge Programming"[23], but what is actually required is much broader than just programming: edge activity does not necessarily involve writing "code" and may also involve substantial changes in organizational structure and governance [1]. In this paper we use the term "Edge Enabled Systems" (EESs) to emphasize our interest in systems that enable the creative contribution of users at the edge. In the remainder of this paper we will describe the principles on which EESs are grounded, and will describe the architectural and process principles for creating and supporting EESs.

A. Enabling the Edge

Yochai Benkler in his book "The Wealth of Networks" [4] puts forth a compelling argument: we are in the midst of a radical transformation in how we create our information environments. This change underlies the open-source movement in software, but open source is just one example of how society is restructuring around new models of production and consumption. The most impressive aspect *"is the rise of effective, large-scale cooperative efforts—peer production of information, knowledge, and culture... We are beginning to see the expansion of this model not only to our core software platforms, but beyond them into every domain of information and cultural production."* [4]

Our networked information environment has transformed the marketplace, creating new opportunities for how we make and exchange information. "Crowdsourcing"—one form of value creation at "the edge"—is already widely used in the arts, in basic research, and in retail business [7]. The world has changed: increasing the role of non-market and non-proprietary production and the role of individuals and loosely affiliated groups, while reducing the power of big business.

Another trend changing our world is that businesses are themselves transforming, in part as a reaction to net-enabled consumers; firms are moving towards service orientation. Service industries account for over 55% of economic activity in the United States [5]. According to Vargo, businesses have moved from *"a goods-dominant view, in which tangible output and discrete transactions were central, to a service-dominant view, in which intangibility, exchange processes, and relationships are central."* [24] Service-dominant logic requires a fundamental shift on the part of businesses, to see consumers not as passive recipients of goods, but as co-creators of value. Inviting and enabling the crowds helps to align systems with the real and rapidly changing needs of their users. In a service-dominant view of the Army, Warfighters are therefore not treated as passive receivers of information, but as

co-creators of information and the value that attends this information. In Section 2 we will show some examples where co-creation is already occurring in the Army.

Vargo claims that the shift from a product/goods focus to a service focus actually entails several shifts in thinking [24]:

1. From thinking about the purpose of firm activity as making something (goods or services) to a process of assisting customers in their own value-creation processes.
2. From thinking about value as something produced and sold to thinking about value as something co-created with the customer and other value-creation partners.
3. From thinking of customers as isolated entities to understanding them in the context of their own networks.
4. From thinking of firm resources primarily as operand—tangible resources such as natural resources—to operant—usually intangible resources such as knowledge and skills.
5. From thinking of customers as targets to thinking of customers as resources.
6. From making efficiency primary to increasing efficiency through effectiveness.

Collectively, these shifts imply more than just a move from goods to services. They are a reframing of the purpose of the enterprise and its role in value creation, for both the entities involved and for society. So our research goal is to determine what an organization (such as the US Army) must do to use and to manage projects in a service-dominant world? How can we create the service systems [15] (or, in our terms, the Edge Enabled Systems) of the future?

These are difficult questions for which existing system and software development models—waterfall, agile, spiral, and so forth—are of little help. These older models all contains "closed world" assumptions: projects have dedicated finite resources, management can "manage" these resources, requirements can be known, software is developed, tested, and released in planned increments. But these assumptions all break—to varying degrees—in a crowdsourced world, where most of the value is created at the "edge".

Traditionally, system analysts have been trained to focus on the "value propositions" of firms instead of "value co-creation." At best, "co-production" with stakeholders on the edge has been considered in design methodologies such as Joint Product Design, Joint Application Design, Rapid Application Development and, more recently, agile methods in which customers' requirements are solicited and modeled through an iterative process that incorporates intense customer feed-back. Examples of projects that have had intensive stakeholder input include the Command Post of the Future (CPoF), Tactical Ground Reporting (TiGR), Combined Information Data Network Exchange (CIDNE), FusionNet and FalconView.

While innovative in many ways, each of these still exhibits a goods-dominant logic perspective. Product-focused and goods-focused design treats customers as isolated entities—as *recipients* of value—and neglects customers' own resources and networks for dynamic collaborative value co-creation. Service-dominant design, on the other hand, considers resource integration from various entities (users, firms, suppliers, and their networks) for value co-creation. Examples of such co-creation have already emerged, from open-source software to Wikipedia, Facebook, Amazon's Mechanical Turk, and many other community-based service systems. Each of these examples is a complex software-intensive or software-enabled system that is co-created by its participants—the crowds.

This paper characterizes edge-enabled systems from two perspectives: first as an organizational construct that contains what we term a "Metropolis" orientation, then as a collection of mechanisms to create and manage edge systems. Building on these two perspectives we provide a set of recommendations for changes in the way that the Army (or any other organization that creates complex life-critical systems) designs, builds, and manages complex systems. This paper does not suggest that Edge enablement is right for all systems—certainly there are and will always be some classes of system that need to be tightly controlled and rigorously developed—but we maintain that a large and important class of systems can be profitably created with an eye to engaging the Edge.

II. Edge Enabled Systems in the DoD

A. *Edge Enabled Systems Today*

One of the oldest examples of Edge enablement, in military systems, is the US Navy's AEGIS Weapon System (AWS) see Figure 1. Excerpts from [20] provide clues to the way that Edge Enabling, in somewhat constrained form, is used in high-end systems today:

Figure 1: Aegis Weapon System Information

"*AWS automates many functions in the ship's operations room such as picture compilation, tracking, identification, target-weapons pairing, 'quick reaction' or 'late detect' procedures and tactical data link management. These*

automated functions reside within 'AWS Doctrine'; a set of standard operating procedures that allow the ship's command to adapt to changing operational situations with a series of ***user-defined 'doctrine statements'****. AWS Doctrine is thus adaptable to various rules of engagement and compatible with different tactical control structures".*

"Aegis tactical doctrine can be implemented in an automatic, semi-automatic or manual mode. The first two modes reduce delays introduced by required operator actions. Doctrine statements are essentially 'standing orders' to the Aegis Combat System (ACS) collated in different types of 'if <expectation>, then <action>' statements that are created/modified and activated/deactivated, one at a time or in sets, by authorized sub-modes. Doctrine statements combine operator and system automation strengths. Principal Warfare Officers and Combat System Operators will use them to allow the combat system to make tactical decisions under human supervision". Note that in this description, the Principle Warfare Officers and Combat System Operators can both be considered "Edge programmers."

There is anecdotal evidence that soldiers are already leveraging and, in some cases, customizing Commercial off the Shelf (COTS) and Open-Source Software (OSS) to solve urgent problems, enhance the dissemination of information, or simply make their jobs easier. In one example, a Sergeant stationed near Mosul used second hand laptops, salvaged wires, and freely available Voice over IP (VoIP) software to improve communications between watch towers and the home base. That same sergeant also integrated open source video software with his Forward Looking Infrared (FLIR) system to assist in monitoring for insurgents [13].

PBS's Frontline published an interview with a Major from 1st Cavalry Division who created CAVNET—a "knowledge transfer system"—accessible to company level commanders for sharing and evolving Tactics, Techniques and Procedures between missions. In the interview, the Major contrasts traditional military organizations with one more consistent with Edge Enabled principles [19]:

"…our culture is inherently hierarchical and stove-piped when it comes to the validation of "new actionable knowledge." Normally it will stay within the unit based on the way we have created our After-Action Review process. The learning that is achieved will be fed back into the same unit. But wouldn't it be great if that learning could be transferred laterally?"
"I had an idea for a series of unit-level networks at each of the major Army installations, re-sourced by high-powered captains who worked directly for the commanding general [and] who would collect, observe, connect, collaborate, and disseminate -- on behalf of the command -- the created knowledge of a unit.
It was not initially received well..." "…'Too hard,' 'Not relevant,' 'Won't work,' 'Not what we're about,' were the common responses. I guess the idea kind of festered in the mind of my old commander, Col. Paul Funk, who brought up a variation of the idea to Maj. Gen. Chiarelli, and Lt. Gen. Thomas Metz (III Corps commander) who saw the power of the idea in a different con-text."

Many commonly used commercial software applications are Edge-enabled, and provide varying degrees of user programmability. Email clients enable users to create rules that automatically perform any number of actions according to user definable criteria. A number of popular video games give users the tools to create customized characters and environments. Productivity suites such as Microsoft Office have long included scripting and macro capabilities to extend or auto-mate functionality and as of Mac OS 10.4, Apple has provided Automator which lets users build customized workflows and automate repetitive tasks across applications.

B. Inhibitors to Enabling the Edge

All change introduces risk, and all change is disruptive; and edge programming is especially disruptive to an organization such as the Army where the consequences of risk may be matters of life and death. Inhibitors to adopting edge technology arise in several areas of concern:

- *Established Practices*: US Warfighters are trained to solve their own problems when existing tools do not meet their needs. However, for many good reasons, it is often argued that all changes to a sys-tem should be done through a Program Manager. Only in this way can the Army avoid configuration management problems, duplication of efforts or the development of over-lapping capabilities, and an inability to standardize across the forces to include training, maintenance, interoperability, logistics and sustainment.
- *Cultural Disconnect*: Warfighters are still seen primarily as consumers of information rather than producers of information, and many are not convinced a community would form that could sustain an edge enabled environment in the long run. Moreover, there is skepticism that crowd-sourcing has any bearing on combat operations, and there is no doctrine that corresponds to value creation and co-creation.
- *Information Assurance and Policy*: There exists a lengthy certification and accreditation process for getting new systems approved for use on DoD networks. Prior to becoming operational a system must obtain an Authority to Operate and in some cases a Certificate of Networthiness. Unfortunately, that process is difficult to navigate, can be very time consuming and requires specific personnel certified to perform the analysis. The fear is that by enabling edge programming, Warfighters may inadvertently create security vulnerabilities in the application or system being modified.
- *Security and Classifications*: Some fear that enabling users to "mash up" existing data from various sources might un-intentionally result in hybrid information that requires a higher level of classification than its original components. Additionally, there are concerns with rights management and controlling user access to data across systems. In fact, over-classification of data can itself be regarded as a fundamental inhibitor to edge-enabled systems.

III. Structure and Characteristics of Edge Organizations

A. *"The Edge" in the DoD*

In "Power to the Edge: Command… Control… in the Information Age" Alberts and Hayes discuss the concept of an "Edge Organization" and what it means to migrate from an "Industrial Age" organization of stovepipes and hierarchies to an "Information Age" organization of empowered individuals [2]. In their book, they focus on the need for interoperability, agility and enabling rapid information exchange through highly networked peer-to-peer relationships. The goal is "not to be able to perform well in a particular mission, but to create an organization that is agile … able to meet unexpected challenges, accomplish tasks in new ways, and learn to accomplish new tasks." GAO Report (GAO-04-547) to Congress endorses this thinking by stating:

While senior leaders are becoming increasingly involved in operations, information is also being distributed to lower and lower organizational levels, raising the potential for increased autonomy for small units and individual soldiers. For example, one of the principal organizing and operating tenets of network-centric operations is the concept called power to the edge. This concept involves empowering individuals at the "edge" of an organization—where it interacts with its operating environment—by expanding access to information and eliminating unnecessary constraints on action. [10]

This mentality must extend down to the systems and applications that facilitate information ex-change within the organization. In effect, getting away from the notion of "one size fits all" systems and enabling users to work together to more rapidly create applications that better suit their needs: "capabilities that are better tailored, better understood and easier to use and modify." [2]

B. *The Acquisition Process*

To become an "Edge Organization", organizations have to rethink their current acquisition processes. Originally, designed for an Industrial Age military and the procurement of military specific hardware (e.g. tanks and ships), the US Army's acquisition process was not designed to keep up with the ever increasing speed of technology change. This problem is compounded by the sheer size and number or organizations involved in the creation of an Army System. The Training and Doctrine Command (TRADOC) creates the doctrine and identifies gaps in capabilities. Program Executive Offices (PEOs) and Program Managers (PMs) plan, design, develop, test, field and maintain software that fills the identified gap. Research and Development organizations and private contractors develop the systems according to PEO/PM requirements. The Office of the Chief Information Officer (CIO) and other organizations (e.g. Defense Information Systems Agency (DISA)) create and enforce the policies to which software based systems must adhere. All of this happens in multi-year cycles with minimal input from the actual users of the systems: the experts with the best understanding of the problem space and operating environment.

But most soldiers today are "digital natives": they own a smart phone, belong to social networking sites and have used or even contributed to OSS projects. Compare this with the Army-provided environment: a centralized procurement process that often delivers systems that are not what the soldier asked for, no longer applicable to the current mission, or are desperately outdated. It should be of no surprise that soldiers often resort to solving their own problems.

C. *Implications of Enabling the Edge*

Software engineering has long embraced centralized production models, where requirements are collected, projects are managed, architectures are created, and correctness is determined in a tightly controlled process. It is hierarchical and rule-oriented (not commons-based or egalitarian). Software development methods emphasize centralized planning and control. Even Agile methods stress the importance of face-to-face communication and the advantages of the "bullpen"—an open office where workers freely interact.

But if EESs are to be truly embraced then the rules and tools must be radically changed. Such projects will, to varying degrees, be community driven and de-centralized with little overall control, as is the case with the major social networking communities (e.g. MySpace, Facebook), open content systems (e.g. Wikipedia, YouTube), and with OSS development today ([9], [16]). Thus we can no longer design and implement such systems using older models. If systems are constantly in a state of perpetual beta [18], if they are regularly updated and combined in novel ways, and if a large part of their utility is in their comprehensiveness and ubiquity then our concerns, from a software engineering and project management perspective, must reflect this. All successful EESs and the organizations that develop and use these systems, share a common structure, shown in Figure 2. We refer to this as a *Metropolis* structure.

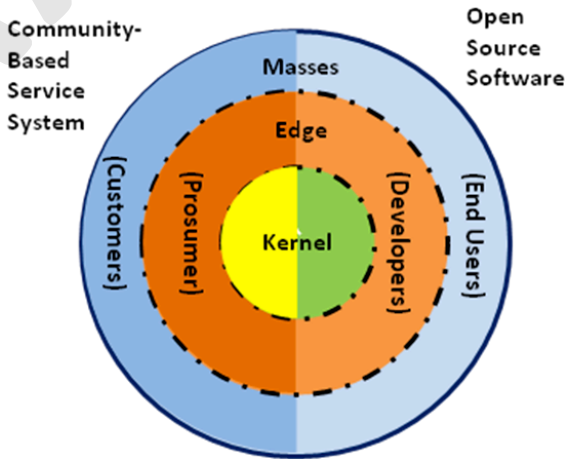

Figure 2: The Metropolis Structure of an Edge-Enabled System [14]

There are three concentric realms of roles (and associated infrastructure) in an EES, as indicated by the "circles" labeled *kernel*, *edge*, and *masses* in Figure 2. In the outermost ring are the masses of end users of such systems. They contribute requirements, but not content. The middle ring contains

developers and *prosumers* (a portmanteau of *pro*ducer and con*sumer*), and these are the stakeholders at the edge whose actions and whose value-creation the organization would like to facilitate. All of this is held together by the kernel. Some example roles for individuals involved with the kernel are architects, business owners, and policy makers.

There are also differences in the "permeability" between the realms—as the figure indicates by the dashed and solid lines—between the two major types of edge-enabled systems: community-based service systems (such as Wikipedia, Twitter, YouTube, Slashdot, and Facebook) and OSS systems (such as Linux, MySQL, Apache, Eclipse, and Firefox). For example, in OSS development it is possible to move from the role of an end user to a developer to a kernel architect, by consistently contributing and moving up in the meritocracy. Thus a key question for an organization that wishes to foster edge-enabled systems is: how should we craft the kernel and what development principles and practices should we embrace?

In addition, we must recognize that there are many forms of contribution to EESs that are not programming. To be truly successful EESs must foster the *prosumers*, who are typically not programmers or technical contributors. But *prosumers* are responsible for most of the content on Wikipedia, YouTube, and on the recommender systems [22] and collaborative tagging systems [11] that provide so much of the content of today's Web.

Not all *prosumers* are created equal. Frequently there is a system of regulation that accompanies the contributions to EESs. For example in Slashdot, the on-line technology news and news commentary site, users are only granted moderator status after they have earned enough "karma" (positive recommendations) from the community. Moderations perform an accreditation function which is continually responding to the reactions of the *prosumers* to a moderator's postings. And there are meta-moderators, whose function is to rate moderators. This accreditation function is similar to the process of peer review found in academic communities, relying on the accumulation of comments by peers (who themselves have undergone some form of vetting to be in this position) to establish trustworthiness and quality. [4]

We see *prosumers* functioning in "quality control" functions in many aspects of the Web. Wikipedia has a group of volunteer editors (administrators) who have oversight responsibilities for its integrity. As with Slashdot, such rights and responsibilities are earned, as determined by a group of peers (other editors). In this way EESs are self-monitoring and self-regulating. In the OSS domain there is an analogous system of rights and responsibilities. Anyone can contribute to an OSS project, but only a few can contribute to the kernel and these individuals must be approved by existing members of the core group [8].

Metropolis projects (and the organizations that create them) share the following characteristics [14]:

1. *Mashability:* the systems are seldom created from scratch; creation as "mashups" is far more common, borrowing freely from other Metropolis efforts
2. *Conflicting, Unknowable Requirements:* requirements in a peer-produced system emerge from its individuals, operating independently; requirements are never knowable in any global sense and they will inevitably conflict, just as the requirements of a city's inhabitants often conflict
3. *Continuous Evolution:* Metropolis systems are constantly changing; resources are non-centralized and so a peer-produced system is never stable. One cannot conceive of its functionality in terms of "releases" any more than a city has a release: parts are being created, modified, and torn down at all times
4. *Focus on Operations:* Metropolis systems focus on operations as a core competency. This implies high availability, scalability, and seamless evolution.
5. *Open Teams:* these projects have decentralized production processes with no traditional management. Teams are diverse with differing, sometimes irreconcilable, views.
6. *Sufficient Correctness:* Metropolis systems do not claim to be complete or correct. Sufficient correctness and perpetual beta are the norm. This is a deliberate tradeoff to achieve agility and rapid alignment with user needs.
7. *Unstable Resources:* Applications that are peer-produced are subject to the whims of the peers; however large numbers tend to ameliorate the actions of any individual. Despite the lack of guarantees, unstable resource pools have resulted in significant computational achievements (consider, for example, that Skype achieves near telephone quality through peer-contributed resources).
8. *Emergent Behaviors:* in contrast to existing systems emergent behavior is considered normal and desirable. Metropolis systems regularly push the boundaries of what their creators intended.

In the next section we will discuss a number of different models for structuring the software of a Metropolis system. Some of these models apply just to the kernel and others are applicable to the entire system.

IV. Mechanisms for Managing the Edge

Here we summarize a variety of products, processes, information access models, and technologies found in practice which, although not invented for use in EESs, has particular resonance with the Edge. We refer to these collectively as "mechanisms." We characterize each of these mechanisms and then we describe their advantages and disadvantages.

A. Configuration

The system provides a number of parameters that the user can choose from and set. This is a fairly traditional notion of system configurability and typically only provides for a small amount of edge-based tailoring. Examples of such systems include:

- Enterprise Resource Planning (ERP) systems (such as Oracle Financials, Oracle Mobile Supply chain Applications, SAP Business Suite, PeopleSoft etc.).
- Server-based applications systems such as web servers, databases, content management (such as Apache, Microsoft IIS, Google GWS, etc.)
- User applications (such as Microsoft Word, Excel, Safari, Internet Explorer, Integrated Development Environments).

ERP systems are highly configurable to fit the needs of different users, with varying requirements and in different environments. In fact, configurability is perhaps the most salient architectural characteristic of ERP systems. Here configurability extends well beyond simple configuration files and typically also includes templates, scripting, plug-ins, execution rules or other methods that can be used to extend or modify the behavior of a software system. Thus as systems become more configurable they tend to include a mix of mechanisms to extend or modify their behavior. These types of systems win in the marketplace with respect to buy-versus-build decisions as they can be customized quickly and cheaply. However, due to the highly configurable nature of these systems their kernels are often more expensive to build and maintain than a simple custom solution, thus they are usually only cost-effective with a very large user base. Moreover, the process of configuring an ERP system can itself be daunting, and this should be sufficient to demonstrate that highly configurable kernels is not sufficient to achieve agility, which bears with it notions of quick adaptation.

At the other end of the scale, user applications typically only allow limited configurability. This level of configurability usually only allows the user to turn on and off various program features and occasionally some customization of the application displays. At the high end of user application configurability, Integrated Development Environments (IDE) often allow the user to include scripts to support customized build environments and applications such as Microsoft Word and Excel allow the user to build customized user environments through the use of Visual Basic Macros which would fall into the category of mixed mechanisms (i.e. configuration with scripts or plug-ins).

Advantages: Providing the user the ability to choose among a pre-determined set of possibilities limits possible options at the edge. For this reason the system's creators can employ substantial quality assurance effort to validate that the system operates consistent with its specification. Consequently such a system can provide much greater quality of service guarantees.

Disadvantages: In highly dynamic environments—where the nature of the use of the system can change dramatically—the configuration approach typically adds relatively little value to the end user, particularly as the user's requirements extend beyond the envelope of what the system's specification cover.

B. Scripting

The system provides a special-purpose scripting language that allows the user to tailor, specialize, and augment the system's capabilities. Examples of such systems include web servers, web browsers, and IDEs. Scripting languages and the environment in which they are used can also control the kinds of modifications and/or enhancements that can be made by the end user or script writer. Some applications allow scripts completely unrestricted access (un-sandboxed) with respect to the application, its data, the underlying operating system and connected hardware. In this situation, the ability to modify the application is only limited by the application's inherit architecture along with the restrictions placed on the application by the underlying operating system. Scripts that are allowed to run with unrestricted access can put system security, robustness, and integrity into jeopardy. For this reason scripting languages typically only allow restricted access to resources and capabilities. These types of restrictions are also referred to as "sandboxing".

Advantages: Scripting languages provide a great deal of flexibility and generative power to the users, and relieve the system creators from the difficulty of trying to anticipate all the requirements in advance.

Disadvantages: Scripting languages are very complex to create, and users typically do not want to learn special-purpose languages. And because they are special purpose, they often have subtle performance or security flaws that their creators did not envision. Scripts can cause application instability issues due to unforeseen memory and CPU resource allocation by the script developer. Since such languages do not have the widespread scrutiny of major programming languages, such flaws may go undetected indefinitely. Last, as is well known, scripting languages often emphasize ease of writing scripts rather than ease of reading, or maintaining scripts. A perverse consequence of scripting is that systems can become less adaptive as the volume of ad hoc scripts, with their uneven quality and undocumented interactions grows over time.

C. Application Platforms

The system provides a coherent set of APIs (application programming interfaces) and supporting infrastructure that collectively represent an "application platform"—a platform upon which user-created applications can be built. The user or third party module/applications that utilize these APIs are often referred to as "Plug-Ins". The user programs their own application using the plat-form's services as primitives. Examples of well-known application platforms include Facebook, Google Earth, Firefox and Internet Explorer. The Java language/JVM (Java Virtual Machine) and the .NET framework are also application platforms, but with a much broader mandate—to support virtually any kind of application development on top of a consistent computational base.

Advantages: application platforms are a widely accepted and widely successful way of enabling third party creation of functionality. They provide an easy (and potentially seamless) method for adding additional functionality to an application. They provide a structure for programmers, guiding them in creation and freeing them from many of the mundane platform and resource management issues. The JVM, for example, frees a developer from many concerns regarding porting and memory management.

Disadvantages: application platforms are complex and hence difficult to engineer successfully and contain tradeoffs that may

or may not be appropriate for the application under development. It can be difficult to restrict access to resources and capabilities. For example, the JVM sacrifices performance in return for platform independence. The platform may be a single point of failure (for example, a failure in the infra-structure may create many common mode failures among the applications which are built on the platform). A poorly designed extension architecture along with the platform can needlessly limit a programmer's creativity.

D. Sandbox

The user can create any kind of application, but it can only be run in a "sandbox"—an execution area with limited resources and capabilities. Sandboxes have long been used in software testing and maintenance activities, to provide a safe area to try out new ideas and new code. But this is typically just a matter of cloning an existing system or environment and providing it to developers as their personal "play area". A more relevant use of sandboxing comes from computer security, providing a virtualized environment for running unproven or untested applications. Such sandboxes typically have limited access to system resources, system information, network, and I/O devices. Common forms of sandboxing are seen in the applets that most modern web browsers support (e.g. Flash, Java applets, etc.). Google's Chrome web browser has implemented sandboxing as a security mechanism. Each tab within the browser is its own process and cannot directly affect other browser processes (for example, malware running in one tab could not capture credit card information entered in another tab). Nokia platforms running the Symbian OS accomplish sandboxing though the use of certificates that control application capabilities and data access.

Advantages: this form of programming provides few constraints on a programmer's creativity, so long as the programmer's creation does not attempt to use resources beyond those prescribed by the sandbox. Also a sandbox can put limits on resource usage (e.g. CPU, disk I/O, network bandwidth) which can serve as a way to manage performance amongst many competing applications.

Disadvantages: this form of programming, while providing few constraints, also provides no structure for enabling or enhancing a programmer's creativity; it is a mechanism for controlling what they do, not guiding them in the act of creation. Also the requirements for the limits of the sandbox will be difficult to determine, since they must limit what a programmer can do, to ensure the safety, security, availability, and performance of the system, but at the same time they must provide the programmer with enough resources to be able to create something of value. Finally, creating a sandbox that is "bullet-proof" is a substantial software engineering challenge.

E. Qualification

The user can create any kind of application, but before it is included in the system it must be qualified (approved, certified, and signed) by a third-party agency. Applications that are not signed by the agency will not run in the system. For example, Microsoft runs the WHQL (Windows Hardware Quality Labs) testing process on third-party software or hardware. Apple has provided a similar mechanism for the iPhone, attempting to limit iPhone apps to just those provided by Apple's "App Store". Another form of qualification is third-party, often relying on crowdsourcing. For example, Facebook applications are rated by users, not by Facebook.

Advantages: qualification can provide some assurance to users that the applications they are using have passed some level of quality assurance testing.

Disadvantages: There are three common problems with software qualification: 1) the qualification process itself may have exploitable flaws in it that allow an unsafe application to be signed; 2) like any security measure, software signing can be circumvented, either by providing fake signatures, by tricking a user into running unsigned code; 3) the assurance may not address all qualities of interest (e.g. safety, robustness, performance, security, etc.).

F. Monitoring

The user can create any kind of application, but the system monitors its execution and, if it exceeds any limits (resource usage, behavioral) it is either terminated or non-complying operations are ignored and/or non-complying operations are recorded and appropriate entities are notified. Monitoring is already common practice for networks (detecting intrusions, overloaded servers, crashed servers, etc.), disk usage, CPU loading, transaction throughput, etc.

Advantages: monitoring software and hardware sets few limits on what an application can be and what it can do. And monitoring can free humans from the tedious oversight process, where fatigue is a perpetual concern.

Disadvantages: monitoring, when it works, will at best detect a problem that has already occurred. This approach is thus limited to organizations where there is tolerance for some forms of aberrant behavior (for example, the organization might tolerate overloaded servers for a short duration). Also monitoring systems often need to be "trained" to establish a baseline of acceptable performance parameters.

G. Adaptive Need-to-Know Information Access

The way in which access to information is managed at the Edge is a crucial consideration. Contrary to common binary risk assessments, in which any possibility of exfiltration is considered an absolutely negative consequence, risk assessment for unauthorized information access in an EES requires considerations of timeliness of information, as well as its useful lifespan. For example, legitimate users of an EES may need to rapidly exchange confidential information at lower-than-usual levels of security to foster timely access to information that will quickly lose its value. The application of the usual security controls and procedures that would be appropriate for the information may actually shorten the lifespan of the information, or render it completely useless. The issue of disseminating information in an EES, therefore, becomes primarily one of evaluating if a security control is a sufficient impediment to retard unauthorized access, with the notion of sufficiency to include both time-to-access and the lifespan of usefulness of the information.

The most frequently used technique for protecting access to confidential information is an Access Control List (ACL). There have been three major approaches to this type of security [12]. The first is a multi-level security (MLS) approach that imposes mandatory security policies on all confidential information hierarchically. The second approach is a discretionary access control (DAC) that works on the basis of the creator's permissions: the creator of confidential information is responsible for the decision of who has what access. Finally, a role-based access control (RBAC) approach maintains a list of roles that encapsulates access rights to a set of confidential information. The users under this type of system are assigned corresponding roles according to their responsibilities. These approaches employ ACLs in response to requests for confidential information. They are slightly different from one another in what they emphasize, however: DAC and MLS focus on the item to protect and RBAC focuses on the role of the user. These approaches suffer three short-falls that render them difficult to apply to EES contexts:

1. **Inadequate coverage:** determining membership in an access control list is difficult and time-consuming, and so rapid membership decision criteria will either be too restrictive, excluding some on the Edge who should have access, or too broad, giving access to all members of a category, irrespective of their need-to-know.
2. **Non-adaptive:** in accessing information on the edge, the process of vetting information or a user for membership in an ACL can be a significant overhead, and possibly defeat the usefulness of access to the information.
3. **Expensive maintenance:** The maintenance and assignment of ACLs is challenging. Since the number of ACLs that must be updated increases cubically in proportion to the units of: confidential information, the groups of constituents, and the number of operations; it is computationally and temporally expensive to maintain.

A more adaptive technique for allowing security classification of documents based on a user's need-to-know, has been proposed by [21]. This technique, which can be applied in conjunction with an ACL, treats the determination of kneed-to-know as a statistical document classification problem. With this technique, the user declares positive and negative examples of the types of information that they have a need-to-know. The classifying system creates a profile of the user-declared "topic", and subsequently matches all future documents requested by the user to that profile. If the document matches the user's profile, they can access the information, if not, they may not. Such profiles are tamper-resistant and can be verified by external security auditors. This technique addresses the shortcomings of ACLs, by allowing adequate coverage, being adaptive, and cost-effective to maintain. The drawback is that it requires a training set of around 100 documents to create the initial profile.

Advantages: ACLs are quick and easy to implement for small numbers of classification categories and users. Adaptive need-to-know authorization allows for the scalable tuning of access control based on a generalized profile description of a topic in a document collection. Since it is complementary to the ACL method, adaptive authorization is best applied to a large document collection that is already protected by a coarse-grained level of classification that can potentially facilitate access by a large variety and number of users.

Disadvantages: Access control lists are not scalable or tunable to fine-grained discrimination, requiring human classification for each new document and classification category. The lack of scalability typically results in little or no access to timely sensitive information, which is more likely to occur in an EES. While adaptive need-to-know authorization is scalable and provides for the automatic classification of documents, it requires initial training of around 100 documents to prevent unauthorized access to confidential information.

V. Recommendations for a New Model of Software Development and Acquisition

The "Edge" is the intersection between users and their operating environment. If future systems are to be EESs then the organizations that are tasked to build these systems must change. "Edge organizations are characterized by the widespread sharing of information and the predominance of peer-to-peer relationships. Edge organizations have a fundamentally different power topology as compared to traditional organizations. In an Edge organization, virtually everyone is at the edge because they are empowered. An edge organizations means that everyone is empowered by information and has the freedom to do what makes sense" [2].

To move towards Edge-enablement—some form of a Metropolis Model—many things about the process of developing, testing, and fielding such systems must change. First, a Metropolis Model must be a hospitable place. As with a city, people must want to "live" there. Therefore infrastructure and rules must be in place to create the social and technical mechanisms to entice long-term participation, encouraging community custodianship, recognizing the merits of individuals and promoting them through different "ranks" with appropriate rights and responsibilities, and protecting the community from the acts of malicious participants. Below we suggest a number of changes that an organization must consider if it wishes to foster EESs.

A. Leadership and Management

To manage a Metropolis project the periphery must share in its success. The project must be—to a far greater extent than is the norm today—self-governing and self-adaptive. Many leaders of OSS projects have admitted that they do not "lead" such projects in any traditional sense. In OSS project work is not assigned; developers choose their work. As such, the leaders of such projects spend much of their time attracting, motivating and coordinating developers. The little structure that does exist in such projects is based on a meritocracy [16]. Periphery members cannot be strictly controlled and managed in the way that traditional projects are controlled and managed today (largely top-down, hierarchical, and rule-oriented), but must instead be inspired, persuaded, and motivated.

B. Project Structure and Communications

Because of its distributed nature, a Metropolis project must have a minimum of hierarchy and bureaucracy, and there must

be collaboration technology in place for communication and coordination [16]—typically email lists, wikis, and discussion forums but perhaps including teleconferences, videoconferences, and web-conferences. Even the entrance of many for-profit companies to the OSS movement has not changed the nature of their project management; they remain consensus based meritocracies rather than top-down hierarchies. This implies the need to focus project management on communication and negotiation to guide contributors and to persuade them to share in the project vision.

Any environment that claims to enable or facilitate the Edge must provide some mechanism for members of the community to discover, communicate and network with others to freely exchange knowledge, insight and experiences. The nature of being on the edge means users will be distributed, have varying technical skills, different backgrounds and will not all be expert software developers. To create new capabilities users must be able to leverage the knowledge, experience and work of others.

C. *Requirements Management*

The requirements process in such projects needs to be radically changed from what exists today, where the Army still follows a waterfall lifecycle model in most cases. In a Metropolis the kernel is centrally specified and controlled (and may be created as today's current projects are created: plan driven and top-down), but the requirements of the edge emerge from the participants who are intense users of the system. These requirements and their fulfillment are what allow the organization and the system as a whole to be agile; and they contribute the vast majority of the value in such a system. This means that there is still a role for central planning and deployment—as suggested in Section IV, a model for managing the edge must be chosen and implemented—but the main purpose of the kernel is to enable the edge and not to provide a comprehensive "one size fits all" system, as was discussed in Section III.C.

Information and knowledge can no longer be treated as a scarce and guarded commodity. Edge enablement requires that everyone has the ability inspect and repurpose the work of others. This not only ensures knowledge transfer, it lowers the barrier of entry and reduces duplication of work.

D. *Quality Assurance*

The system must be able to maintain its core security, robustness, and integrity characteristics throughout modification. But the process of quality assurance must change in a Metropolis organization. First and foremost, the kernel must be highly reliable. However, this requirement is tractable because the kernel is typically small; often orders of magnitude smaller than the edge. Also the kernel will be highly controlled, and slow to change (as are most Army systems today). This can and does work: the reliability and security of the most popular OSS products has been reported to be quite high [6], [17]. The reliability of the edge, on the other hand, is indeterminate; sufficient correctness is the norm. This is why it is critical to choose the appropriate model (as discussed in Section IV) for the enablement and management of the edge.

E. *Architecture*

Because so much depends on the architecture of the kernel, it must be designed to accommodate the needs of the edge. For this reason, the architecture cannot "emerge" as it often does in traditional and agile lifecycle models. The architecture must be designed up front, built by a small, experienced team who focus on: 1) modularity, to enable the parallel activities of the edge, and 2) the core quality attributes of the kernel (security, performance, availability, etc.). The kernel creators also need to pay attention to the usability (simplicity and learnability) of the kernel, so that it is easy for the periphery to build on it. Wikipedia succeeds, in part, because it is trivial for a prosumer change an article. Facebook and the iPhone succeed, in part, because a developer can create simple applications in a few hours. Thus the kernel creators also need to make examples and tutorials available, to aid the programmer at the edge.

F. *Delivery Mechanisms*

Edge-enabled systems are never complete and they are not delivered in an all-or-nothing fashion. Therefore delivery mechanisms must be created that work in a distributed, asynchronous manner. And these mechanisms must be flexible enough to accept incompleteness of the installed base as the norm [25]. Thus, any delivery mechanism must be tolerant of older versions, multiple co-existing versions or even incomplete versions.

G. *Moving Forward*

Any organization that wishes to foster the edge needs to think about the sources of risk. There are many operational risks that an EES creates as compared with the relatively "locked down" systems of today. However, a locked-down system also creates risks—principally from being poorly aligned with the needs of the Warfighter. As the system is opened up and controls are loosened, operational risks increase. And as a system is locked down, misalignment risks will increase. This situation, along with its inherent tradeoffs, is depicted in Figure 3.

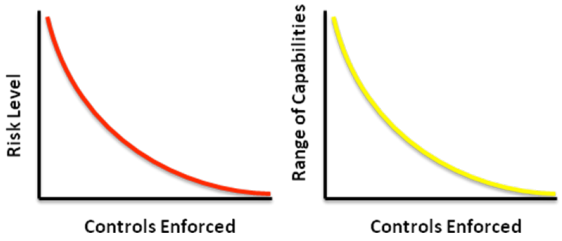

Figure 3: Edge Models and Risk

H. *Conclusions/Road Map*

Some of the key questions that must be answered for an organization or group that wishes to foster EES are:

1. How adaptable should the system be? Greater adaptability increases both flexibility and the risk of losing control of integrity, reliability and consistency. Guidance on how assess Risk/Return tradeoffs need to be developed.

2. What development principles should be employed to maintain to ensure EESs?
3. What project management procedures and methods should be applied to EESs?
4. What fundamental changes need to occur in the QA process when fielding and operating an EES?
5. What fundamental changes need to occur in the acquisition organization and their processes?
6. Does an organization provide additional incentives to the participants? For example is there a need to compensate or acknowledge high performers?

No one set of recommendations can possibly fit all of the variability implied by the above discussion. Different projects have different risk/reward profiles. Hence we recommend that a small number of model projects be started that follow a Metropolis model. Such projects will allow the Army to better understand the risks and rewards of employing the edge in a realistic context, and will best inform the way forward.

To make progress in enabling the edge we believe a documented, repeatable edge design and evaluation method is required. This would include:

1. Needs analysis and problem focus would involve a classification of edge problems and the selection of problems for edge experimentation.
2. Empirical analysis of EES mechanisms would involve designing experiments that permit the analysis of management mechanisms in simulated EES programming workflows. The experiments would be designed to determine, for a given class of Edge problems, how well the mechanism facilitates agile responses to evolving problems, and if not, what future requirements might be.

   An example of this would be building a number of systems with varying levels of constraints utilizing the mechanisms discussed in Section IV. Subjects would then extend these systems to include edge enabled capabilities and the impact of the constraint mechanisms would be analyzed.
3. The development of an Edge design and evaluation methodology would enable all stakeholders to analyze the criteria used in determining the applicability of an EES, the architecture of the EES, the appropriateness of the choice of Edge management mechanisms, as well establishing criteria to know when the EES architecture is being stressed beyond its capabilities.

REFERENCES

[1] A—Edge Programming, Solicitation Number: W15P7T08P00, Agency: Department of the Army, Office: Army Contracting Command Location: CECOM Contracting Center (CECOM-CC), Oct 13, 2009 10:06 am https://www.fbo.gov/index?&s=opportunity&mode=form&id=520059c516f2031d3de912d982414006&tab=core&tabmode=list, Retrieved 12/2/ 2009

[2] D. S. Alberts and R. E. Hays, Power to the Edge: Command Control in the Information Age (CCRP: April 2005) ISBN 1-893723-13-5

[3] J. Barr, "The Paradox of free/open source project management", http://www.linux.com/feature/42466, Retrieved December 18, 2009

[4] Y. Benkler, *The Wealth of Networks: How Social Production Transforms Markets and Freedom*, Yale. University Press, 2006

[5] M. Bergman, "Commerce Secretary Evans Says New Economic Indicator on Service Industries Will Help Close 'Critical Gap', 2004 http://www.census.gov/Press-Release/www/releases/archives/economic_census/005366.html

[6] Coverity.com, "Analysis of the Linux Kernel", http://www.coverity.com/library/pdf/linux_report.pdf. Retrieved 12/21/ 2009.

[7] J. Howe, "The Rise of Crowdsourcing", *Wired*, 14.06, June 2006.

[8] R. Fielding, "Shared Leadership in the Apache Project", *Communications of the ACM*, April 1999, Vol. 42, No. 4, 42-43.

[9] R. Fielding, R. Taylor, "Principled Design of the Modern Web Architecture", *Proc. International Conference on Software Engineering* 2000, 407-416.

[10] GAO 04-547, "Military Operations: Recent Campaigns Benefited from Improved Communications and Technology, but Barriers to Continued Progress Remain," GAO Report to Congressional Committees, June 2004.

[11] S. Golder, B. Huberman, "The Structure of Collaborative Tagging Systems". Technical report, Information Dynamics Lab, HP Labs, 2005.

[12] D. Gollmann, *Computer Security*, John Wiley and Sons, Inc., (1999).

[13] Gunnar Hellekson, OnePeople Blog "Open Source on the Battlefield", http://onepeople.org/node/1600, Retrieved 12/16/ 2009.

[14] R. Kazman, H-M Chen, "The Metropolis Model: A New Logic for Development of Crowd-sourced Systems", *Communications of the ACM*, 2009, No. 7.

[15] P. Maglio, S. Srinivasan, J. Kreulen, J. Spohrer, "Service Systems, Service Scientists, SSME, and Innovation", Communications of the ACM, July 2006, Vol. 49, No. 7, 81-85.

[16] M. L. Markus, B. Manville, C. Agres, "What Makes a Virtual Organization Work?" Sloan Man-agement Review, Fall 2000, 13-25.

[17] A. Mockus, R. Fielding, J. Herbsleb, "Two Case Studies of OSS Development: Apache and Mozilla", *ACM Transactions on Software Engineering and Methodology*, Vol. 11, No. 3, July 2002, 309-346.

[18] T. O'Reilly, "What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software", http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html, Retrieved 11/2/ 2009.

[19] PBS interview with Maj. Patrick Michaelis, "Innovating & Improvising", http://www.pbs.org/wgbh/pages/frontline/shows/company/lessons/, Retrieved 12/16/ 2009.

[20] Semaphore Issue 07, JUNE 2009 Sea Power Centre–Australia Department of Defence Canberra Act 2600 http://www.navy.gov.au/Semaphore_Issue_7,_June_2009, Retrieved 12/4/ 2009.

[21] Y.-W. Seo, J. Giampapa, K. Sycara, "A Multi-Agent System for Enforcing `Kneed-to-Know' Security Policies," *6th Int'l Bi-Conference Workshop on Agent-Oriented Information Systems*, July, 2004.

[22] G. Shani, A. Gunawardana," Evaluating Recommender Systems", Microsoft Research Report MSR-TR-2009-159, Nov. 2009, http://research.microsoft.com/apps/pubs/default.aspx?id=115396, Retrieved 12/14/ 2009.

[23] K. Sullivan, "Edge Programming", *Proc. 29th International Conference on Software Engineering Workshops*, 149, 2007 , ISBN:0-7695-2830-9

[24] S. Vargo, R. Lusch. "Evolving to a new dominant logic for marketing", Journal of Marketing, Vol. 68, Jan. 2004, 1–17.

[25] Werner Vogels "All Things Distributed Werner Vogels' weblog on building scalable and robust distributed systems", "Eventually Consistent" http://www.allthingsdistributed.com/2007/12/eventually_consistent.htm, Retrieved 12/14/ 2009.