Raging Incrementalism—System Engineering for Continuous Change

Michael M. Gorlick
The Aerospace Corporation
El Segundo, California
310.336.8661
gorlick@aero.org

***Abstract*— For nearly every technology there is a *doubling period* over which the price/performance ratio of that technology improves by a factor of 2. Careful examination of long-term historical trends reveals that these doubling periods are themselves shortening every decade; implying that the rate of technical progress is *hyperexponential*. If we take the 1990s as the benchmark decade of technical progress, then the first 25 years of the 21st century will see the equivalent of 100 years of 1990s progress. Hyperexponential progress has profound implications for all facets of system procurement, design, development, deployment, management, improvement, operations, maintenance, and modernization. We offer an approach to system engineering—*raging incrementalism*—that embraces and leverages hyperexponential change to construct systems in far less time and with dramatically lower life cycle costs than traditional methods. We discuss the rationale for raging incrementalism, its programmatic and system engineering principles, and its influence on total system cost from concept to disposal.**

Focus Issues:

- **Standards and interoperability**
- **Off-the-shelf**
- **Open-source components**
- **Program management, including risk and life-cycle issues**
- **System integration**
- **Ground communication architectures**
- **Operations and sustainment concepts**

I. Introduction

To many, change—be it in the form of requirements creep, funding rates, or the ongoing evolution of technology—is the enemy of systems development. System developers and customers lock out change by freezing requirements, guaranteeing funding, and commiting early to key technologies. For shame—ignoring change is as foolhardy as ignoring gravity. Change is now so pervasive that it constitutes a fundamental driver for systems large and small. Not only is change the one and only constant of systems acquisition and deployment, but the pace, degree, and breadth of change is growing at a *hyperexponential* rate. In this paper we offer a perspective on systems engineering that embraces change as a force for system improvement and suggest a methodology that leverages change to deliver quality systems at lower costs and in less time than traditional methods.

Consider recent technical history. For several technologies relevant to space ground systems—obvious examples include computation, data storage, and communications—there is a technology-dependent *doubling period* (measured in months) over which the price/performance ratio of that technology improves by a factor of 2. To illustrate, the doubling period of modern processors—such as the Intel Pentium or the Power PC—is approximately 18 months. In fact, processor technology exhibits two related and concurrent doubling periods in that roughly every 16–24 months *both* the gate density and clock speeds double, yielding a *4-fold* performance improvement every doubling period. Similar and equally dramatic (but different) doubling periods are observed for dynamic memory, disk drive storage density, network bandwidth, router switching speeds, and raw fiberoptic carrying capacity. In each instance cited above, the doubling period is less than 24 months—and in some cases, markedly less. For example, until recently disk-drive density was doubling *every 6–8 months*; however, industry analysts predict a period of far more modest improvement as vendors reach the limits of present technology and retreat to the laboratory to explore promising advances. Quiescent periods are not uncommon in the historical performance curves of technologies; however, they rarely last longer than 12–24 months, and are merely a brief respite prior to another extended period of exponential improvement.

Exponential performance improvement is change with a vengence—and merely a harbinger of things to come. Careful examination of the long-term historical trends for just about any relevant technology one cares to name strongly suggests that the rate of change is *accelerating*—doubling every decade [1]. In other words, taken in the large, and over numerous disparate disciplines, the overall technical progress in each decade of the 21st century will be double that of the prior decade. If we set the last decade of the 20th century as the "benchmark" decade of technical progress, the period 2001–2010 will see the equivalent of 20 years of 1990s progress, the decade following, 2011–2020, will see the equivalent of 40 years of 1990s progress, and so on. In the first 25 years of this century we will experience as much technical progress as was seen in the whole of the prior century. When 2100 rolls around our technical mark will rest at the equivalent of **20,000 years** of unrelenting late 20th century technical progress.

When measured over a span of years, the overall rate of technical improvement is *hyperexponential*, collapsing into months what might have taken years just one or two decades ago. Hyperexponential progress is not merely "more" change—it is change of an order that obliterates everything

in its path in a breathtakingly short period of time. The very thought of hyperexponential progress calls into question the wisdom of executing programs whose lifespan, from conception to disposal, is more than a few years. Furthermore, it is not just our own national technical ability that is improving at this nearly inconceivable rate—our competitors and adversaries exploit the same technical elements as we.[1] When high performance processors, commercial grade switching gear, and spools of fiberoptic cable can be purchased with a credit card via the web or at your neighborhood "computer mall," then the barrier to entry has been reduced to nothing at all.

Hyperexponential progress has profound implications for all facets of system procurement, design, development, deployment, management, improvement, operations, maintenance, and modernization. From this perspective modern procurement practices are, in effect, superbly designed engines of delay and obsolesence whose sole goal is deliver a system that is irrelevant to the threats of the day, obsolete long before it is deployed, and horrifically expensive to sustain and operate. Battling hyperexponential progress is futile. Far better to embrace and exploit nonstop progress to improve systems. In other words, constant, unending, accelerating progress is the ally, not the enemy, of system engineers and program managers. I introduce here a new system engineering discipline—*raging incrementalism*—expressly designed to exploit hyperexponential technical progress to improve all aspects of ground systems from "lust to dust."

The remainder of the paper is organized as follows. Section II sketches some of the programmatic foundations of raging incrementalism. Section III outlines the software engineering principles that make possible rapid, highly-incremental system development in a manner that preserves system flexibility and ease of adaptation. Section IV remarks on the life cycle costs of raging incrementalism while Section V describes two ongoing experiments in the development of raging incremental systems.

II. Programmatic Principles

As a programmatic discipline, raging incrementalism emphasizes brief development cycles (on the order of weeks to a few months), rapid prototyping and experimentation, and the virtues of the "good enough." Hyperexponential progress is simultaneously terrifying and exhilarating (like riding a world-class roller coaster). One of its benefits and principal virtues is a neverending supply of "golden screws"—piece parts so powerful that they fundamentally reshape the process and practice of construction. Commodity processors, network routers, inexpensive, specialized add-on cards (such as megasample-rate analog/digital converters), open source software, and the like permit a small team to construct a substantial system in a matter of a few weeks or months. Rapid prototyping grounds the system in reality, forcing the developers to confront (or at least acknowledge) challenges of performance, function, or utility. Brief development cycles reduce the burden of program management and offer important benefits, including observable and measurable progress, lower costs, increased team cohesion, and greater customer satisfaction.

More subtly, brief development cycles and modest funding firmly bound expectations and near-term goals. In a world of onrushing progress, "*the better* is the enemy of *the good enough*," meaning that one can rely on hyperexponential progress to resolve those issues that are peripheral or incidental to the task at hand. A brute-force solution that leverages golden screws is probably "good enough" for now; the next cycle of development can focus on other germane improvements. In any case, the chances are excellent that someone else somewhere else sometime soon will solve (at no cost) the problem that your team has just set aside, and when that solution appears it can be quickly incorporated in the next incremental development cycle.

Rapid progress has many manifestations; in particular, the ascension of electronic commerce. This is relevant to raging incrementalism in a profound way, since it alters—forever and irrevocably—the provisioning of systems. Given the ever-astonishing advances in the price/performance of commodity hardware, it is ridiculous and wasteful to procure any hardware before it is absolutely required. "Just-in-time" procurement is as important to raging incrementalism as brief periods of development. When hardware is required, purchase it over the web and have it delivered to your doorstep. Doing so permits your project to have the most cost-effective hardware precisely when it is most needed. The engineering principles of raging incrementalism (discussed in Section III) ensure that new hardware can be transparently substituted for old as better-performance, higher-functionality components become available.

III. Software Engineering Principles

As an engineering discipline, raging incrementalism rests upon the exploitation of commodity hardware, open source software, open standards, peering architectures, and the architectural style of REpresentational State Transfer (REST) [2]. From a systems perspective the goal of raging incrementalism is to preserve system modularity and flexibility above all else—for the simple reason that if the previous increment has degraded the ability of the system to adapt to change, then succeeding increments will be more complex, more fragile, and more difficult to implement and deploy.

The underpinnings of raging incrementalism are the triumverate of commodity hardware, open source software, and open standards. The extraordinary explosion of inexpensive, versatile, high-performance computing hardware means that, for all practical purposes, computation is free and the hardware costs for most systems are so low as to be irrelevant.[2] Many bedeviling system problems may be resolved (or at least pushed aside) by brute force computation. Raging incrementalism dicates that—wherever practicable—problems should be solved in software, since commodity open source software can be rehosted at no cost and older hardware can either

[1] Consider the recent adoption by China of Linux as the "national operating system."

[2] A 1U rack server containing a 1 GHz processor, 256 MB of memory, 20 GB of disk storage and a 100 Mbs ethernet port can be purchased over the web for $500.

be repurposed or discarded outright.[3] Reliance on commodity hardware harnesses the power of hyperexponential progress to one's advantage. The pressure on commodity hardware vendors is harsh and unrelenting as they must accomodate improvements as quickly as possible or be forced into insolvency by competitors. Their suffering is your joy, since slightly older high-performance hardware can be purchased on demand at bargain basement prices—a perfect example of exploiting change for the purposes of incremental development.

Open source software allows system developers to harness change in another way. No one project, much less an individual, can pace hyperexponential change. However, "many hands make light work," and the burden of accomodating accelerating change is shared among the inhabitants of the "creative commons." The creative commons, that landscape of independent open-source developers, harnesses the firestorm of hyperexponential change by creating software piece parts, "golden bits," that revolutionize, in much the manner of commodity hardware, the process and practice of system construction. Software technology, like its hardware counterpart, is also characterized by an ever-shortening doubling period. However, while the software doubling period lags behind that for hardware, and is more reasonably measured in years rather than months, it is counterbalanced by the size of the community and the low (nonexistent) barrier to entry. Anyone with a personal computer and an Internet connection (of any sort) may contribute to the ever-widening pool of open-source software, and a plentitude of Internet resources (web sites and repositories) are devoted to open-source. Further, hyperexponential change guarantees that the doubling period for software advances will quickly (in the long view) shrink to timespans of few months.

Finally, the size and diversity of the open-source community guarantees that many developers tackle many problems in many ways in many places many times. While much open-source software is utter rubbish, the gems (Apache, gcc, Linux, MySQL, and Python, to name but a few), are powerful engines of progress whose effects are multiplicative. Raging incrementalism suggests that programs develop just the software that they need at just at the moment they need it and no sooner. Premature development is wasteful and needlessly duplicative, since someone else may solve your problem for you at no cost. In a nontrivial way, Google is a mighty mechanism of software development—searching for software is far easier and less time-consuming than implementing it.

Harnessing the diversity of the Internet-fed software ecology would be fruitless were it not for the co-development of open standards that are freely accessible and unencumbered by proprietary or patent claims. The fundamental protocols of the modern Internet, such as IP/TCP, DNS, and HTTP, are superb examples of open standards. These standards hide implementations behind protocols, and integrating systems based on protocols may be less difficult than integrating systems implemented as libraries. Protocols have many advantages, as they tend to be simple, transparent, and programming-language agnostic. Protocols standardize interfaces, and in a world of hyperexponential change, the interfaces—not the implementations—constitute the real value.

As an architectural endeavor, raging incrementalism rests upon two recent innovations, peer-to-peer constructions and REST. Systems are not amenable to rapid incremental change unless their constituent elements are decoupled. By this I mean that the system architecture must endure (nay, encourage) ongoing change. Without adequate decoupling, rapid incremental change is impossible, as insertion, modification, or deletion of components may destabilize the system to the point of failure or collapse. Peering architectures are resilent to commmon sources of system instability, such as the movement of resources, changes in membership, and variations in connectivity or provisioning. REST allows system designers to regard their system as a large finite state machine in which state transitions are transparent to cooperating peers. This transparency elevates, to a system level, many of the accepted techniques of industrial-strength object-oriented programming—including, but not limited to, delegation, mix-ins, filters, triggers, contracts, and behaviors. REST permits system assembly from large-scale piece parts (golden screws and golden bits) on an unprecendented scale. The modern world-wide web is a working example of the robustness and resilence of RESTful design and implementation.

IV. Life Cycle Costs

For many large-scale systems, such as space ground systems, the total system life cycle cost is dominated entirely by the operations and maintenance costs. In other words, irrrespective of what the system cost to procure, its lifetime operations and maintenance expenditures will swamp that. Raging incrementalism directly attacks life cycle costs on multiple fronts:

- Purchasing commodity hardware on demand minimizes the total hardware investment and *eliminates* depot costs, since replacement components may be ordered only as needed and delivered within hours to days
- Eschewing speciality hardware in favor of commodity hardware and software reduces time to delivery and forestalls obsolescence
- Relying on open-source software drastically reduces or eliminates licensing costs
- Hosting on interchangeable commodity platforms reduces training costs, since all platforms are largely interchangeable and repair costs evaporate because it is often more economical to replace the commodity platform outright than to repair it
- Locking customers into a vendor is impossible when both hardware and software are fungible—sole-source procurements become a quaint anachronism
- Sustaining systems is incremental and upgrades can always take advantage of the most cost-effective commodity hardware and open source software

[3]Google is built entirely on inexpensive commodity hardware running open source operating systems and utilities. Individual Google clusters contain thousands of hosts and when the hardware fails it is more cost-effective for Google to consign it to the trashbin than to repair it.

Raging incrementalism attacks costs at all points in the system life cycle from concept to disposal.[4]

Deployment comes early and often in the life cycle of a "raging incremental" program—in fact deployment is just another (small) repeated incremental milestone in the program's history. From that point forward raging incrementalism guarantees that the system can be replaced, augmented, expanded, contracted, or encapsulated element-wise. Since no one element of the system is expensive (recall that open-source software is free and commodity hardware is essentially free) the sustainment costs are modest and—in a world sustained by raging incrementalism—the system may be replaced (at a cost/performance ratio that is improving hyperexponentially) however the customer wants whenever the customer chooses at whatever pace the customer desires.

V. Summary

In a world of hyperexponential change systems that fail to acknowledge and exploit change are obsolete long before they are deployed. Raging incrementalism embraces change as a fundamental driver of system design, construction, deployment, and sustainment. To demonstrate the utility and efficacy of raging incrementalism we are developing two demonstration systems for launch range operations. The first is a distributed countdown clock service designed to replace the obsolete timing and countdown infrastructure of the ranges with a peering system constructed entirely from commodity hardware and open-source software. The second is a network-centric range video system to replace an aging video plant supporting hundreds of cameras with one capable of managing thousands of cameras with greater flexibility and dramatically lower costs.[5] We hypothesize that the "raging incremental" systems outlined above will prove to be more cost-effective and sustainable, in all respects, than the systems that they replace.

References

[1] Ray Kurzweil, *The Law of Accelerating Returns*, March 7, 2001, www.kurzweilai.net/articles/art0134.html?printable=1.

[2] Roy Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California, Irvine, 2000. See www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[4] Recycle the commodity hardware by donating it to a local grade school or the nearest third world country.

[5] For example, the raging incremental video system replaces an obsolete and expensive ($>$ \$1,000,000) video archive storing less than 13 camera-hours of video with commodity hardware ($<$ \$15,000) storing more than 4,000 camera-hours of video. The price/performance of the legacy archive system is approximately \$77,000/camera-hour while the raging incremental replacement is less than \$4/camera-hour.