# Opinion—Stay on course with an evolution or choose a revolution in computing

Ramesh K. Karne, Alexander L. Wijesinha, and George H. Ford Jr.,
Department of Computer and Information Sciences, Towson University, Towson, Maryland.

## *The Nature of the Problem:*

Since the inception of personal computing in the early 80's, the evolution of computing resources has created a myriad of software applications, programming languages, operating systems, platforms, and execution environment sensitive applications. Many of the developed hardware and software system components have come and gone before they had any longevity allowing return on investments in the technology by the market place. In some instances, the changeover in technology has met or exceeded Moore's Law in the swiftness with which the fleeting technology has appeared only to be replaced. This has created a tremendous obsolescence in software, hardware, and people's skills. The effect on software development has been an endless requirement for additional training and retraining of personnel on the use of new tools and software environments (as shown in Figure1). The long term effect is a dearth in efficiency of people's time, and capital investments and the ability to use time, people, and investments to solve new and interesting problems left unchallenged.


**Figure 1. Evolving Computing Environments**

Consider the disposal of millions of personal computers and other computing devices [6] when new software is incompatible with legacy hardware. This lack of backward compatibility has necessitated the disposal of hardware and software to maintain compatibility with new software, due to a lack of support for prior released hardware and software. In many cases, new software systems have been developed based on sprawling architectures which have ignored the "divide and conquer" principles supporting good software designs. Instead, software has grown to demand a huge consumption of hardware resources – expecting more memory, secondary storage, and increased speeds in processing power. Instead of programming for efficiency and re-use, developers have fallen into a habit of "programming in the huge," where application and operating system solutions have grown to expect that faster, cheaper, hardware will be created for their resource hungry application hosting needs. Instead of spending time to rework the architecture, analyze it, efficiently integrate to it and re-use existing software, often new designs

simply create and add to an expanding system base. This new system base often antiquates its preceding marketplace solutions.

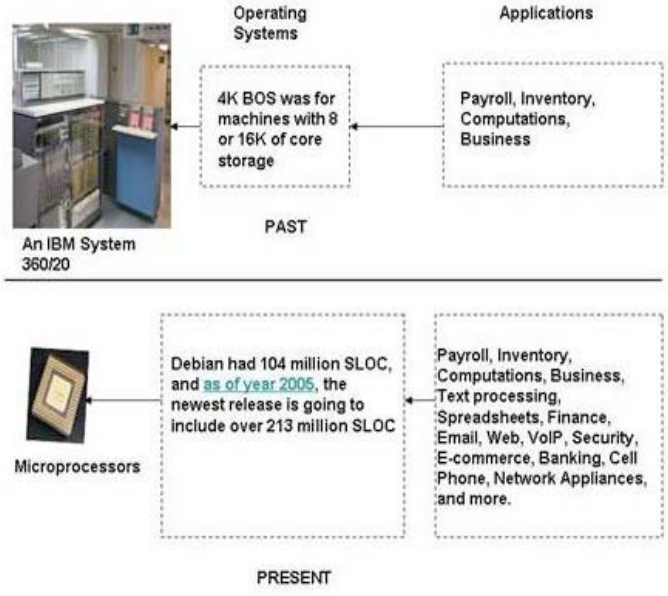


**Figure 2. Historical Progression**

## *The Prognosis as it Appears Today:*

There is no solution in sight to address the above problems. As shown in Figure 2, the evolutionary path in computing technology continues with rapid turnover of products and new releases. This evolutionary pattern in software system development appears to have little regard for efficiency and planned application growth. For instance in an 18 year period from 1985-2003, Microsoft released 15 major versions of operating systems [5]. It appears that development has succumbed to the habits of today's "throw away" society, trading efficiency, architecture planning, controlled-growth, and controlled investment expenditures for the latest, greatest, fastest, resource-guzzling, shiny, new toy available in the marketplace. The evolutionary approach has wreaked havoc in the consumer marketplace, wasted natural resources on throwaway system components, and wasted time on retraining of developers and integrators of systems. In addition, according to experts [4], the complexity in information technology is mainly caused due to rapid changes in operating systems, applications and computing environments. It is time to take a revolutionary step of foresight in planning the development of software system advances in a manner that exhibits good software engineering practices.

## *How did we get in this Dilemma?*

Current trends in open source systems, ubiquitous computing, and Web technology have created expanding variety of products which can be applied to solve a given consumer software system need. The lack of coordination and planning towards such solutions has increased the speed of obsolescence in development efforts provided in prior system solutions. Indeed, the vast set of applications which could be used to solve a consumer problem may leave the person with a mind boggling and confusing choice of what product to buy, while guessing as to the product's compatibility with their computing system. One aspect of this exasperation found in today's information technology development can be attributed to the focus on a variety of evolving host

environment platforms rather than on the purpose for an application in solving a consumer problem. Environment influences may stem from operating systems, programming languages, protocols, and layers of programs, tools, and hardware platforms as shown in Figure 3.

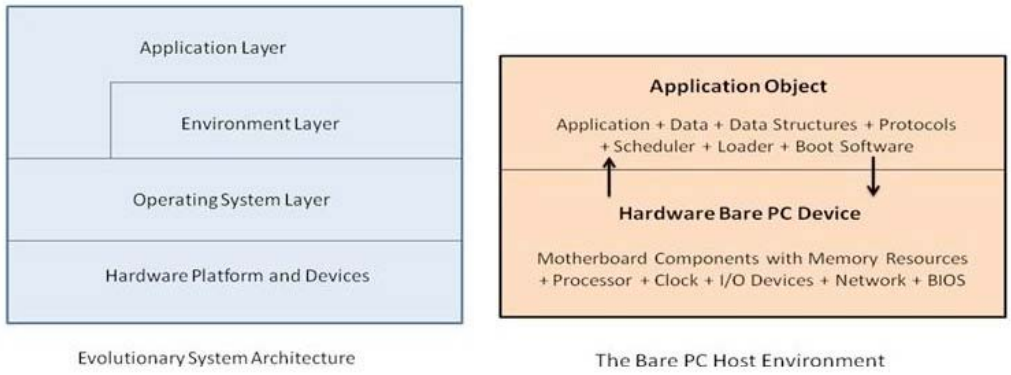


**Figure 3. The Evolutionary Architecture compared with the Bare Machine Architecture**

As an example, a Web server is an application which, ideally, should be independent of any computing environment. However, the Microsoft's IIS (Internet Information Server) server requires an environment to provide a Web server application, including: an operating system, Web support tools, and a client protocol that is compatible with an Internet explorer (IE) or a Netscape browser. It may also depend upon the tools and libraries used to create it, which depend upon the operating system host and software drivers specific to the hardware platform. This is not unique to IIS, but also applies to other web servers and services also. We choose this simple Web server application to illustrate a typical computer application that clearly depends on some designated computing environments, and is likely an application which is familiar to the reader. Ideally, a web server, such as IIS, would be able to run within any operating system host on any hardware Intel compatible platform produced to date so that all consumers might benefit from use of the product.

Most contemporary computing applications share this dilemma related to development and execution environments, which directly relates to application evolution requiring adaption for new platforms. The real-world computer application and computing environments are increasingly intertwined to create a system that dynamically changes and becomes obsolete as its underlying environments rapidly evolve. As in our example, when a new release of operating system arrives, then there is a need to get a new version of an application, such as a web server like IIS. Similarly, when there is a new version of protocols and standards, like HTTP (hypertext transfer protocol) or a PHP (preprocessor for hypertext), then there is a need for a new version of the IIS.

The list of products and versions for each environment is expanding without limit over time. Often, the consumer will find that several versions of a given product will be released before the user can comprehend the various options and nuances for a product. Today, this can be easily observed in the developer environments such as Java, XML, ASP, JSP, DCOM, CORBA, Windows, Linux, and so on. As software development professionals, we should be concerned about learning and building computer applications. Unfortunately, we have less control over the evolution of computing environments, as listed above, which apparently have little or no stability and even less clear direction in their future.

### *How do we change rapidly evolving technological paradigms?*

We believe that the root cause of this problem is intertwining computer applications with computer environments. The most pervasive computing environment is the operating system. Operating systems were primarily designed to facilitate computer programmer development efforts, providing hardware abstractions and shared resources to facilitate multi-user and multi-tasking environments. It should be noted that there are many other capabilities and features provided by the operating systems that are not relevant to our discussion.

Operating systems have become large, complex, and inefficient due to an increasing demand upon the operating system to provide services and capabilities for applications. Researchers successfully tried to eliminate some level of operating system abstractions to achieve higher throughput [1]. One of the ubiquitous design approaches to operating systems has been a layered approach to the kernel core and its associated services and internal applications. This invariably produces a foundation which is subject to obsolescence of its myriad interacting components over time. When a single underlying layer changes, it has a ripple effect on other layers resulting in a need for new software.

However when an operating system is totally avoided, it results in a new computing paradigm, where applications are *Self-contained, Self-managed,* and *Self-executed* (SSS). This may look like an old paradigm, as initial computers did not have any OS. An application that satisfies the SSS properties does not require any resident software in the hardware to load and run its application. A given application can be made to arrive through a network or be resident in a flash drive, load itself into the machine, control its execution, and exit the processor platform when the execution is complete. There is no need for a kernel or OS support for such applications. That is, a computer application can be made to be independent of any environment. Using this paradigm, a Web server designed once should be able to run in any device including: PC, laptop, handheld, cell-phone, or network appliance provided they are based on the same CPU architecture. If we followed such a computing paradigm, this would be a revolution in computing technology and it would reduce the speed of obsolescence in software engineering development efforts dramatically.

In order to achieve such a revolution, we need a fundamental change in our approach towards building computer applications. It would require a different mindset for both application development programmers and manufacturers. When the operating system is avoided, the hardware must be totally "bare", i.e. there is no OS, nor mini/micro/nano kernel, nor any software that is resident in the hardware storage or preloaded. Such a computing device can be referred as a "bare machine". This bare machine could be any pervasive device or a computer system.

The purpose of the bare machine is to provide an execution basis for an application and without the need for the machine to provide any persistent storage on behalf of the application. The application carries its own application program along with its data storage and execution control to possess SSS properties (it could also carry its own basic input output system (BIOS) that is relevant to the application). Such an application directly interacts with the bare machine to load and run its program and exits the machine when it is complete. In this paradigm, such

applications are referred to as an Application Object (AO) [3]. As the necessary operating system is dispersed into an application, this execution environment is also referred to as a Dispersed Operating System Computing (DOSC) paradigm [2].

### *Benefits of the Bare Machine, Application Objects, and DOSC Approach:*

When all computing devices (pervasive devices and computer systems) are bare, then an application, such as a Web Server, can be designed to run on any computing device. We can write a Web server application once and reuse it, extending it (when desired) using an object-oriented paradigm. In order to achieve this, we also believe that we need to use a common computer architecture (which could be an Intel 386, i.e. IA32, or above) as the bare machine, and a common development computing language (perhaps C++) for software creation. This potentially makes the programmer's job harder, as now he/she must be both a systems developer as well as an application programmer. However, the result is that there is only one Web server AO in the world, which can be reused and customized, by extension thru inheritance, by developers and users for their need.

While this, at first, may sound absurd: how can a single CPU architecture be used in all devices? We suggest, why not? The hardware is becoming cheaper as the number of computing machines reaches a larger base of consumers. Under this DOSC approach, the required AOs will be designed to run on this common platform of bare machines. If you need more power, one could add more bare machines or get a new machine with faster speed and more memory. An advantage of this approach is that the performance gains are achieved through technology speeds instead of tweeking the software and hardware for temporary performance gains due to a changing environment. The scalability required for applications will be gained by adding more homogeneous hardware chips (designed to extend and enhance the same architecture). The AOs would be planned and designed to work with scalability as a fundamental requirement. While AO enhancements will be done through object extensions, older AOs would still be supported to run on old hardware, without making the hardware and software obsolete.

In such revolutionary approach, information technology development will be focused on application construction instead of developing environments. The innovation and efforts will be dedicated to development of new real-world applications such as a Web server, or some novel application to solve new software challenges, instead of focusing skills and talents to chase the latest developing environments and expending time and effort to re-host old applications.

When computer devices become bare, Application Objects will be capable of being carried on a flash drive and people can use them at their home, the airport, their office, the mall, or anywhere there is a computing device. As the bare device has no need for resources to be owned by single user, there would be no need for expensive and unique resources in the device. By carrying their own flash drives or portable storage devices with their Application Objects, users would also enhance the security of their personal data.

When this computing paradigm becomes feasible, the information technology industry will be polarized on applications. When industry establishes its focus on application development, then software productivity will reach a theoretical maximum. For any given application software effort, there would be a focus on only one development effort involved to create and enhance a

given Application Object. For example, the Web server development will be done by only one development group or organization and rest of the world can reuse it. Similarly, other Application Objects can be developed by other organizations that can be hosted on a network server to be reused by all users.

In this paradigm, an AO developer may charge a nominal fee for the ownership of the object and its reuse. The developer of an AO will be responsible to provide extensions and enhancements needed for new application domains and shifts in the paradigm for the use of their application.

## *Experimenting with the Future: Computational Application Objects on Bare Machines:*

To study the bare machine concept and its use, we have started a bare PC computing laboratory, where we have developed numerous applications including: a Web server, a secure VoIP softphone, an Email client, an Email Server, an Editor, a Web client, a TLS capability, a FTP application, and various other network applications. We have constructed these AOs and have successfully executed them on a bare PC without the need for any operating system during their execution. These AOs have successfully run individually on a bare PC, and it has been observed that they can be combined together to run as a single AO – sharing data. If one needs an email capability in a Webserver, then a single AO can be constructed for running this application.

These applications are network centric, and network aware. The use of these AO based applications does not require local hard disks, nor do they require any other local machine persistent storage. Most of the bare PC applications fit on a single floppy disk or can be placed in a flash drive. These applications run on a bare PC which simply has standard BIOS with no additional software. The applications themselves directly communicate to the hardware and carry their own boot software, loader, scheduler, process manager, and other code needed to run the application.

Our current research demonstrates the feasibility of the proposed revolution in computing. It demonstrates that such systems can be built to last forever. Imagine that every computing device could one day run applications without depending upon any operating system or other environments. This will bring a major revolution in the information technology industry that will produce products that stay longer and reduce electronic consumer "throw away" habits, while encouraging reuse and providing global software advances. The software development "brain trust" will be able to focus on solving new challenges and problems, instead of wasting time and effort on re-hosting their older solutions caused by chasing platform changes.

### References


[1] D. R. Engler and M.F. Kaashoek. Exterminate all operating system abstractions. In *Fifth Workshop on Hot Topics in Operating Systems*, p. 78, 1995.

[2] R. K. Karne, K. A. Venkatasamy and T. Ahmed. Dispersed Operating System Computing (DOSC). In *Onward Track, OOPSLA 2005, San Diego, CA,* October 2005.

[3] R. K. Karne. Object-oriented Computer Architectures for New Generation of Applications. In *Computer Architecture News,* Vol. 23, No. 5, pp. 8-19, December 1995.

[4] Complexity Is Killing IT, Say Analysts. IDG News Service (04/13/07) Krill, Paul, http://www.techworld.com/opsys/news/index.cfm?newsID=8525&pagtype=all.

[5] http://en.wikipedia.org/wiki/History_of_Microsoft_Windows.

[6] http://www.usatoday.com/tech/news/techinnovations/2006-07-30-ecyling-oregon_x.htm.