BMC Principles

Obsolescence and Security are the two most daunting problems facing in today's information systems (IS). Both problems are created by design in the current evolutionary computer architectures and systems. Conventional solutions are further aggravating obsolescence and security vulnerabilities over the years. We need a revolutionary and unconventional ideas to address these problems. Perfect solutions lie in our history of computing, where applications were running on bare hardware.

According to our studies and investigations, operating systems (Oss) are the root cause of the above problems. When OSs are eliminated, information systems pyramid collapses into a flat model resulting in applications and hardware that eliminate all computing environments. When computer applications directly communicate to hardware, most of the security vulnerabilities disappear in addition to minimizing obsolescence. This approach results in a bare machine computing (BMC) paradigm.

We believe that there are ten most significant engineering principles (EP) in BMC paradigm:

EP1. Universal Computing Unit (UCU): When there is only one processor complex type, all applications will use this unit and eliminate all heterogeneity and duplication. When new technology arrives, it must use the best options at the time to implement a UCU (More details on the UCU described later).

EP2. Bare Computing Device: When a computing device becomes bare, it does not have any valuable resources. This device can be physically secured by an owner at a given time, and many users can use the same device one at a time. This is an owner centric approach, where a given owner physically controls the device while it is running.

EP3. Bare Machine Computing (BMC) Paradigm: Use the BMC paradigm [44] to build end-user applications. It is a two-prong approach, computing device is bare, and applications follow BMC programming methodology in addition to BMC principles.

EP4. No OS: An OS is a breeding ground for multiple layers, heterogeneity, and it is a root cause of obsolescence. When an OS changes (frequently as mentioned before), its related programs and applications need to be ported to the current version. This causes previous applications to be obsolete. When an OS is avoided, it results in a flat model, avoiding hierarchy in computer and information systems. This helps reduce obsolescence and results in longevity in computer systems.

EP5. Direct Hardware Interfaces (DHI): A UCU provides direct hardware interfaces to domain-specific applications. These applications are independent of computing environments and platforms. Some of these direct hardware interfaces are shown in [31], which were written in C++/C/ASM code. These interfaces were used to program bare machine computing applications

and systems over a dozen real-world applications. Thus, we claim that these interfaces code can be pushed onto the processor thus avoiding all device drivers and related obsolescence.

EP6. Single Programming Language: When we adopt a single programming language (e.g. C++/C) to build applications, it eliminates all language-based heterogeneity and redundant applications. Higher-level language abstracts may be implemented as macros. In our opinion, all programming languages eventually must be mapped to an underlying instruction set architecture (ISA).

EP7. Domain-specific Applications (DSA): Perform higher levels of abstractions on end-user applications across all domains. For example, email is a domain-specific user application; its implementation is different in smartphones, laptops/desktops, and other devices. We must build a common email end-user application with different user interfaces. This will eliminate redundancy among similar type of applications and thus reduce obsolescence.

EP8. Standard I/O Interfaces (USB): At this point, we use a USB as a standard I/O interface for all devices in our BMC applications. Provide USB interfaces on the UCU, along with a built-in host controller. This eliminates all diverse interfaces and reduces obsolescence.

EP9. Standard Instruction Set Architecture (ISA): Define standard instruction set architecture and extend it as needed. There is no need to discard the previous models. One can pick a latest ISA from Intel or similar processor types and use it as a baseline or a building block for future applications. The performance of processors can be realized using the latest technology at the time. One can build higher level macros in assembly to implement new instructions for a customized application. We have used this approach to build direct hardware interfaces for a variety of applications.

EP10. Object-oriented Methodology: Every component and system must follow an object-oriented methodology throughout its life cycle. This must also include hardware. This implies abstraction, encapsulation, inheritance, extension, reuse, and polymorphism.

The following observations justify the above ten engineering principles:

1. Device drivers are constantly changing due to new versions of OS, hardware, and planned obsolescence. They also vary due to heterogeneity in vendors, architectures and device implementations. If we adopt all device drivers to be based on USB standard, all pervasive devices can be standardized. The USB device controllers can be pushed on to the processor chip and provide standard instructions to access devices. These interfaces can be referred to as direct hardware API, which can be used by application programmers, eliminating all diverse device drivers. This will simplify building computer and information systems and also dramatically reduce obsolescence.

2. Currently, there are a variety of programming languages and corresponding applications. A given end-user application may be written in a variety of languages and designed to run on different platforms. At the end, these end-user applications written in different platforms perform similar functions with very little differences. Why not agree on a single

programming language (known as a computer language), which will eliminate heterogeneity in end-user applications. This computer language can be based on C/C++/Assembly for now, as it is closer to hardware, and it can be extended to cover other programming language constructs by developing macros. The C/C++ compiler can be extended to cover the programming language extensions thus eliminating heterogeneity in programming languages.

3. Similar to a common computer language, it is also possible to devise a universal processor unit (UPU) that can be used across all computer or information systems. The UCU can be manufactured in a large scale to make the price in pennies. One or more UCUs can be connected in parallel to yield in scalable performance. This approach will result in a major building block that serves as a basic unit to build computer and information systems. The UCU can also be logically extended in object-oriented manner to reduce obsolescence.

4. Most of the computer applications have common programming constructs and implementations that can be abstracted to be generalized. These common constructs such as sorted arrays, hash tables, switch and case statements, networking primitives (send, receive, status checking), data buffers, control buffers, and so on can be implemented in the UCU to reduce programming efforts using a single programming language as mentioned above.

5. Today's Internet is monstrous and global in nature. This can be divided into two areas including secure and non-secure domains. It can also be further divided into domain specific applications such as manufacturing, banking, marketing, government, countries and so on. All nodes on the Internet must be bare and physically secured to provide ultimate security on the Web. Current Internet security is unsolvable as it is based on Globalization and accessibility to billions of users.

6. All components in computer and information systems must adhere to strict object-oriented principles including encapsulation, inheritance, polymorphism, reuse, and extension.

7. Obsolescence resiliency must be a mandatory design requirement at all levels.

8. Planned obsolescence must be a crime and it must be severely punished and abandoned.

9. Innovation and novelties must focus at application level. Performance improvements should be obtained using the evolving technology without destroying the basic building blocks such as single UCU, single computer language, no heterogeneity, no layers or middleware, and so on. There must be strict adherence to engineering principles to build computer and information systems.

10. All hardware blocks must be bare, no intelligence in hardware, strictly and physically secured by an owner.

11. In addition, the BMC paradigm has some hidden treasures in computing that have not been observed and seen by many researchers at this point.

12. The hidden treasures in BMC are analogous to PC revolution started in 1980s.

13. The current trends in computing and information systems are based on industry revenues, there is no stability in our current systems and trends.

14. It is possible to build computer and information systems that can last for a century by following the above ten engineering principles.

15. The benefits of the BMC paradigm and its implications on society are enormous including economy, people skills, longevity of systems, avoiding recurring costs in technology, and so on.

16. Finally, one must accept the problems facing today in obsolescence and security in IT, and willing to change the current course to build computer and information systems like historical monuments and structures. In our experience and research, the BMC paradigm addresses the above problems and provides a solution to build sustainable systems.