No Warranty Express or Implied: Why do We Have so many Problems With the Computer Systems that Pervade our Lives?

John W. COFFEY
Department of Computer Science
The University of West Florida
Pensacola, FL. 32514

ABSTRACT

Computer systems, large and small, are everywhere. From the 100+ electronic control units in a modern car to mobile devices, to tablets and desktop computers, to petabyte databases that are mined for information, computers pervade our lives. When any factor in our lives becomes so pervasive, a range of problems will certainly follow ranging from basic frustration and inconvenience, to lost productivity, to losses due to using the devices apart from problems with the devices themselves, to loss of life. This article explores the unique role of computers in our lives from the perspective of their complexity, limits on our ability to ensure that systems are built without errors, tradeoffs inherent in the design of computer systems, and measures that can be taken to address these problems.

Keywords: computer systems, complex artifacts, computer program correctness, usability, security, technical workforce, software engineering

1. INTRODUCTION

Computers are everywhere. Upwards of 100 different electronic control units in cars control the engine, air bags, speed of the vehicle, provide diagnostic capabilities, and more. All mobile devices are computer systems that interact with the world through other computers. At work, a large percentage of people use computers. A report from NCES [1] indicates that the percentage of people using computers at work grew from 46% in 1993 to 56% in 2003. By 2014, 71% of all Americans used the Internet.

The range of problems that occur with computer systems is dizzying. Computer programs freeze up in the middle of work, printers stop working, voices mysteriously stop transmitting in smartphone calls, major companies experience disruptive system failures, shocking security breaches take place in top companies and at the highest level of government, and the list goes on. Through it all, users agree to licensing agreements that provide absolutely no warranties for losses due to use of the software or remedies when such events occur, an astonishing state of affairs.

The remainder of this article contains a description of why this state of affairs exists. The discussion addresses why computer systems are some of the most complex artifacts ever created, limits on the ability to show conclusively that they have been constructed correctly, fundamental tradeoffs in how they must be designed, and a brief case study in large system failure. The article concludes with some indications of what can be done about these problems and with some conclusions.

2. COMPUTERS AS COMPLEX ARTIFACTS

In his seminal paper on software engineering, “No Magic bullet,” Brooks [16] made the distinction between what he termed accidental and essential complexity in software development. Accidental complexity arises from immaturity in the field and a lack of software programming languages and development tools that foster production of good software efficiently. Brooks states that, over time, languages and tools improve and accidental complexities recede.

Much more troubling are what Brooks termed the essential complexities of software development. He states that software development is subject to numerous essential and unavoidable complexities. There are complexities stemming from the large size of computer programs and the combinatorial explosion of ways components can interact. Software is complex because of its abstract, invisible nature and because of the fact that software is created to address a huge range of tasks. While it is very easy to change the way software works (and all competing software companies and open-source developers continually do so), it is equally easy to introduce new errors when making changes and it is correspondingly difficult to understand fully all the implications of making ongoing changes to large programs.

Integrated hardware and software systems are arguably the most complex artifacts humankind has ever created. OS 360, first released by IBM in 1964 as the first truly multi-tasking operating system [7], was comprised of more than 1,000,000 lines of assembly language code, and it had 10,000 documented "features" or bugs. Every attempt to fix a coding error produced additional errors, many of which were worse than the error the programmers were trying to repair.

It is estimated that the Microsoft Office software suite has between 30,000,000 and 60,000,000 lines of code [8]. Significant portions are essentially "zombie code" (typically unused subprograms) that cannot be removed because some other isolated piece of code might occasionally call it. By contrast, an open-source competitor to Microsoft Office, Libre Office, weighs in at a relatively lightweight 12,500,000 loc. It is very difficult to imagine what 12,000,000 of anything looks like.

The Windows 10 operating system is estimated to contain 50,000,000 lines of code. It is Microsoft's policy to push out code patches on a regular basis. The patches take two forms – bug fixes and security patches. Introductory software engineering courses universally bemoan the problems that come with a "code and fix" approach to software evolution, characterizing it as an "anti-pattern," the exact opposite of a good programming practice [18], for software development.

While software itself is extremely complex to develop and maintain, separate problems come from the complexity of the human-computer interface (HCI). Software companies have competed for decades by adding features to software. Overly complex HCIs cause frustration and loss of productivity (in the best case) and lead to serious human error with significant consequences in the worst case. With the advent of GUI-based windowing systems such as MacOS in the 1980s, predictable locations for basic functionality such as opening and saving files simplified human-computer interactions. With the advent of Web-based interfaces, often designed to appear artful at the expense of functional, any semblance of HCI uniformity is gone. Usability suffers correspondingly, and frustration at the inability to perform simple tasks is pervasive.

Two fundamental strategies (one for hardware and another for software) are utilized to maximize the likelihood of proper execution, or in the worst case, to allow system performance to degrade gracefully and end in a controlled way in the presence of a catastrophic error. For hardware, redundant systems with uninterruptable power supplies are used. However, building redundancy into systems is expensive and hence, many systems have single point of failure vulnerabilities. For software, the concept is "defense in depth" which dictates that software should have a series of defenses so that, if an error cannot be successfully handled at one level, there are additional levels of defense to handle the problem.

Exception handlers are a basic capability to provide defense in depth for software errors. Design of exception handlers is complex, and many programmers actually create anti-patterns in place of good design patterns for exception handling. In the some cases, such poor programming practices can lead software to behave incorrectly or to bomb in the presence of errors, and in other cases, they can open up security vulnerabilities.

3. LIMITS ON THE ABILITY TO PROVE PROGRAM CORRECTNESS

Computer programs are based upon logic used to implement algorithms. A great many aspects of computer programs are amenable to proof techniques that enable developers to show conclusively that some condition holds. For example, in many cases, the time a program will take to process a dataset can be determined either a-priori or empirically. Proof techniques can be applied to demonstrate program correctness for small programs. Unfortunately formal methods are mathematically intensive to use and do not scale well to large programs.

Interacting programs make matters even worse. A modern architectural design pattern for large software systems is service-oriented architecture [11]. Service-oriented architectures seek to combine different pieces of program functionality (often realized in separate programs) into a composite application. It is a new way to tie together different legacy software systems that were often stand-alone applications before Web-based apps became so pervasive. As distributed applications with different functions running on different machines and data being accessed through message passing, a great many things must go right for the result to be right, and very small esoteric problems can cause the system to fail. No scalable formal methods exist that are generally accepted to show the correctness of such systems.

4. USABILITY AND SECURITY IMPACTS

The usability of a device refers to how readily the user can achieve the goals of use of the device. Studies in Human-Computer Interface (HCI) and usability have been ongoing for decades. In 1998, Hartson [22] stated that HCI work has the goal of making software use efficient, effective, safe, and satisfying to use. He discusses both ease of use and usefulness as equally important aspects of usability. With the advent of graphical user interfaces on standalone computer systems, some consistency started to emerge in software interfaces. However, as early as 1998, Hartson appreciated the lack of standardization in Web interfaces and the isolation of WebApp users who could not necessarily ask a colleague nearby how to perform a particular function.

Technical support is also often problematic. Help capabilities associated with large-scale software programs routinely return voluminous, mostly irrelevant results following a help query. Tech support by live humans is usually expensive when it is available. Constantly changing software ensures that user documentation is constantly lagging behind the actual program and unreliable. Web sites are often designed with an eye to aesthetics at the expense of usability and often come with no tech support at all.

Usability is also impacted by ever-increasing needs for security. The evolution of security concerns since Robert Morris unleashed the first worm on the Internet in 1988 [15] have been profound. Two areas that illustrate this evolution of concerns are the tradeoffs inherent in ease of access and use and the accompanying security concerns that are raised, and in the evolution of viruses and anti-virus software.

Donaldson [14] states that security professionals continually have to make decisions regarding the tradeoffs among security, privacy, and convenience. He states that very broadly, people and organizations have opted for convenience over privacy, at the expense of raising new security concerns. As an example, the decision is made to put mission-critical data in the cloud, possibly improving security in some ways (reliable backup so it will not be lost) and possibly worsening security in other ways (the CEO wants to access the data through an ever-evolving series of devices with unknown security vulnerabilities).

Recent successful hack attacks such as the attack on the 2016 Democratic National Committee [17] are typical of this concern. Best evidence is that state actors from Russia had been inside the DNC network for up to a year before the attack was made public. Attackers utilized a "software implant" which behaves as a bootkit which the computer system BIOS launches on system start. A successful implant might have the status of a privileged background process passively inspecting data or actively injecting malware payloads [18].

The outlook for cybersecurity is not likely to improve [19]. Dishman reports that since the successful 2013 attack on the Target Corporation, JP Morgan Chase, Scottrade, UPS, Goodwill, Sony, the FBI, Experian, and others have been attacked successfully. The average financial cost of a successful data attack is almost $4 million and rapidly rising. The financial cost to clean up after a successful security breach does not include intangible costs of bad publicity, customer concern that using the capabilities after the cleanup might entail new risks, etc.

Earlier antivirus software relied on detecting signatures that indicated malware, and antivirus software processed a rapidly growing collection of these signatures. In the early days, there were not a great many different virus types and new ones did not appear that quickly. Furthermore, they propagated via removable media such as diskettes rather than over the Internet. Today, viruses are polymorphic, making them able to generate many variants of themselves that can defeat signature-based virus checkers. When detecting them by traditional means is possible, it is at the expense of consuming significant system resources in monitoring. Slow systems quickly become a usability/convenience issue. A consequence of the multiple layers of security that are put in place is that computers that are unimaginably more powerful than could have been envisioned decades ago sometimes run more slowly than machines from those eras.

The supply of computer professionals is also problematic. Historically, the United States has had large shortfalls in the number of young people wanting to go into computer-related fields. Many who go into the field in school do so under the misapprehension that ability to use computer systems (such as their smartphones and video game consoles) will translate into success in Computer Science curricula. A consequence of this inaccurate perception coupled with low levels of preparation in math and computer classes in K-12 has meant fewer entrants to the field at the University level, higher than desirable rates of attrition, and significant shortfalls in the production of qualified computer professionals.

5. A TYPICAL COMPUTER PROBLEM

The landscape of potential examples of problems with computer systems is truly expansive. As of summer, 2016, a major US airline was in the middle of an extremely high-profit time due to the lingering effects of low fuel prices. In August 2016, a major computer system lost power and backup systems failed to come online [4]. The systems were down for more than 6 hours, leaving passengers stranded for much longer times than that. The airline was ultimately forced to cancel more than 1500 flights, and experienced major problems over the course of several days. Some weeks before, essentially the same thing happened to another major carrier. In the past year, three other carriers also had computer-related disruptions.

The large-scale computer systems employed by these airlines are called Enterprise Resource Planning (ERP) systems [12]. They are typically aggregates of multiple systems, some decades old that have been retrofit to work together over the Internet. Most involve a great many layers of software (called virtual machines) that execute on distributed physical hardware. Such systems are case studies in what can go wrong with complex interacting programs [6].

The airlines and many other large companies struggle with very old, mainframe-based, legacy systems. Noffsinger et al [25] describe legacy systems as ranging from some that are well maintained to those that exist with only a program codebase and data, obsolete or non-existent technical or user documentation, or even executable code only with no source code. In the latter case, maintainers modify the machine code directly, and it is impossible to know about limits coded into the data until a limit is exceeded and something breaks.

Anytime one sees a "character mode" computer screen in a business or governmental agency, it is probably the front end to a legacy system. They are still pervasive. The leaders of some companies understood that solving the Y2K problem was an opportunity to migrate away from legacy systems, but many perceived such a migration as simply too expensive and disruptive. Weighing the prospect of mass-extinction due to a failed, unthinkably expensive migration to a new system with a slow death by 1000 cuts inflicted by maintaining the old system, many companies chose to keep cutting [28].

6. WHAT CAN BE DONE?

Obviously, such a multi-faceted problem must have a multi-faceted solution. Despite all the difficulties, ongoing efforts are being made to address the wide range of problems with computer systems. While problems exist on a great many fronts, underlying them all are an enumerable set of root causes including:

- an insufficient and insufficiently educated technical workforce
- difficulties replacing massive, mission-critical legacy systems
- a moving target due to the rapid proliferation of IT that potentially has serious safety or security design flaws
- a large number of individuals and state-sponsored groups trying to exploit system vulnerabilities
- a naïve belief that more technology cures all problems

The US Bureau of Labor Statistics publishes the Occupational Outlook Handbook [25], which cites more than 1,400,000 computer specialist jobs in computer and information technology. Most of those jobs are experiencing rapid demand growth. On the supply side, the amount of exposure to computer-related courses has been variable in K-12, but has tended to be low. Some students go through K-12 with only a few end-user courses in computers. Some technical high schools have a full slate of computer-related classes, but they are the exception.

Initiatives to foster interest in the STEM disciplines are increasing in the United States, particularly for under-represented groups such as women. According to the U.S. Bureau of Labor Statistics, women earn approximately 18% of the technical degrees awarded and comprise 25% of the computer and information technology work force. Eleven percent of Tech executives are women [26]. The source of the problem is well known – in the absence of consistent mentoring and encouragement, young women lose interest in technology at a very early age – often as early as middle school. Efforts to foster more participation by women have been ongoing for decades. Advances in how to encourage women hold some promise.

If the cohort of computer and information technology professionals cannot be homegrown, the only other option is to import them. The H-1B visa program provides 65,000 visas per year for foreign national graduates of the US university system plus 20,000 extra for people who have earned a Master's or Doctorate. The H-1B program is restrictive; one can only be obtained via a sponsoring employer and is lost if employment with that employer ends.

Furthermore, the relatively small number of visas is for all "specialty occupations" including but not limited to all of STEM, medicine, social sciences, education, law and business. The Computer and Information Technology field alone could consume all of the H-1B visas in any given year. Tech companies nationwide have long clamored for liberalization of the H-1B requirements in order to improve their prospects of recruiting the talent they need.

Improving security training for both software developers and system administrators is a rapidly growing concern and endeavor. Introductory programming courses are being modified to introduce the concepts of robustness and security early on. The concept of a Center for Academic Excellence (CAE) is a new designation for institutions that offer computer-related curricula that conform to a model curriculum for security professionals. Overall awareness of security concerns are rapidly growing. The Common Weakness Enumeration [27] is an evolving listing of the most damaging vulnerabilities is software and steps to eliminate them.

Improving the computer programming languages that are used is a key idea. Historically, we have learned that truly general-purpose languages such as PL-1 and Ada have not worked well. Sometimes, programming languages are imposed upon developers (for example, Objective C and Swift by Apple), even though other languages might be better designed and more usable. Targeted, special-purpose languages (all Turing-computable, but tailored to specific applications) often have distinct advantages, and these advantages are becoming better known. The Tiobe group [26] track programming language usage, and the results are revealing and do not change rapidly.

Software engineering techniques have improved markedly over the years. In the early days, the waterfall model provided a lockstep process of developing requirements, a design, an implementation, testing and maintenance. The spiral model expanded on the waterfall model to create an iterative process that evaluated progress, future direction, and risks to the project at each iteration.

For smaller projects, so-called agile development approaches are becoming the norm. Agile development entails rapid cycles of software release, definition of new capabilities for the next iteration, and implementation. Often work according to an agile development strategy is performed in small teams. Specific approaches under the agile development umbrella include extreme programming, DSDM, FDD and SCRUM.

The Software Engineering Institute [28] at Carnegie-Mellon University developed the Capability Maturity Model Integration – a five-level model to characterize the quality of a software development organization' development process. The model ranges from "chaotic" at level 1, having no set approach to software development at all, to an optimizing, continuous improvement process at level 5. Department of Defense contractors must gain certification at level three of the CMMI to be eligible for most contracts. Such approaches improve software quality in large projects. Through all these initiatives, a lingering question remains: is the complexity of the systems we want to produce out stripping our increase in ability to produce complex systems?

7. SUMMARY AND CONCLUSIONS

Some problems with computer systems are destined to remain intractable because of the difficulties inherent in the development of immensely complex artifacts. Other problems, such as the supply of a properly educated workforce are, while difficult and recurring, addressable. Particularly, gaining more participation by women in computer and information technology can yield significant payoffs. Liberalizing H-1B visas specifically for ICT positions could help. Having more open and interoperable standards for software and software systems improves reliability and efficiency in the development process. Better curricula, programming languages, and improved development processes might at least prevent further loss of ground.

8. REFERENCES

[1] NCES. Percentage of Workers 18 or older using computers on the job. Online, available at **https://nces.ed.gov/programs/digest/d08/tables/dt08_432.asp**

[2] Avery, R. The ENIAC. Online, available at **http://www.ushistory.org/oddities/eniac.htm**

[3] MIT. The Robert Morris Internet Worm. Online available at: **http://groups.csail.mit.edu/mac/classes/6.805/articles/morris-worm.html**

[4] Zhang, B. The airline industry has a massive problem – and there's no real fix. Online, available **http://www.businessinsider.com/deltas-computer-outage-airline-industry-problem-2016-8**

[5] ERP. ERP – Enterprise Resource Planning. Available, online: **http://www.webopedia.com/TERM/E/ERP.html**

[6] Jones, C., and Weise, E. Travel Trouble? Here's why your airline flight is delayed. Online, available: **http://www.usatoday.com/story/money/2016/08/11/airlines-complex-aging-systems-lead-to-flight-delaying-computer-glitches/88539190/**

[7] OS/360. International Business Machines Operating System/360, Online, available at: **https://www.britannica.com/technology/IBM-OS-360**

[8] Dvorak, J. C., Microsoft Office's Spaghetti Code Mess. Online, available at:

http://www.pcmag.com/article2/0,2817,2418117,00.asp

[9] Code.org. How many lines of code?. Available online: **https://code.org/loc**

[10] DevTopics. 20 Famous Software Disasters. Online, available: **http://www.devtopics.com/20-famous-software-disasters/**

[11] Sward, R., and Bolenq, J. Service-oriented architecture (SOA) concepts and implementations. HILT '12: **Proceedings of the 2012 ACM conference on High integrity language technology**. Boston, ACM 978-1-4503-1505-0/12/12. p11.

[12] Stoilov, T, and Stoilova, K. Functional Analysis of Enterprise Resource Planning Systems. Proceedings of CompSysTech '08, **Proceedings of the 9th International Conference on Computer Systems and Technologies.** Pp IIIB.8-1 - IIIB.8-6.

[13] Slagell, A. Thinking Critically about Computer Security. Skeptical Inquirer. 34(3). 2010. Online available: **http://www.csicop.org/si/show/thinking_critically_about_computer_security_trade-offs**

[14] Donaldson, S. Security Tradeoffs, a Culture of Convenience. Security Week. Online, available: **http://www.securityweek.com/security-tradeoffs-culture-convenience. 2013.**

[15] Morris Worm. Wikipedia. Online. Available: **https://en.wikipedia.org/wiki/Morris_worm**

[16] Brooks, F. No Silver Bullet: Essence and Accident in Software Engineering. **Proceedings of the 10th IFIP World Computing Conference.** H.-J. Kugler, ed., Elsevier Science B.V.,Amsterdam, NL. 1986.

[17] Kovacs, E. More Evidence links Russia to DNC Attack. SecurityWeek. Online, available: **http://www.securityweek.com/more-evidence-links-russia-dnc-attack**

[18] Horovitz, O. 2014 Prediction: Smart Cyber Criminals Learn from NSA "Software Implants" Online, available: **https://privatecore.com/blogs/tag/software-implants/**

[19] Oracle. Exception-Handling Antipatterns Blog. Online, available: **https://community.oracle.com/docs/DOC-983543.**

[20] Dishman, L. The Most Critical Skills Gap: Cybersecurity. Online Available: **http://www.fastcompany.com/3062210/the-future-of-work/the-most-critical-skills-gap-cybersecurity**

[21] Middlebury Interactive Languages. Coding vs Foreign Languages: Do We Really Have to Choose? Online, available: **http://www.middleburyinteractive.com/blog/coding-v-foreign-languages-do-we-really-have-choose.**

[22] Hartson, H. R. Human-computer Interaction: Interdisciplinary roots and trends, **The Journal of Systems and Software** 43. 1998. pp 103-118.

[23] Neilson, J. Usability 101: Introduction to Usability. Online. Available: **http://www.nngroup.com/articles/usability-101-introduction-to-usability/**

[24] BMC. Remedy Incident and Problem Management. Online. Available: **http://www.bmc.com/it-solutions/remedy-incident-management.html**

[25] Bureau of labor Statistics. Occupational Outlook handbook. Online, Available: **http://www.bls.gov/ooh/occupation-finder.htm**

[26] Fidelman, M. Here's the Real Reason There are Not More Women in Tehcnology. Online, Available: **http://www.forbes.com/sites/markfidelman/2012/06/05/heres-the-real-reason-there-are-not-more-women-in-technology/2/#479f1ed83c67**

[27] Mitre Corp. Common Weakness Enumeration. Online, available: **https://cwe.mitre.org/**

[28] Patrizio, A. Bad Migration Experiences Ldeave IT Bisses Gun-shy. Online, Available: **http://www.cio.com/article/3120951/it-industry/bad-migration-experiences-leave-it-bosses-gun-shy.html**