

可信计算的产业趋势和研究

SIEWIOREK Daniel P.¹⁾ 杨孝宗²⁾

CHILLAREGE Ram³⁾ KALBARCZYK Zbigniew T.⁴⁾

¹⁾(卡内基梅隆大学计算机科学与电气和计算机工程系 匹兹堡 15213 美国)

²⁾(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

³⁾(Chillarege 公司 纽约 10566 美国)

⁴⁾(依利诺大学阿贝纳溪分校交叉科学实验室 乌尔班纳 61801 美国)

摘 要 可信计算实验研究已经进行了 30 多年,特别是在航空、航天、金融、证券、交通等安全关键领域取得了令人瞩目的成就.为了从数量和质量两方面综述可信计算的发展和进一步推动可信计算的研究,文中分析了可信计算的产业趋势,包括:(1)差错源的变化;(2)复杂性的迅速增加;(3)计算设备总量的增加.针对每一种趋势,指出了那些可以应用于终端产品或实验性产品以及生产这些产品过程的研究技术.文中的研究给出一个框架,既能反映可信计算过去的研究情况,也指明了今后的研究需求.

关键词 可信计算;可信性与安全性的实验研究;计算产业的趋势

中图法分类号 TP301

Industry Trends and Research in Dependable Computing

SIEWIOREK Daniel P.¹⁾ YANG Xiao-Zong²⁾ CHILLAREGE Ram³⁾ KALBARCZYK Zbigniew T.⁴⁾

¹⁾(Department of Computer Science and Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh 15213, USA)

²⁾(Harbin Institute of Technology, Harbin 150001)

³⁾(Chillarege Inc., New York 10566, USA)

⁴⁾(Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 61801, USA)

Abstract Experimental research in dependable computing has evolved over the past 30 Years. To understand the magnitude and nature of this evolution, this paper analyzes industrial trends, namely: (1) Shifting Error Sources, (2) Explosive Complexity, and (3) Global Volume. Under each of these trends, the paper explores research technologies that are applicable either to the finished product or artifact, and the processes that are used to produce products. The study gives a framework to not only reflect on the research of the past, but also project the needs of the future.

Keywords dependable computing; experimental research in dependability and security; computing industry trends

1 Introduction

For over four decades Moore's Law has been a driving force for the computer industry. Doubling on a yearly basis leads to a three orders of magnitude increase in only a decade. Such large increases

in capacity (i. e., number of transistors, processing performance, bits of data storage, and communications bandwidth) require fundamental rethinking of all phases of a product's life cycle from design through usage through maintenance to replacement. In addition to capacity, Moore's Law

also applies to volume. Intel produces more transistors yearly than the number of ants on the planet Earth. Doubling in volume means that every couple of years more computers will be produced than were previously produced in all of history.

The IT industry has grown in many directions. While the Von-Neuman machine is still at the core of the computer concept, the industry that built at best a few thousand machines in 1970, today ships tens of millions annually. Employment changed from a few thousand to a few million. And the breadth of the industry spans technology, manufacturing, software, and IT enabled services, amounting to a world wide figure in the range \$ 2~3T.

This paper identifies three trends in a computer industry fuelled by Moore’s Law that has direct impact on computer system dependability and security: Shifting error sources, increased system complexity, and global volume. Some of the trends were identified decades ago and, hence, there is a rich history among the research threads with respect to these trends. Other trends are just emerging and can be used to predict future directions for research among the three threads.

Section 2 provides background and a framework for motivating the three industrial trends. The next three sections describe each trend in turn and how research in dependable and secure systems responded to each trend. Section 9 provides concluding observations.

2 Trends, artifacts & process

From the days of early computers (that employed vacuum tubes to perform logic and arithmetic operations) to today’s generation of computing systems, dependability has been considered as a fundamental system attribute that determines the system’s ability to provide continuous service to the end user. Evidence of these early efforts can be found in multiple publication from the 60’s, e. g. [1] and [2]. An excellent review describing the advances of IBM computer systems in the RAS(reliability, availability, and serviceability) area from that time may be found in [3].

An important milestone in the evolution of dependable computing (theory and practice) was the establishment of a technical conference on fault-tolerant computing: The First International Symposium on Fault Tolerant Computing was held in 1971. This forum has established itself as a primary arena for presentation, discussion, and dissemination of new ideas and concepts in development of dependable systems.

Over the years fault/error models have evolved along with the advances in the system hardware/software. Table 1 summarizes the changes over the last four decades in terms of the technology, error/fault sources, number of users and their level of sophistication/training.

Table 1 Fault/error sources, level of integration, users, and user sophistication over the last four decades

Year/decade	1970	1980	1990	2000
Typical Systems	Mainframes	Workstations	Personal Computers	Mobile devices, e. g. , cellphones, PDAs
Fault/error Sources	Hardware	Hardware, network.	Hardware, network, software, human errors	Hardware, software, wireline/wireless networks, environment e. g. , frequent connectivity loss
Integration/complexity	Close systems; Highly custom designs, where both hardware and OS are fully controlled by the vendor	Mostly close systems; Network connectivity; Standard interfaces exposed to users	Open systems; Wide access to network; COTS operating systems; Third-party hardware and software	Open systems; Proprietary and COTS operating systems; Highly integrated PC-like systems
People/Users	Tens of thousands	Millions	10 of millions	100 of millions
Level of sophistication/ training	BS in engineering; 5000h	Basic knowledge in computing; 500h	Basic computing literacy 50~100h	Training at the time of a purchase of a device. Hours

The technology has evolved through dramatic changes starting from mainframes in 1970s (where highly skilled personal was required to operate the systems) through an era of workstations in 1980s and personal computers in 1990s, to the current generation of mobile/handheld devices (e. g. , cell-phones, PDAs), where the technology reaches the

general public. Today devices must often operate in highly variable and harsh environments. As a result the technology must: (1) hide complexity so that relatively unsophisticated customer can operate the device and (2) provide continuous operations despite errors/failures.

Initial focus (in 1970s) was mainly on hard-

ware errors as the hardware devices were the major cause of problems. The following decade (1980s), with introduction of workstations and their network connectivity, makes the network an important additional source of errors. The wide use of personal computers (in 1990s), executing commodity software, makes the software become a primary source of failures. The current decade can be characterized with the dominance of failures due to the environment and operators.

Our framework is defined by three attributes: Trends, artifacts and process. The trends refer to industry trends that have been taking place, and which have a direct impact on dependability. Each trend is distinct and had been consistently present for a substantial period of time — Often a couple decades. We identify three trends and discuss them in considerable detail for the purposes of this paper.

Separate from the trends are artifacts and

processes — The other two dimensions of our framework. The artifact is the product of the industry — Be it, a piece of hardware, software or service. An example being a computer, a piece of shrink wrap software, or a cell phone contract. It is the entity of commerce and defines the work product of engineering effort.

The process is the means to produce an artifact. It is the engineering methods, tools, or labor which the industry employs to create a viable method of manufacturing or development. As we reflect on artifact and process, we recognize that much of engineering and research is often directed to one of the two, or both.

Table 2 links together the three dimensions of our framework with trends as rows, and artifacts and process as the two columns. At the intersection of each row and column is the subject of our study.

Table 2 Industry trends, artifacts and process

Industry Trend	Artifact	Process
Trend 1: Shifting Sources Failure rates drop in hardware and change to other sources	Monitoring Failure data analysis Fault Injection	Raise the level of abstraction
Trend 2: Explosive Complexity Growth in the system complexity, users, and shrinking user tolerance to failures.	Anomaly Detection Trend Analysis & Formal Methods	Model checking ODC Software Reliability Testing
Trend 3: Global Volume High level of integration and emerging open systems, a source of new dimensions in failures	Tools to assess resilience to both malicious and non-malicious errors	

Trend 1: Shifting Sources, clearly began in the 1980s, but was noticeable towards the end of the 1980s and by mid 1990s had caused a major change in the dependability area. Trend 2: Explosive Complexity, began toward the early 1990s when the cost of computing was dropping substantially and distributed computing was on a growth path. By mid 1990s the internet boom contributed to an even larger growth. Trend 3: Global Volume, is only at its inception, and can be argued to have begun with the huge increase in small yet powerful devices flooding the market. The cell phone, the PDA and the availability of wireless digital networks will have their impact.

3 Trend 1 — Shifting sources

One of the dominant trends has been the changes in failure rate, as well as the sources of failures that dominate in a particular timeframe. By and large, we can conclude that the hardware failure rates are down while the relative contribution of software is up. In addition as technology matures, the user set changes, and the degree of sophistica-

tion in the products increase, new sources of failures become prominent.

Transient faults have traditionally been associated with the corruption of stored data values. This phenomenon has been reported as early as 1954 in adverse operating conditions such as near nuclear bomb test sites and later in space applications^[4-5]. Continuously decreasing feature sizes and supply voltage of devices reduce capacitive node charge and noise margin, even flip-flop circuits inevitably become susceptible to soft-errors^[6]. Constantly pushing the processor performance envelope will shortly place us in an unfamiliar realm where logically correct implementations alone cannot ensure correct program execution with sufficient confidence. As a result vendors of high-availability platforms have long incorporated explicit error detection and correction techniques in their architectures. The basic techniques involve information redundancy (e. g. parity and ECC), space redundancy (achieved by carrying out the same computation on multiple, independent hardware at the same time and corroborating the redundant results

to expose errors), and time redundancy (where redundant computation is obtained by repeating the same operations multiple times on the same hardware).

3.1 ECL to CMOS circuit technology

The significant change in technology, over the past decades, is not only the increase in speed and reduced power consumption, but also the reliability of the devices. Fig. 1, reproduced from IBM data^[7] shows that system outage as caused by hardware failure has dropped by two orders of magnitude in two decades.

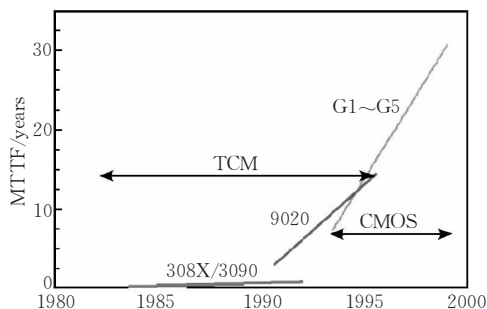


Fig. 1 Failure rate changes in hardware

This dramatic change in reliability has been the driver of a major portion of Trend 1. The shift in circuit technology from ECL (the technology for the 308X/3090 series of mainframes) to CMOS is dramatic in terms of reliability, as shown in the figure. TCM in the figure stands for Thermal Conduction Module, a ceramic multi-chip package that is liquid cooled.

Considering the beginnings of the FTCS conference, which dates to 1971, one can see why the focus in the early years was on hardware fault tolerance. Product dependability was defined by how well one could keep a box running in spite of hardware malfunction—Be it permanent or temporary. To combat the transient failures, up to 25% of the circuitry was utilized for error detection and correction. These architectures allowed for very high data integrity with no data path inside the CPU left unchecked. Instructions retry mechanism further increased fault tolerance^[8].

It was also becoming clear that the next generation of circuit technology, CMOS would obsolete ECL. As Fig. 1 illustrates, the MTTF of a high end machine is over 30 years, almost two orders of magnitude better than that class of machines two decades ago.

The same time period has witnessed the growth of the microprocessor and the PC industry. Today Intel in the P6 family processors brings

high-end features to the mass market. All P6's internal registers are parity-checked, and the 64-bit path between the CPU core and Level-2 cache uses ECC. Built-in diagnostic features allow monitoring and reporting on more than 100 events and variables inside the chip, including cache misses, register contents, and occurrences of self-modifying code. The P6 also improves support for check-pointing (i.e., rolling back the machine to a known state in the event of an error), however, the operating system has to be written to take advantage of machine-check interrupts.

The drop in the failure rates of current COTS (commercial off the shelf) processors to 100 FITs^[9] (MTTF of over 1110 years), could suggest that device failures are no longer a major dependability problem. However, as indicated in [10], there is an increase in significance of other threats, including: (1) susceptibility to environmental interference due to reduced device sizes and power levels of logic (as discussed at the beginning of this section); (2) hardware design faults that are discovered after the design is completed (For example, processors of the Intel P6 family in April 1999 had from 45 to 101 reported design faults, of which about 60% remain uncorrected), and (3) uncertainty about wearout, which may lead to an increase in the failure rate over time. This shows that the benefits of a very low device failure rates will only be significant if the likelihood of system failures due to transient faults and design faults can be reduced as well.

3.2 Software failures

One of the consequences of the dropping hardware failure rate is that the other failure modes have become more prominent. Software, which has also been growing in complexity, has gradually contributed to a larger proportion of system outages. Through the 1980s, while the fault tolerance methods were being developed for hardware and the incidence rate of hardware failures was dropping software failures became more prominent. At the same time, the focus on software reliability methods was marginal. In the high-end server business, most of the development budgets were focused on new function, since that was a growth segment all the way into the 1990s. The PC segment was at its inception and the focus was functionality. As a consequence, software failures—The class of problems that had damaging effects as significant as a hardware outage that took the entire system down became evident.

The high end server industry responded rapid-

ly. Both fault avoidance and fault tolerance techniques were applied. Just like the hardware platforms for the high end servers, the software operating systems included more and more recovery code. The result is impressive. Two decades later, a high end IBM server has almost no cold starts in an entire year.

3.3 Planned outage

Faults and failures produce the mental image of uncertainty, and catastrophic consequence. While they do happen, they are by far less common in high end servers. However, high end computing has a disturbing problem called planned outage when, on purpose, systems need to be shutdown. Planned outage used to be common with installation and maintenance of hardware, and then became common with software updates and maintenance. Databases needed to be reorganized or networks needed to be reconfigured. While, the surprise element was not present, the un-availability and disruption of services caused just as much a problem. With businesses running globally, 24×7 availability was vital and planned outage accounted for more downtime in the 1990s than un-planned outage.

3.4 Desktop software expectations

The desktop industry in the mid 1980's had been a novelty and enjoyed user tolerance to fail-

ures. As time progressed and the dependency on desktop software grew it acquired the burden of any successful segment that enters the stable stage of its lifecycle^[11]. In this stage, the importance of novelty fades and dependability rises. Although we have witnessed significant growth in the dependability of products, there is yet much more to be addressed. The issue is complicated by newer sources of failures that arise such as viruses and security holes. Global usage and lower training levels among the user set push the demands on dependability further.

4 Dependable system research versus industry Trend 1

Research on dependability has advanced following the changes in hardware and software technologies.

4.1 Fault/error classes

Understanding fault/error models is of primary importance in providing sound methods and techniques for dependability assessment and in developing efficient detection and recovery mechanisms and algorithms. Table 3 gives a generic classification of faults based on their temporal persistence and origin. The fault/error classification in Table 3 is a simplified version of the more comprehensive taxonomy provided in [12].

Table 3 Fault classes

Fault classes	
Based on the temporal persistence ^[13]	Based on the origin ^[14]
Permanent faults, whose presence is continuous and stable. Intermittent faults, whose presence is only occasional due to unstable hardware or varying hardware and software states, e. g. , as a function of load or activity. Transient faults, resulting from temporary environmental conditions.	Physical faults, stemming from physical phenomena internal to the system, such as threshold change, shorts, opens, etc. , or from external changes, such as environmental, electromagnetic, vibration, etc. Human-made faults, which may be either design faults, introduced during system design, modification, or establishment of operating procedures, or interaction faults, which are violation of operating or maintenance procedures.

4.2 Monitoring and emulation of fault sources

Table 4 summarizes the trends in experimental dependability research across four decades organized into methods and focus of monitoring operational systems and artificial evaluation by fault injection. The individual columns highlight emerging: (1) sources of data being collected from systems in the field and (2) fault/error types being integrated into fault injection tools and environ-

ments. Analysis of failure data from operational systems provides insight into the dominant error categories in deployed systems. It also gives valuable feedback for driving fault/error injection experiments. Fault/error injection allows accelerate failure occurrence in the system and, hence, provide very rapid validation of prototype design and further guidance to design decisions.

Table 4 Examples of experimental dependability research

	1970's	1980's	1990's	2000's
Operational life monitoring	Crash dumps	Error logs	Natural workloads	Human-computer interaction errors
Fault injection	Stuck-at	Memory	API	Security

4.2.1 Operational life monitoring and failure data analysis

Understanding the characteristics of a fault source evolves through several stages. In order to gain an understanding of the significance of a fault source, initial measurements focus on summarizing the underlying statistical distribution with averages such as mean time to an event. Since little is known about the fault source, existing measurement frameworks are used to make estimates. This monitoring may be primary (such as analysis of system event logs) or secondary (such as reports from the field). In order to discover more about the statistical properties of the source (such as distribution type and distribution parameter values), customized error monitoring systems that are sensitive to the fault source, while at the same time filtering out extraneous information on other sources, have to be developed. In the next stage, a deeper semantic understanding of the fault source and how it propagates are used to devise real time anomaly detection so that the onset of a new fault can be discovered and isolated quickly. This pattern of the evolution of stages applies to each fault source and the depth of understanding of a fault source is directly related to how many stages have been explored.

Direct monitoring, recording, and analysis of naturally occurring errors and failures in the system can provide valuable information on actual error/failure behavior, identify system bottlenecks, quantify dependability measures, and verify assumptions made in analytical models.

There are three basic elements in an online trend diagnosis system^[15]:

Gathering data/sensors. Sensors must be provided to detect, store, and forward performance and error information (e.g., event-log data) to a diagnostic server whose task it is to interpret the information.

Interpreting data/analyzers. Once the system performance and error data have been accumulated, they must be interpreted or analyzed. This interpretation is done under the auspices of expert problem-solving modules embedded in the diagnostic server. The diagnostic server provides profiles of normal system behavior as well as hypotheses about behavior exceptions.

Confirming interpretation/effectors. After the diagnostic server interprets the system performance and error information, a hypothesis must be confirmed (or denied) before issuing warning mes-

sages to users or operators. For this purpose, there must be effectors for stimulating the hypothesized condition in the system. Effectors can take the form of diagnostics or exercisers that are down-line loaded to the suspected portion of the system, and then run under special conditions to confirm the fault hypothesis or to narrow its range.

Challenged by the increasing number and severity of malicious attacks, security has become an issue of primary importance in designing dependable systems. Analysis of data on security related system activities permits the identification of security attacks, and vulnerabilities exploited by the attacker, and enables classification of the attacks. Many classifications of attacks have been tendered, often in taxonomic form. A common basis of these taxonomies is that they have been framed from the perspective of an attacker—They organize attacks with respect to the attacker's goals, such as privilege elevation from user to root (from the well known Lincoln taxonomy). Taxonomies based on goals are attack-centric; those based on defender goals are defense-centric. Defenders need a way of determining whether or not their detectors will detect a given attack. It is suggested that a defense-centric taxonomy would suit this role more effectively than an attack-centric taxonomy. Research has led to a defense-centric attack taxonomy based on the way that attacks manifest as anomalies in monitored sensor data. Unique manifestations, drawn from 25 attacks, were used to organize the taxonomy, which was validated through exposure to an intrusion-detection system, confirming attack detectability. The taxonomy's predictive utility was compared against that of a well-known extant attack-centric taxonomy. The defense-centric taxonomy was shown to be a more effective predictor of a detector's ability to detect specific attacks, hence informing a defender that a given detector is competent against an entire class of attacks.

4.2.2 Network user interaction failures

Resolving network interoperability problems is difficult and time consuming. Almost every user has experienced such a problem either directly, or as a by-product of a task they were attempting to complete. Problems may originate or be complicated by system heterogeneity, administrative policies, security practices, and end user errors or improper mental models.

Advances in network flexibility, self-repair, and reconfiguration will improve underlying per-

formance (Meseguer et al, 2003) but lead to increased complexity. Furthermore, it is likely that the user will be unable to rely on a consistent detailed mental model of network state and topology. As such, new human-computer interaction methods and tools will be required to enhance user awareness and problem resolution.

As an example, consider remote network access that generates frequent problems and has considerable detrimental impact on user efficiency. Data on remote access trouble tickets covering 2.5 years from June 2000 through December 2003. The analysis of incident reports is a common and accepted data collection methodology (Wickens, 1995; Salvendy & Carayon, 1997).

During the period in question there was a phase out of DSL service, an introduction of a VPN option, and a beta test of a licensed dial-up ISP for traveling users. The latter two may explain the slight rise for incoming cases near the end of the sampled period, while the DSL phase out may explain the small hitch in late 2000. However, it was interesting to note that, in general, the rate of incoming cases was remarkably linear (Fig.2). This suggests that little progress was made over this period in developing methods of reducing case-load.

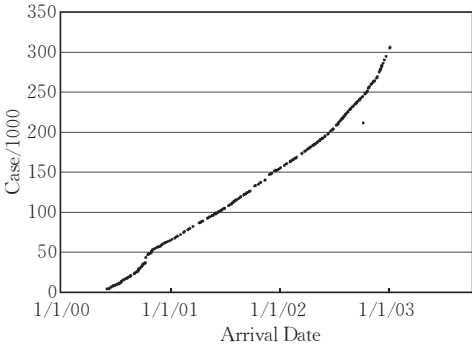


Fig. 2 Case arrival rate

It is important to note that Hours to Resolve is not a good measure of staff time consumed. This is simply the time from the initial user query to the time the case was closed. The most salient observation upon looking at Problem Type was the high caseload and time sink resulting from phone number queries (Table 5). Also apparent was the high mean time to resolve problems stemming from third party networks. Problems due to single configuration change events were frequent (22%) and likely the result of shifting policies and network options (i.e., DSL phase-out, VPN roll out, and licensed dial-up ISP beta test).

Table 5 Problem type statistics

	N	Hours to Resolve		
		Mean	Std Dev	Sum
Core	69	58	111	4013
Network	45	77	132	3447
Leaf	66	60	133	3957
Single	86	52	104	4490
Phone Number	132	27	86	3523
Overall	398	49	110	19431

Over 25% of the cases and 3500h were requests related to phone number requests. Some of these were influenced by time zone effects (e.g., user is in Asia and staff is only fielding queries during the work day).

A large number of cases were resolved within the first day (Fig. 3). Inspection of the pace of problem resolution shows that Leaf cases linger on much longer than other types.

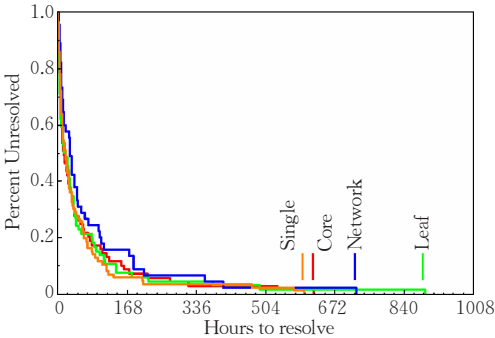


Fig. 3 Resolution by type

Comparison of case sample totals for the Win, Mac, and Li/unix categories is shown in Table 6.

Table 6 Operating system statistics

	N	Hours to Resolve		
		Mean	Std Dev	Sum
Win	92	61	118	5592
Mac	22	60	68	1322
Li/unix (no Mac)	23	61	99	1392
Unknown	121	54	108	6493
Mixed	8	139	304	1109
Overall	266	60	118	15908

The curves in Fig. 3 exhibit the typical decay pattern expected in help desk operations.

Perhaps the most interesting observation from Fig. 4 is the almost linear dive to full resolution by Mac users shortly after the first week. This manifested as considerably reduced variability for the Mac category when compared to the other categories.

The high rate of problems associated with the VPN implies many cases were specifically due to problems originating from the use or application of

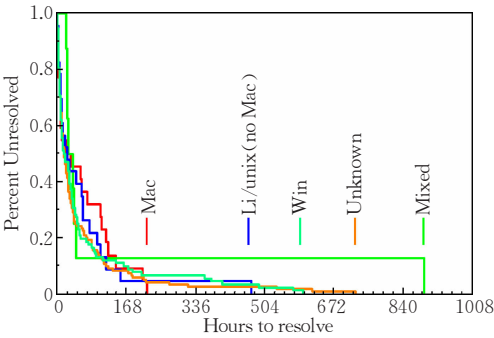


Fig. 4 Resolution by operating system

security policies. Two security categories were examined: VPN and Realm. VPN cases included problems installing, configuring, and using the VPN. Cases associated with Realm included conflicts with security policies (e.g., mail relaying), authentication (e.g., password errors), and network card registration. Security problems were common — 41% of the cases and 47% of the time involved VPN and/or Realm (Table 7). VPN cases seemed to be particularly time consuming but this may be due to users waiting for new VPN client distributions. Windows deployment was more rapid than Mac and *nix.

Table 7 Security statistics				
	N	Hours to Resolve		
		Mean	Std Dev	Sum
None	158	53	112	8356
Realm	54	53	108	2855
VPN	42	104	156	4375
VPN, Realm	12	27	29	322
Overall	266	60	118	15908

This effort was valuable in that certain classes of problems and nuances surfaced rapidly. A few key points were identified during this effort:

- (1) The mean time to resolve all help related cases was about 2 days(49h/case). This jumped to 60 h/case for the subset not including phone number requests.
- (2) Over 25% of the cases were due to phone number requests.
- (3) Problems with configuration changes resulted in 22% of the cases.
- (4) Time to resolve cases with Mac OS(9 and X) was considerably less variable than other platforms. The longest Mac case was resolved after just over 10 days.
- (5) Security problems were frequent — Especially for end users where over half of the problems were related to obtaining necessary user rights.

4. 2. 3 Fault/error injection

Fault injection has been used for more than three decades and it is generally recognized as an important method for dependability analysis. One reason for the success of fault injection is that it enables evaluation of the actual implementation and attempts to capture how a programming error or manufacturing defect may affect the system’s dependability. Fault/error injection can be employed to conduct detailed studies of the complex interactions between fault/error and fault/error handling mechanisms. This approach complements analytic evaluation and simulation, which apply to earlier design stages.

One important step in the evolution of fault injection was formalization of the key components of a fault injection experiment: The faults set, the set of activations, the set of readouts, and the derived measurements^[10,16]. These concepts were further extended by formalizing failure acceleration (i.e., injecting faults at a high rate to increase the number of failures observed) and applying it to an IBM 370 mainframe^[17]. In the late 1980’s an idea of software implemented fault injection (SWIFI) was proposed. Multiple tools were developed to enable rapid assessment of system dependability without the need of building, often expensive, hardware fault injection tools. Another important method of injecting faults was the use of radiation sources to induce single event upsets (SEUs) on exposed ICs^[18-19].

While fault/error injection methods and techniques have been extensively studied in academia, industry also employs fault injection. Work in [20] reports on fault injection based testing of recovery and serviceability in the IBM ES/9000 systems. Fault injection and software testing were used by Ansaldo-Cris, Italy to assess dependability of new generation of Railway Control Systems^[21]. In [22] physical fault injection at the pin-level was employed to validate error-handling mechanisms of teraflops supercomputer developed by Intel. Software implemented fault injection assists in evaluation of embedded flight control system^[23].

4. 2. 3. 1 Fault injection at the operating system API level

The success of many products depends on the robustness of not only the product software, but also operating systems and third party component libraries. But, until now, there has been no way to quantitatively measure robustness. Ballista changes this by providing a simple, repeatable way to

directly measure software robustness without requiring source code or behavioral specifications. Ballista is a "black box" software testing tool, and it was demonstrated on testing the APIs of Commercial Off-The-Shelf (COTS) software. [24] provides a comprehensive assessment of 15 POSIX-compliant operating systems and libraries as well as the Microsoft Win32 API.

Each of 15 different operating system's robustness has been measured by automatically testing up to 233 POSIX functions and system calls with exceptional parameter values. Overall, only 55% to 76% of tests performed were handled robustly, depending on the operating system being tested. Hardening can be accomplished by first probing a software module for responses to exceptional inputs that cause "crashes" or "hangs". When these robustness bugs have been identified, a software wrapper can be automatically created to filter out dangerous inputs, thus hardening the software module. More details and ideas on use of protective wrappers may be found in [25-27].

4.2.3.2 Fault injection at the application level

To analyze dependability at the application level, the faults are injected into the application program itself or the layer between the application and the operating system. The methods of injection can be classified into two types according to when the faults are injected^[28].

(1) Compile-time injection

Faults are injected into source codes or assembly codes of target program, and the fault injection generates an erroneous software-image at compile-time. This method alters instructions of target program, but it can cause no perturbation during runtime.

(2) Runtime injection

Some mechanisms can be required to trigger injection during runtime. There are many common mechanisms of trigger, including: Interrupting the application execution by trap, timer, exception, or executing the application in trace mode; and executing specific fault injection code by adding instructions to application code.

Software implemented fault injection without special hardware is low complexity, low development effort, and low cost^[29], but it also has higher perturbation and lower monitoring time-resolution. In HFI-4^[30], the Single-Cycle fault injection for bus signals is presented based on technique of socket insertion. User can set one address or category of addresses by masking code as conditions of

injection, at the same time, the duration of injection is restricted into one bus cycle (read or write), and the behavior of injection can not affect subsequent cycles. Thus, it can inject faults into registers or memory accurately, and simulate software-implemented injection, at same time, it can not incur any perturbation to target program and supply time-resolution of hardware monitor.

5 Trend 2 — Explosive complexity

The complexity arises from a few different factors, and with it issues in dependability get more diverse. The lines between pre-ship and post-ship are blurred increasingly with our ability to ship fixes instantly on the internet. Customizability and inter-operability of services further makes products, vendors, and services less distinguishable. Wireless technology makes devices and users location transparent, and the demands 24×7 availability spread across the industry.

5.1 Growth and complexity

This huge growth is not just due to selling more computers and increasing the installed base of software. Applications, infrastructure and services have exhibited compounded growth. Thus, while individual segments of the industry show a steep linear growth, the complexity of the systems built for them grows nonlinearly.

While it is always hard to measure individual engineer productivity, which we believe has grown due to better software engineering tools, the labor market growth figures illustrate substantial growth. Since development continues on larger components, which are themselves growing in features and functionality, the ability to synthesis larger and more complex systems has grown substantially. For instance, a modern web application is built on a number of standard components—Operating system, middleware, web server, transaction manager, database system, the rendering layers supporting streaming media — All built upon a transport layer or net-work that is configured separately. The application and business logic is an industry-specific product layered upon these other components. Integrating the various components and dealing with the complexity of such systems is a new layer of conceptualization with its own dependability issues.

5.2 No test case

In every industry that has faced rapid growth, standardization, layering and multiple sourcing have been used to deal with the growth. The com-

puter industry is no exception. , but with the added ability of extensive customization as a competitive edge. Customization is made possible by the ease of programmability at a late stage in development. It is also an excellent marketing vehicle to differentiate one standard component from another in terms of added features.

Many a system built with standard components is realized only at a customer site. While the piece parts are supposed to work, the fact that they do can only be tested on site. This creates the challenge of developing appropriate test suites. The notion of functional testing and checking interfaces breaks down beyond a point, needing scenario tests that evaluate the end to end function. However, minor variations in individual interfaces make it almost impossible to access and verify reliability, thus, the problem of, "no test case", the ever increasing cost of integration and delays in delivery are often due to this customization.

5.3 Re-defining failure

For decades we have assumed that a failure is a well understood concept. The IEEE/IFIP definition of failure is, "a system not perform up to its specification"^[31]. This is an apt definition for a piece of hardware, or an IC chip, a microprocessor, or a UNIX command that has been unchanging for 20 years. In each of these cases one can find a specification. But, increasingly piece parts do not have specification. And if the specifications do exist, they are vague, incomplete, and never really meant to serve the purpose of providing a clear binary answer to whether a situation is a failure or is working as designed. When there is a specification for a new product or service, it may only be in the form of commercial material and not a detailed technical specification. And if one does find the technical specification, it could be obsolete.

How does one recognize a failure? A good place to observe is the customer service desk. When customers call, the problem diagnosis process usually tries to answer the question: Is it working as designed, or is something wrong? If it is not working as designed, is it because the user did something incorrectly, or is the product or service broken? In many cases the answers are reasonably clear. However, there is now an increasing trend when the blame cannot be assigned. And regardless of the blame being assigned, the desire to retain customers, and develop a better product motivates fixing the situation so that it does not occur.

From this perspective, the definition of a failure needs to be rethought. It is the expectation in the eyes of the customer, the satisfaction of the customer and to a much lesser degree the technical specification of the product or service^[32].

5.4 Speed of service

The focus on customer service is driven by choice, cost and image. Customer service is very expensive but poor customer service is even more expensive. There is a constant tradeoff between managing the cost of customer service and maintaining a positive image that are reflected in customer satisfaction numbers measured from the field.

Customer service spans a broad range of topics, problem determination and solution delivery being only two of them. If there is one metric that captures the concerns of dependability it is "speed of service". This has probably the single highest impact on the image of the company, satisfaction to the customer and cost.

The faster one needs to respond to a problem, the higher the cost of service. The more time one has to manage a situation, the more flexibility there is in the organization to manage the cost of service. This challenge is at the core of building a services strategy and technology to support the products. Remote diagnosis, failure prediction, self repair, are different ways by which service can be improved.

As products have become complex, the piece parts are imported from different sources and the install base is more global, the complexity of service has grown. Outsourcing the service is an attractive option used by many. However, with the outsourcing of service overseas to countries with lower cost structure, the infrastructure to deliver the service has also risen in complexity.

5.5 Constant development

In the past there was a time when a product was built, assigned a product number and shipped. Other than for service the product and its functionality was encapsulated in that part number, and never changed.

This was the day before microcode and downloadable software patches. Our ability to change a product without bending metal has created a new concept of shipping a product. It is conceivable (and one could argue we are already there), that we can buy a widget and then determine what it does.

Assuming that the process of delivering the product to an end user takes zero time, the chal-

lenge becomes the process of developing the product and verifying that it works. The unintended consequence, is that the development process never really has a clear start and end, which historically was forced by the product delivery mechanism. Now, one can argue that the product delivery mechanism is not what should define produce start and stop, but product and investment management.

The consequences are fascinating, especially to dependability. The constant development and delivery places much lower premium on dependability. Since the change cycle is assumed to be fast, it tends to make design far more reactive. It allows a larger field test of situations that otherwise would need to be tested in house. The processes of development, service, and dependability design get rolled into one big cycle, that has few distinctions between them. From a product manageability perspective it can give rise to a far greater number of generations of product that co-exist in the field. This latter point makes it much harder to deliver service, and maintain the install base.

6 Dependable system research versus industry Trend 2

The issues in Trend 2 that drive greater complexity, brings home a central theme in the research area of dependability-breadth. What was once an area that was better defined by failures and faults from well known sources is now dispersed across a broader set of sources and relationships. Thus, the concepts based on clear fault models with a known specifications and design is muddled by less clear notions of failure and blurred lines among design, development, and field life of product.

6.1 Software reliability

Measurement of software reliability from field data has been limited, compared to its counterpart in hardware. The few studies that report measurement from field data have been from large vendors, and the studies conducted often, by academically inclined individuals or partnerships with university^[33-36]. Yet these studies are insightful in giving us an order of magnitude of the MTBF which illustrates that the range for software is between 10—100 days. The studies also illustrate the reliability growth that occurs as the software ages in the field. One of the primary difficulties in these studies is the collection and filtering of field data, and determining an appropriate compensation for under

reporting. The first IBM study and the following Microsoft study, both use educated guesses that seem reasonable. Studies to estimate reporting rates, and methods for good data filtering are open areas of work.

6.2 Use of formal methods

An important area where formal methods were very successful is model checking, applied in the design and verification of finite state systems, e. g. , distributed algorithms and protocols. This is achieved by verifying whether the model, derived from a hardware or software design, satisfies a logical specification (often expressed as temporal logic formulas). Early work on [19, 37] provide foundation for current generation of model checking tools such as SPIN^[38].

6.3 Software testing

There is a vast difference in the level of technology for testing between hardware and software. Hardware testing has over the years developed into a fairly sophisticated set of methods and tools. Software testing, too, has a long history; however, the level of complexity remains largely undressed.

Testing at a program level is fairly well understood and the practice well documented^[39] with a number of tools and measurements. Functional testing which amounts to a slightly higher level of abstraction too has a reasonable body of work, but is far from complete. Black box testing methods that an application programming interfaces are mainstream methods. However the degree of automation is limited by the quality of specification. A number of tools have made considerable progress in developing parsimonious test suites to address the combinatorial explosion that occurs in test generation. The use of orthogonal arrays^[40] that first addressed the issue was advanced by the AETG tool set^[41] with sophisticated heuristics to work with constraints. Open problems in this area have to do with the development of functional test on a richer variety of inputs and constraints. There is also little work that relates the nature of faults to the efficacy of specific methods of test generation.

Model based testing approaches the issue of functional testing from a much different perspective. These are borderline formal methods, since they do have a formal representation of function, but not necessarily derived from exact specifications of the program. State charts^[42] blend graphical representation with functional properties, and are particularly attractive in applications where the

model lends itself to the application such as protocols.

Complexity that arises from integrating several layers and inter-dependent services poses a hard problem for testing. The problems are deepened due to incompleteness of specification and environmental constraints that are inadequately understood. Developing conceptual models for these systems, and developing tests and establishing meaningful measures of coverage are open problems.

6.4 Application-aware security and reliability

The increasing complexity of computer systems and their deployment in mission- and life-critical applications are driving the need to provide applications with security and reliability support. In addition, the Internet's ubiquity makes systems much more vulnerable to malicious attacks that can have far-reaching implications on our daily lives. The catastrophic failure of the AT&T telecommunication network in New York affected financial and life-saving systems such as airline reservations and the American Red Cross blood supply tracking systems^[43]. The Morris worm^[44] that exploited buffer overflow vulnerability in fingerd, and the Code Red worm^[45] that exploited a buffer overflow in Internet Information Service (IIS), are all indications of a rising problem of computer and system security. In this all-pervasive computing environment the need for security and reliability has expanded from being limited to a chosen few expensive systems to a basic computing necessity.

The traditional one-size-fits-all approach to security and reliability is no longer sufficient or acceptable from the user perspective. A potentially much more cost-effective and accurate approach is to customize the mechanisms for detecting security attacks and accidental errors using knowledge about the expected or allowed program behavior. In addressing this challenge, a concept of application-aware checking as an alternative has been proposed^[46]. By extracting application dependability characteristics using compiler analysis and enforcing the characteristics at runtime using the hardware modules embedded in a configurable hardware (e. g. , [47]), it is possible to achieve security and reliability with low overheads and low false-positive rates.

The idea of customizing hardware checkers based on application needs is compatible with recent industry trends such as utility computing from HP and SUN, in which utility computing grids can optimize themselves to suit the application's per-

formance needs. Analogously, an application aware checking methodology lets tune the hardware checkers taking into account application specifics, e. g. , focusing the protection mechanisms on the most critical application data and/or system resources.

7 Trend 3—Global volume

The numbers of end users has grown by orders of magnitude in the IT industry. And with it comes a plethora of changes in every aspect of the business. The user's tolerance for failures is lower, and the higher levels of integration are a source of new dimensions in failures. These trends often alter assumptions we once held that may no longer be true. In other respects ideas that were considered impossible only a decade ago, are already in production.

7.1 Form factor and mobility

The smaller geometry of circuits and lower power has reached the point where devices such as a hand-held PDAs rival the compute power of a desk top computer of only a few years ago. While that trend has been predicted in Moore's law for more than two decades, it is only now that the hand carried, or wearable device has the power for significant computing. Coupled with the corresponding decrease in cost, the volume of devices grows by a couple orders of magnitude over desktop and lap top computers. No longer are computers numbered in hundreds of thousands to a million per year, but in tens to hundreds of millions computers with each model having a useful operating life of a couple years. The cell phone is an excellent example of a form factor, technology, and accepted functionality that will drive more CPU sales with the power of a desktop computer than previously conceived.

This large volume of devices is not mere power in the hands of the end users. There is an infrastructure that needs to be laid down to enable the end-user device, manage the end user device, and make it a channel for new features and services. All this with location transparency and in the ideal situation without forcing direct contact with the end user.

7.2 Lower training threshold

The increase in volume of users leads to lower levels of training of the average end user. While this is quite evident in the consumer segment, such as a cell phone user, it is also true in other segments. Clearly, the need for specialized skill and

the expectation that there will be the necessary training is always true in the high end servers network markets. However, the desktop era demonstrated the need to build computing elements that required lower and lower levels of training for the end user thereby enabling mass markets. Along with this, there has also been the trend to place systems management and repair in the hands of technicians not requiring a degree in computer science and who can be trained rapidly. The growth of education programs in information technology, that focus on solutions and capabilities and less on engineering design reflects this trend.

Dependability demands accelerate with lower training. What may have been once considered an acceptable level of difficulty in use and maintenance of systems, is no longer acceptable. Thus, what are minor irritations in historic systems, become faults today. What would historically be routine maintenance becomes serious failures. While this has some elements of the changing definition of failure it is different in that the capability of the user is much reduced. Thus, the design assumptions on products themselves have dramatically changed to accommodate this trend.

7.3 Dependability in systems management

With more diversity of devices in the market, the task of installing, delivering, maintaining, servicing and upgrading are all now far more challenging. Some tasks that could be done manually, cannot be done manually. Failures in these systems have their effect multiplied by the number of users they serve. For instance, a wrong installation of a version of an application for a thousand sales agents, can be updating databases around the world with incorrect pricing information causing a huge disruption in accounting. This may have been triggered by a human error, due to lack of training, or a programming error in the version management, or a secondary problem due to the integrity of a database.

7.4 Market maturity drives product sophistication

The information technology business directly effects every one. Awareness of computing products occurs early in childhood. A direct consequence of this is the maturity of the average user is on the rise (and this co-exists with lower average training). The consequence is that the expectation of a product in the information technology space is far more advanced. While novelty is always attractive to people, one grows out of the novelty phase and becomes more demanding of capability, per-

formance and durability. Compare for instance, the current attitude towards cars. We not only expect them to work and drive well, but do so without demanding a tune-up for up to a 100000 miles.

8 Dependable system research versus industry Trend 3

Commencing with the workstation boom in the early 1980s, many of the traditional system management procedures fell to the user. Since workstation behavior was not visible to a central site, the research community developed techniques in trend analysis to monitor activity and report deviation from normal behavior. Thus, rather than attempt to reconstruct sequences of events after the fact, operational personnel would merely be notified of unusual events. Operational personnel could focus their attention on events that were likely to be meaningful. In addition, the trend analyzers provided data surrounding the event so no reconstruction would be required.

8.1 Adaptive model of normal behavior

Trend analysis creates a model of normal behavior and looks for deviation between current behavior and normal. Since systems and their applications continually evolve, the model must also learn the new behavior and factor it into a new model for normal. In its' early stages of development, Harbinger? [34] saw traffic grow by an order of magnitude in a matter of months on an Ethernet backbone. Harbinger would flag events that were plus or minus one or two standard deviations away from prior behavior. Operational personnel could examine these behaviors and, if considered normal, do nothing. The Harbinger model would adapt such that any change that persisted for over two weeks would now become part of the new normal behavior. Harbinger reduced the number of events of interest by several orders of magnitude^[48]. The case study on using models of normal behavior for voice mail systems in Telco shows that alarm analysis could predict failures up to a few weeks in advance^[49]. Setting up the alarm measurement and analysis system decreased the mean time to repair at least by a factor of two, with a corresponding drop in un-availability.

8.2 The end-user tolerance of the system response latency

In the face of the computer pervasiveness, the human-computer interaction becomes one of the factors determining the end-user perception of system dependability. Response latency is one of the

important quality of service (QoS) metrics. An early study by Miller? [48] indicated that: (1) response within 0.1s is perceived by the user as an instantaneous reaction of the system, (2) response within 1.0s keeps the user attention to an interactive dialog, and (3) 10 seconds delay is about the limit for holding the user attention to the task at hand. For longer delays the user is distracted and may move to another task. Similar conclusions are drawn from the study on a cognitive coprocessor (a user interaction manager)^[11]. More recent work conducted in HP Laboratories on the user-perceived latency of Web-based services reinforces these findings and indicates that delays of around 11s represent the threshold beyond which it is difficult to keep users attention to the task^[50]. For designers of dependable systems these findings show the importance of fast error detection and rapid recovery in minimizing the system downtime due to errors and, hence, reducing the response latency perceived by the end-user.

8.3 Pervasive computing

For the past decade computer science researchers have been defining new services and architectures for pervasive and ubiquitous computing environments^[51]. Pervasive/ubiquitous computing environments encompass hundreds of embedded computers throughout the physical environment that can provide services such as displays and printing. As mobile users and devices move through the environment, services must be discovered and protocols established for communicating and combining services. Basic research in pervasive/ubiquitous computing may provide some of the mechanisms for "on the fly" monitoring and reconfiguration.

8.4 Cognitive assistants that learn

The DARPA PAL (Personal Assistant that Learns) program's goal, is to create cognitive systems that use a variety of learning techniques to discover user preferences and proactively anticipate user needs. By communicating through cognitive agents, the user that has discovered a technique to work around a problem could have that information shared with other users. Thus, rather than having a dedicated, skilled staff to help users work through problems (a recent study of remoter user authentication problems indicated an average problem resolution time of 72h^[36]) the entire user community could share what has been discovered.

8.5 Proactive management

The number of operational parameters in a computing system has grown explosively. Compa-

nies have been founded whose sole product is to discover network configurations and set the parameter values of the various switches to optimize performance. One can envision such services reaching down to the individual user and their mobile device. Early research in artificial intelligence led to the R1 system^[52] for configuring VAX high-end computers for Digital Equipment Corporation (DEC). Since there were a large number of possible configurations, R1 examined the purchased order and created a bill of materials adding in missing components that were implied by the rest of the configuration. Research in such technology could carry beyond the manufacturing phase into the operational life phase.

8.6 Complexity—Artifact

Historically, the computer industry has used abstractions as a means to handle complexity and allow independent development on opposite sides of the boundary. One of the first highly successful abstractions was the separation of instruction set design from implementation. In the early 1960's, IBM defined the IBM System 360 instruction set architecture. Over the next four decades hardware designers created tens of different hardware implementations increasing performance a thousand fold but guaranteeing that even the software written in the 1960's would run unmodified. Meanwhile, software designers could write new applications without worrying about the moving target of hardware technology.

The contemporary embodiment of abstractions is industrial standards. Standards define functionality that service providers can implement while service consumers can use without concern about the implementation of the service. A research opportunity is to apply previously successful concepts to this new level abstraction.

For example, the Ballista approach to probing the exception handling capabilities at the API (Application Program Interface) could possibly be extended and adapted to standards definitions.

As another example, the mnemonic reminder approach to minimizing the generation of design errors might also be extended to both the providers and consumers of standards. For example, the CHILDREN mnemonic^[52-53] has been demonstrated to decrease the number of errors due to common software programmer omissions and commissions. The mnemonic aids recall of issues to consider while writing code.

9 Conclusions

Our framework, defined by three elements —

trends, artifacts, and processes — Has allowed us to reflect on overarching industry trends and technologies that impact the industry's artifacts and/or processes. Looking back at Table 2, following our discussions, lets us reflect on where we have been and where opportunities for research lurk.

Trend 1 Shifting Error Sources and Trend 2 Explosive Complexity are well underway with a substantial body of research. Nevertheless, there remains a need for more research, especially on issues of complexity and security (e.g., measurement-based analysis of system security). Trend 3 Global Volume, is upon us but is young in terms of research. We have articulated several topics that are currently visible and certainly others will reveal themselves. The pace of industry and its needs for research has grown by the sheer volume of business and domains of application. Thus, the need for this research is imminent.

Acknowledgements This paper is based upon reference [54]. This work was completed under a grant from the Office of Naval Research, Interoperability of Future Information Systems through Context-and Model-based Adaptation (#N00014-02-1-0499).

References

- [1] Sellers F, Hsiao M, Bearson L. Error Detecting Logic for Digital Computers. New York: McGraw-Hill Book Co., 1968
- [2] Avizienis A. Design of fault-tolerant computers//Proceedings of the Fall Joint Computer Conference. Washington D. C.: Thompson Books, 1967, 31: 733-743
- [3] Hsiao M et al. Reliability, availability, and serviceability of IBM computer systems: A quarter century of progress. IBM Journal Research and Development, 1981, 25(5): 453-465
- [4] Normand E. Single event upset at ground level. IEEE Transactions on Nuclear Science, 1996, 43:
- [5] Ziegler J et al. IBM's experiments in soft fails in computer. IBM Journal of Research and Development, 1996, 40(1): 3-18
- [6] Faccio F et al. Single event effects in static and dynamic registers in a 0.25 μ m CMOS technology. IEEE Transactions on Nuclear Science, 1999, 46(6): 1434-1439
- [7] Spainhower L, Gregg T. IBM S/390 parallel enterprise server G5 fault tolerance: A historical perspective. IBM Journal of Research and Development, 1999, 43(5/6): 863-873
- [8] Spainhower L et al. IBM's ES/9000 model 982's fault-tolerant design for consolidation. IEEE Micro, 1994, 14(1): 48-59
- [9] Intel Corp. Intel's Quality System Databook. January 1998, No. 210997-007
- [10] Arlat J et al. Fault injection for dependability validation: A methodology and some applications. IEEE Transactions on Software Engineering, 1990, 16(2): 166-182
- [11] Card S, Robertson G, Mackinlay J. The information visualizer: An information workspace//Proceedings of the ACM CHI' 91 Conference. New Orleans, Louisiana, United States, 1991: 181-186
- [12] Avizienis A, Laprie J-C, Randell B. Fundamental concepts of dependability//Proceedings of the 3rd Information Survivability Workshop, 2000 (Also updated as UCLA CSD Report No. 010028, LAAS Report No. 01-145, Newcastle University Report No. CS-TR-739, 2001)
- [13] Laprie J-C. Dependable computing and fault tolerance: Concepts and terminology//Proceedings of the 15th International Symposium on Fault-Tolerant Computing (FTCS-15). Ann Arbor, Michigan, 1985: 2-11
- [14] Avizienis A. The N-version approach to fault-tolerant software. IEEE Transactions on Software Engineering, 1985, 11(12): 1491-1501
- [15] Siewiorek D, Maxion R, Narasimhan P. Experimental research in dependable computing at carnegie mellow university//Proceedings of the IFIP 18th World Computer Congress. Toulouse, France 2004: 305-328
- [16] Feather F, Siewiorek D P, Segall Z. Validation of a fault-tolerant multiprocessor: Baseline experiments and workload implementation. Carnegie Mellon University: Technical Report CMU-CS-85-145, 1985
- [17] Chillarege R, Bowen N S. Understanding large system failures—A fault injection experiment//Proceedings of the 19th International Symposium on Fault-Tolerant Computing (FTCS-19). Chicago, IL, USA, 1989: 356-363
- [18] Gunneflo U, Karlsson J, Torin J. Evaluation of error detection schemes using fault injection by heavy-ion radiation//Proceedings of the International Symposium on Fault-Tolerant Computing (FTCS-19). Chicago, IL, USA, 1989: 340-347
- [19] Rodriguez M et al. MAFALDA: Microkernel assessment by fault injection and design aid//Proceedings of the 3rd European Dependable Computing Conference (EDCC-3). Prague, Czech Republic, 1999: 145-160
- [20] Merenda A, Merenda E. Recovery/Serviceability/System test improvements for the IBM ES/9000 520 based models//Proceedings of the International Symposium on Fault-Tolerant Computing (FTCS-22). Boston, MA, USA, 1992: 463-467
- [21] Amendola A et al. Experimental evaluation of computer-based railway control systems//Proceedings of the International Conference on Fault-Tolerant Computing Systems (FTCS-27). Seattle, WA, USA, 1997: 380-384
- [22] Constantinescu C. Validation of the fault/error handling mechanisms of the teraflops supercomputer//Proceedings of the International Symposium on Fault-Tolerant Computing (FTCS-28). Munich, Germany, 1998: 382-389
- [23] Vardanega T et al. On the development of fault-tolerant on-board control software and its evaluation by fault injection//Proceedings of the International Symposium on Fault-Tolerant Computing (FTCS-25). Pasadena, CA, USA, 1995: 510-515

- [24] Koopman P, DeVale J. The exception handling effectiveness of POSIX operating systems. *IEEE Transactions on Software Engineering*, 2000, 26(9): 837-848
- [25] Anderson T et al. Protective wrapper development: A case study//*Proceedings of the 2nd International Conference on OTS-Based Software Systems (ICCBSS)*. Ottawa, Canada, 2003; 1-14
- [26] DeVale J, Koopman P. Robust software — No more excuses//*Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)*. Bethesda, MD, USA, 2002; 145-154
- [27] Rodriguez M, Fabre J-C, Arlat J. Wrapping real-time systems from temporal logic specifications//*Proceedings of the 4th European Dependable Computing Conference (EDCC-3)*. Toulouse, France, 1999; 253-270
- [28] Hsueh Mei-Chen, Tsai T K, Iyer R K. Fault injection techniques and tools. *Computer*, 1997, 30(4): 75-82
- [29] Carreira J, Madeira H, Silva J G. Xception: Software fault injection and monitoring in processor functional units//*Proceedings of the 5th Annual IEEE International Working Conference Dependable Computing for Critical Applications*. IEEE CS Press, Los Alamitos, California, 1995; 135-149
- [30] Dong Jian, Qu Feng, Liu Hong-Wei, Cui Gang. Mini-micro Systems, 2003, 24(12): 2335-2337
- [31] Laprie J-C et al. Dependability: Basic concepts and terminology//*Dependable Computing and Fault-Tolerant Systems*. New York: Springer-Verlag, 1992
- [32] Chillarege R. What is software Failure? *IEEE Transactions on Reliability*, 1996, 45(3): 354-355
- [33] Chillarege R, Biyani S et al. Measurement of failure rate in widely distributed software//*Proceedings of the International Symposium on Fault-Tolerant Computing Systems (FTCS-25)*. Pasadena, CA, USA, 1995; 424-433
- [34] Gray J. A census of tandem system availability between 1985 and 1990. *IEEE Transactions on Reliability*, 1990, 39(4): 409-418
- [35] Maxion R. Distributed diagnostic performance reporting and analysis//*Proceedings of the IEEE International Conference on Computer Design*. Port Chester, New York, 1986; 362-365
- [36] Steinfeld A et al. An examination of remote access help desk cases. School of Computer Science, Carnegie Mellon University; Technical Report CMU-CS-03-190, CMU-HCII-03-100, 2003
- [37] Clarke E, Emerson E. Synthesis of synchronization skeletons for branching time temporal logic//*Proceedings of the Logic of Programs; Workshop*. Lecture Notes in Computer Science 131. Yorktown Heights, NY, 1981; 52-71
- [38] Holzmann G. The model checker SPIN. *IEEE Transactions on Software Engineering*, 1997, 23(5): 279-295
- [39] Beizer B. *Software Testing Techniques*. New York: Van Nostrand Reinhold, 1990
- [40] Phadke M. *Quality Engineering using Robust Design*. New Jersey: Prentice Hall, 1989
- [41] Cohen D M et al. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 1997, 23(7): 437-444
- [42] Harel D, Politi M. *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*. New York: McGraw-Hill, 1998
- [43] Wickens C D. *Aerospace techniques*//Weimer J ed. *Research Techniques in Human Engineering*. Englewood Cliffs, NJ: Prentice Hall PTR, 1995; Seecof M. AT&T "Deeply Distressed" over outage. *The Risks Digest*, 1991, 12(43): 112-142
- [44] Schmidt C, Darby T. The morris internet worm, 2001. <http://snowplow.org/tom/worm/worm.html>
- [45] CERT Advisory Committee. Code Red worm exploiting buffer overflow in IIS indexing service DLL. January 2002, CERT Advisory CA-2001-19
- [46] Pattabiraman K, Kalbarczyk Z, Iyer R. Application-based metrics for strategic placement of detectors//*Proceedings of the Pacific Rim Dependability Conference*. Changsha, Huan, China, 2005; 75-82
- [47] Nakka N, Kalbarczyk Z, Iyer R, Xu J. An architectural framework for providing reliability and security support//*Proceedings of the International Conference on Dependable Systems and Networks (DSN-04)*. Florence, Italy, 2004; 585-594
- [48] Maxion R, Tan K. Anomaly detection in embedded systems. *IEEE Transactions on Computers*, 2002, 51(2): 108-120
- [49] Dorron L, Chillarege R. Early warning of failures through alarm analysis — A case study in telcom voice mail systems//*Proceedings of the International Symposium on Software Reliability Engineering*. Denver, USA, 2003; 271-280
- [50] Bhatti N, Bouch A, Kuchinsky A. Integrating user-perceived quality into Web server design//*Proceedings of the 9th International WWW Conference*. Amsterdam, The Netherlands, 2000; 1-16
- [51] *IEEE Pervasive Computing Magazine*, 2004, 1(2)
- [52] Maxion R, Olszewski R. Eliminating exception handling errors with dependability cases: A comparative, empirical study. *IEEE Transactions on Software Engineering*, 2000, 26(9): 888-906
- [53] Maxion R, Olszewski R. Improving software robustness with dependability cases//*Proceedings of the International Symposium on Fault-Tolerant Computing (FTCS-28)*. Munich, Germany, 1998; 346-355
- [54] Chillarege R, Siewiorek D, Kalbarczyk Z. Reflections on industry trends and experimental research in dependability. *IEEE Transactions on Dependable and Secure Computing*, 2004, 1(2): 109-127
- [55] Avizienis A. A fault tolerance infrastructure for dependable computing with high performance COTS components//*Proceedings of the International Conference on Dependable Systems and Networks (DSN'00)*. New York, NY, USA, 2000; 492-500
- [56] Chillarege R. The marriage of business dynamics and software. *IEEE Software*, 2002, 19(6): 43-49
- [57] Killourhy K S, Maxion R, Tan K M C. A defense-centric taxonomy based on attack manifestations//*Proceedings of the International Conference on Dependable Systems and Networks*. Florence, Italy, 2004; 102-111
- [58] Miller R. Response time in man-computer conversational transactions//*Proceedings of the AFIPS Fall Joint Computer Conference*. Montvale, NJ, 1968, 33; 267-277

- [59] McDermott J. R1: A rulebased configurator of computer system. *Artificial Intelligence*, 1982, 19(1): 39-88
- [60] Spainhower L, Gregg T. G4: A fault-tolerant CMOS main-frame//*Proceedings of the International Symposium on Fault Tolerant Computing (FTCS-28)*. Munich, Germany, 1998: 432-440
- [61] Wood A. Software reliability from the customer view. *IEEE Computer*, 2003, 36(8): 37-42
- [62] Graham J, Hart B. Knowledgebase integration with a 24-hour help desk//*Proceedings of the 28th Annual ACM SIGUCCS Conference on User Services*. 2000: 92-95

- [63] Kuo J, Burns C M. A work domain analysis for virtual private networks//*Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 2000, 3: 1972-1977
- [64] Meseguer J, Agha G, Gunter C, Roman G-C, Venkatasubramanian N. MURI project CONTESSA on adaptive system interoperability. <http://formal.cs.uiuc.edu/contezza/>
- [65] Salvendy G, Carayon P. Data collection and evaluation of outcome measures//Salvendy G ed. *Handbook of Human Factors and Ergonomics*. New York, NY: Wiley-Interscience, 1997



SIEWIOREK Daniel P., Professor

Daniel P. Siewiorek is the Buhl University Professor of Electrical and Computer Engineering and Computer Science at Carnegie Mellon University. He has designed or been involved with the design of nine multiprocessor systems and has been a key contributor to the dependability design of over two dozen commercial computing systems. Dr. Siewiorek leads an interdisciplinary team that has designed and constructed over 20 generations of mobile computing systems. Dr. Siewiorek has written eight textbooks in the areas of parallel processing, computer architecture, reliable computing, and design automation in addition to over 475 papers Dr. Siewiorek has served as Associate Editor of the *Computer System Department of the Communications of the Association for Computing Machinery*, as Chairman of the *IEEE Technical Committee on Fault-Tolerant Computing* and as founding Chairman of the *IEEE Technical Committee on Wearable Information Systems*. Currently Director of the *Human Computer Interaction Institute*, he was previously Director of the *Engineering Design Research Center* and co-founder of its successor organization, the *Institute for Complex Engineered Systems*, where he served as Associate Director. He has been the recipient of the *American Association of Engineering Education Frederick Emmons Terman Award*, the *IEEE/ACM Eckert-Mauchly Award*, and the *ACM SIGMOBILE Outstanding Contributions Award*. He is a Fellow of IEEE, ACM, and AAAS and is a member of the *National Academy of Engineering*.

Professor Siewiorek received the B. S. degree in Electrical Engineering from the University of Michigan, Ann Arbor, in 1968, and the M. S. and Ph. D. degrees in Electrical Engineering (minor in Computer Science) from Stanford University, in 1969 and 1972, respectively.

Professor Siewiorek received the B. S. degree in Electrical Engineering from the University of Michigan, Ann Arbor, in 1968, and the M. S. and Ph. D. degrees in Electrical Engineering (minor in Computer Science) from Stanford University, in 1969 and 1972, respectively.

YANG Xiao-Zong, born in 1939, professor and Ph. D. supervisor, a senior member of CCF. His research interests include dependable computing and mobile computing.

CHILLAREGE Ram received the Ph. D. degree from the University of Illinois, Urbana-Champaign in computer engineering, the ME and BE degrees from the Indian Institute of Science, Bangalore, and the BSc degree from the University

of Mysore. He is a management consultant in software engineering optimization. He had been the Executive Vice President of Software and Technology at Opus360, and prior to that was with IBM Research for 14 years, where he founded and headed the Center for Software Engineering. He received the IEEE Technical Achievement for inventing Orthogonal Defect Classification (ODC) and an expansive body of work, which ushers new methods to the business of managing software. In the mid 1990s, he led the effort, which culminated in IBM establishing a corporate-wide software testing practice initiative. He has authored around 50 peer reviewed articles and is an active speaker at conferences. He serves on the IEEE steering committees of Software Reliability and Dependability symposiums, several conference committees, and the US National Science Foundation panels.

KALBARCZYK Zbigniew T. is a research professor at the Center for Reliable and High-Performance Computing in the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign. He received the Ph. D. degree in computer science from the Technical University of Sofia, Bulgaria. After receiving his doctorate, he worked as an assistant professor in the Laboratory for Dependable Computing at Chalmers University of Technology in Gothenburg, Sweden. His research interests include automated design, implementation, and evaluation of dependable and secure computing systems. Currently, he is a lead researcher on the project to explore and develop high availability and security infrastructure capable of managing redundant resources across interconnected nodes, to foil security threats, detect errors in both the user applications and the infrastructure components, and recover quickly from failures when they occur. His research involves also developing of techniques for validation and benchmarking of dependable computing systems. He served as Program Chair of the Dependable Computing and Communication Symposium (DCCS) track on the International Conference on Dependable Systems and Networks (DSN), 2007 and as program co-chair of the Performance and Dependability Symposium (PDS), a track of DSN 2002; and is regularly invited to work on the program committees of major conferences on design of fault-tolerant systems. He is a member of the IEEE and the IEEE Computer Society.